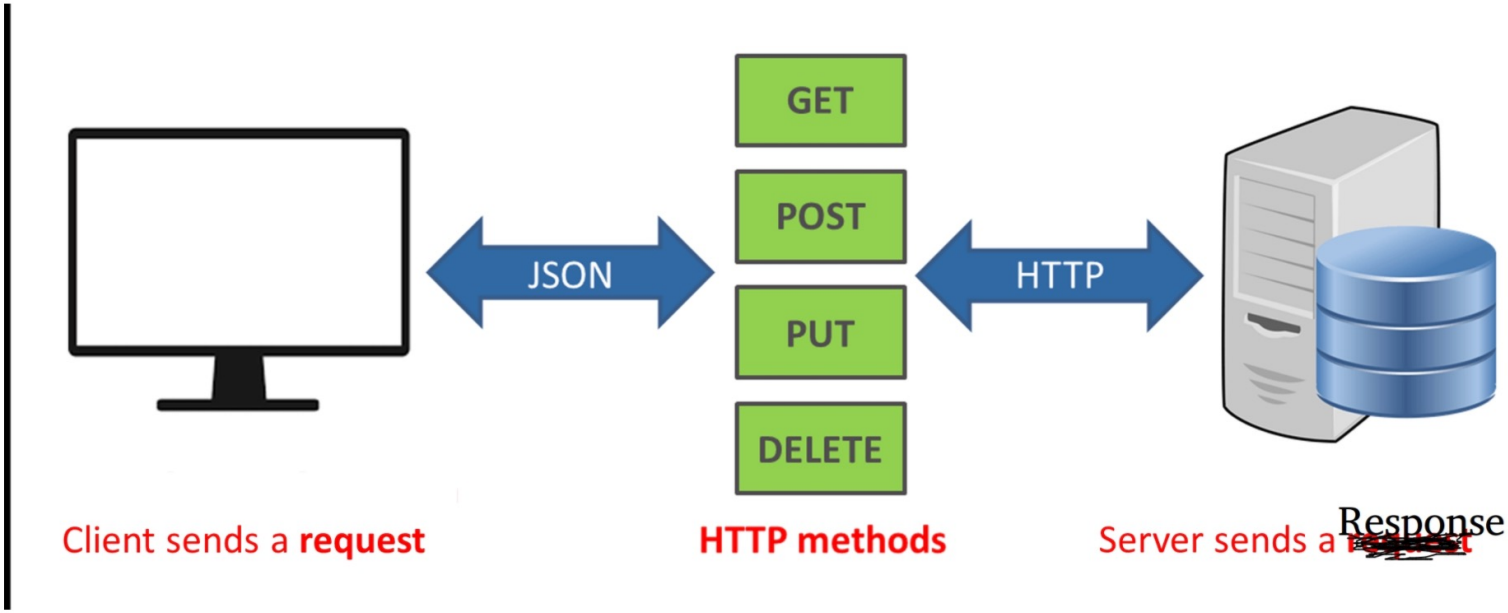
Software Defined Network
SDN

Ch 4

Ryo Controller – Part II

REST API Introduction

A RESTful API is an application program interface (API) that uses HTTP requests to GET, PUT, POST and DELETE data



REST API Introduction

HTTP

HTTP is INTERNET based protocol. It allows computers from anywhere in the world to send requests to remote servers, and get back responses that can be displayed in browsers.

For example, to create a new article on our blog, we should send a request to a remote server using the POST HTTP method. To view a single article or a list of articles, we use the GET method. The PUT method can be used to edit an existing article, and the DELETE method to delete.

REST API Introduction

Methods

The following table summarizes the 4 most useful HTTP protocol methods:

GET

Retrieves data from a remote server. It can be a single resource or a list of resources.

POST

Creates a new resource on the remote server.

PUT

Updates the data on the remote server.

DELETE

Deletes data from the remote server.

REST API Introduction

URLs

The requests should be sent to URLs on the remote server.

REST API Endpoints:

POST /api/articles

GET /api/article/1

PUT /api/article/1

DELETE /api/article/1

URL:

http[s]://Server:portnumber/api-endpoint

Example:

http://localhost:8080/api/article/1

REST API Introduction

HTTP Header & Payload & Return Status code:

HTTP Header

Below are important payload , which we use frequently

Authorization Header:

Authorization: Basic xxxxxxxxx

Type: Basic, OAuth, Token, etc.

Data type Header:

Content-Type : application/xml / application/json

Accept : application/xml / application/json

REST API Introduction

Request & Response Data:

Request Data:

Client sends the Data to the server. This is called as Request Body. POST, PUT method should require the data to be submitted to the Server. GET , Delete Method - doesnot

Response Data:

Server responds with the data(result). This is usually called as response data.

All the HTTP Methods operations can respond the data. GET must respond the data.

POST,DELETE,PUT may/maynot respond with data.

Return Status Code:

2XX (200, 201, 204) indicates that the data was received and the operation was performed.

4XX, 5XX Indicates Error

REST Clients

we require REST Client to perform the REST API operations.

Example:

- POSTMAN REST Client
- CURL

....

REST Clients

CURL

CURL is a CLI based REST Utility(multipurpose). Its a small binary.

Installation

```
sudo apt-get install curl
sudo apt-get install jq
```

jq is a tool for json pretty print.

REST Clients

How to use

curl has lot of options(flags) for various features/funtions. we see the minimal options required for us.

syntax

> *curl -u username:password -H "header1" -H "header2" -X HTTP-METHOD URL -d '@input-datafilename'*

Example

```
curl --user "admin":"admin" -H "Content-type: application/json" -X POST
http://localhost:8181/api/v1/create -d '@data.json'
```

REST Clients

Demo

We use the http://dummy.restapiexample.com/ REST API Server for testing.

GET Method:

```
curl -H "Accpet: application/json" -X GET  http://dummy.restapiexample.com/api/v1/employees

curl -H "Accpet: application/json" -X GET  http://dummy.restapiexample.com/api/v1/employees  | jq .
```

POST Method

Data file

```
$cat data.json
{
    "name":"suresh kumar",
    "salary":"123",
    "age":"23"
}
```

> *curl -H "Content-type: application/json" -H "Accept: application/json" -X POST http://dummy.restapiexample.com/api/v1/create -d '@data.json'*

REST Clients

Example

```
$ curl -H "Content-type: application/json" -H "Accept: application/json" -X POST
http://dummy.restapiexample.com/api/v1/create -d '@data.json' | jq .
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   115    0    62  100    53     91     78 --:--:-- --:--:-- --:--:--    91
{
  "name": "suresh kumar",
  "salary": "123",
  "age": "23",
  "id": "1288"
}
```

REST Clients

PUT Method:

cat data.json

```
{"name":"suresh kumar","salary":"123","age":"89999"}
```

> *curl -H "Content-type: application/json" -H "Accept: application/json" -X PUT http://dummy.restapiexample.com/api/v1/update/1288 -d '@data.json'*

DELETE Method:

```
curl -H "Accpet: application/json" -X DELETE http://dummy.restapiexample.com/api/v1/delete/1288
```

ในส่วนของการใช้ POSTMAN ลองเข้าเว็ป https://www.youtube.com/watch?v=Y9oVAkTXbzU&ab_channel=PatiphanPhengpao

RYU OFCTL REST API application

1. Introduction

OFCTL - Openflow control using REST Interface.

ryu.app.ofctl_rest application provides REST APIs for configure/update/retrive the switch flows /stats etc. This application helps you debug your application and get various statistics.

This application supports OpenFlow version 1.0, 1.2, 1.3, 1.4 and 1.5.

This application is not switch application.

ใช้สำหรับอ้างอิงเพิ่มเติม https://ryu.readthedocs.io/en/latest/app/ofctl_rest.html

RYU OFCTL REST API application

2.Example - Get Statistics

Objective

Retrive the Switch Details/Stats,

Steps

1. Run the mininet simple topology

> *sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=linear,4*

2. Run the RYU OFCTL REST & Simple Switch application

> *ryu-manager ryu.app.simple_switch_13 ryu.app.ofctl_rest*

3. do pingall from mininet. and check the ovs flows
4. Get All switches

```
suresh@suresh-vm:~$  curl -X GET http://localhost:8080/stats/switches
[1, 2, 3, 4]
```

RYU OFCTL REST API application

5. Get the desc stats of the switch which specified with Datapath ID in URI.

```
suresh@suresh-vm:~$ curl -X GET http://localhost:8080/stats/desc/1
{"1": {"dp_desc": "None", "sw_desc": "2.9.2", "hw_desc": "Open vSwitch", "serial_num": "None",
"mfr_desc": "Nicira, Inc."}}
```

6. Get all flows stats of the switch which specified with Datapath ID in URI.

```
suresh@suresh-vm:~$ curl -X GET http://localhost:8080/stats/flow/1
{"1": [{"actions": ["OUTPUT:1"], "idle_timeout": 0, "cookie": 0, "packet_count": 3, "hard_timeout": 0,
"byte_count": 238, "duration_sec": 213, "duration_nsec": 654000000, "priority": 1, "length": 104,
"flags": 0, "table_id": 0, "match": {"dl_dst": "00:00:00:00:00:01", "dl_src": "00:00:00:00:00:02",
"in_port": 2}}, {"actions": ["OUTPUT:2"], "idle_timeout": 0, "cookie": 0, "packet_count": 2,
"hard_timeout": 0, "byte_count": 140, "duration_sec": 213, "duration_nsec": 653000000, "priority": 1,
"length": 104, "flags": 0, "table_id": 0, "match": {"dl_dst": "00:00:00:00:00:02", "dl_src":
"00:00:00:00:00:01", "in_port": 1}}, {"actions": ["OUTPUT:1"], "idle_timeout": 0, "cookie": 0,
"packet_count": 3, "hard_timeout": 0, "byte_count": 238, "duration_sec": 213, "duration_nsec":
634000000, "priority": 1, "length": 104, "flags": 0, "table_id": 0, "match": {"dl_dst":
"00:00:00:00:00:01", "dl_src": "00:00:00:00:00:03", "in_port": 2}}, {"actions": ["OUTPUT:2"],
"idle_timeout": 0, "cookie": 0, "packet_count": 2, "hard_timeout": 0, "byte_count": 140,
"duration_sec": 213, "duration_nsec": 632000000, "priority": 1, "length": 104, "flags": 0, "table_id":
0, "match": {"dl_dst": "00:00:00:00:00:03", "dl_src": "00:00:00:00:00:01", "in_port": 1}}, {"actions":
["OUTPUT:1"], "idle_timeout": 0, "cookie": 0, "packet_count": 3, "hard_timeout": 0, "byte_count": 238,
"duration_sec": 213, "duration_nsec": 614000000, "priority": 1, "length": 104, "flags": 0, "table_id":
0, "match": {"dl_dst": "00:00:00:00:00:01", "dl_src": "00:00:00:00:00:04", "in_port": 2}}, {"actions":
["OUTPUT:2"], "idle_timeout": 0, "cookie": 0, "packet_count": 2, "hard_timeout": 0, "byte_count": 140,
"duration_sec": 213, "duration_nsec": 612000000, "priority": 1, "length": 104, "flags": 0, "table_id":
0, "match": {"dl_dst": "00:00:00:00:00:04", "dl_src": "00:00:00:00:00:01", "in_port": 1}}, {"actions":
["OUTPUT:CONTROLLER"], "idle_timeout": 0, "cookie": 0, "packet_count": 129, "hard_timeout": 0,
"byte_count": 15332, "duration_sec": 215, "duration_nsec": 993000000, "priority": 0, "length": 80,
"flags": 0, "table_id": 0, "match": {}}]}
```

RYU OFCTL REST API application

7. Get Port stats

```
suresh@suresh-vm:~$ curl -X GET http://localhost:8080/stats/port/1
{"1": [{"tx_dropped": 0, "rx_packets": 0, "rx_crc_err": 0, "tx_bytes": 0, "rx_dropped": 131, "port_no":
"LOCAL", "rx_over_err": 0, "rx_frame_err": 0, "rx_bytes": 0, "tx_errors": 0, "duration_nsec":
436000000, "collisions": 0, "duration_sec": 287, "rx_errors": 0, "tx_packets": 0}, {"tx_dropped": 0,
"rx_packets": 25, "rx_crc_err": 0, "tx_bytes": 20026, "rx_dropped": 0, "port_no": 1, "rx_over_err": 0,
"rx_frame_err": 0, "rx_bytes": 1906, "tx_errors": 0, "duration_nsec": 449000000, "collisions": 0,
"duration_sec": 287, "rx_errors": 0, "tx_packets": 159}, {"tx_dropped": 0, "rx_packets": 131,
"rx_crc_err": 0, "tx_bytes": 5923, "rx_dropped": 0, "port_no": 2, "rx_over_err": 0, "rx_frame_err": 0,
"rx_bytes": 15895, "tx_errors": 0, "duration_nsec": 449000000, "collisions": 0, "duration_sec": 287,
"rx_errors": 0, "tx_packets": 52}]}
```

RYU OFCTL REST API application

3. Example - Hub

Objective

Enable the Openflow switch will act as HUB.

Steps

1. Run the mininet simple topology

> *sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,4*

2. Run the RYU OFCTL REST Api application

> *ryu-manager ryu.app.ofctl_rest*

RYU OFCTL REST API application

3. do pingall from mininet. and check the ovs flows

Flow table will be empty and ping fails.

4. Add a "Flood" flow using REST API.

cat hub_flow.json

```
{
    "dpid": 1,
    "table_id": 0,
    "idle_timeout": 0,
    "hard_timeout": 0,
    "priority": 100,
    "match":{
    },
    "actions":[
        {
            "type":"OUTPUT",
            "port": 4294967291
        }
    ]
}
```

Here Port Number 4294967291 (0xFFFFFFFB) means OFP_FLOOD Port.

RYU OFCTL REST API application

API command:

> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@hub_flow.json'*

5.check the ovs flows

```
suresh@suresh-vm:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=3.536s, table=0, n_packets=0, n_bytes=0, idle_timeout=0, hard_timeout=0,
priority=100 actions=FLOOD
suresh@suresh-vm:~$
```

6.Now, perform ping between hosts in mininet

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1.11 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.189 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.153 ms
^C
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.153/0.485/1.113/0.444 ms
mininet>
```

RYU OFCTL REST API application

4. Example - Switch

Objective

Enable the Openflow switch will act as Layer2 Mac based Switch.

Steps

1. Use the same procedure as Example1 for running mininet and ryu ofctl application.
2. Add the Broadcast flow(for L2 Broadcast/ARP packet)

- flow1: dst-mac(ff:ff:ff:ff:ff:ff) - output FLOOD

> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@switch_arp.json'*

RYU OFCTL REST API application

3. Add the destination mac specific flows

- flow1: dst-mac(00:00:00:00:00:01) - output port 1
- flow2: dst-mac(00:00:00:00:00:02) - output port 2
- flow3: dst-mac(00:00:00:00:00:03) - output port 3
- flow4: dst-mac(00:00:00:00:00:04) - output port 4

> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@switch_flow1.json'*
>
> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@switch_flow2.json'*
>
> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@switch_flow3.json'*
>
> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@switch_flow4.json'*

RYU OFCTL REST API application

4. Check the flows and do pingall from mininet

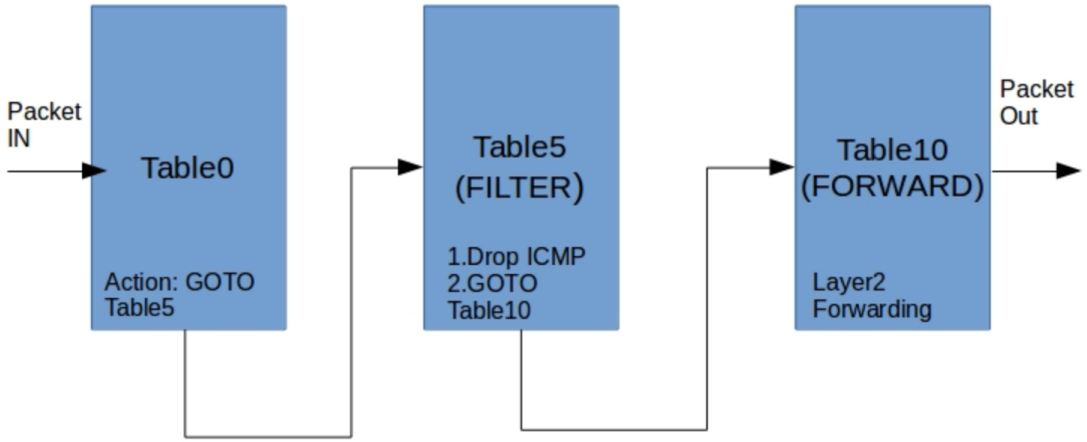
```
suresh@suresh-vm:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=307.210s, table=0, n_packets=26, n_bytes=2212,
priority=100,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth2"
 cookie=0x0, duration=304.663s, table=0, n_packets=10, n_bytes=756,
priority=100,dl_dst=00:00:00:00:00:03 actions=output:"s1-eth3"
 cookie=0x0, duration=300.185s, table=0, n_packets=27, n_bytes=2254,
priority=100,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=260.081s, table=0, n_packets=9, n_bytes=714,
priority=100,dl_dst=00:00:00:00:00:04 actions=output:"s1-eth4"
 cookie=0x0, duration=39.302s, table=0, n_packets=6, n_bytes=252, priority=100,dl_dst=ff:ff:ff:ff:ff:ff
actions=FLOOD
suresh@suresh-vm:~$
```

RYU OFCTL REST API application

5. Example - Multitable

Objective

Demonstrate the Multitables(pipeline processing) concepts, and ACL Rules(Drop).



RYU OFCTL REST API application

We will create two tables

- FILTER TABLE(Table 5) and FORWARD TABLE(Table 10).
- FILTER TABLE : will add packet filter rules (ACL rules). Block ICMP traffic for h4.
- FORWARD TABLE: our forwarding rules (L2 switch)

High level Steps:

1. In Table 0(default), Add the Match all Flow with action GO TO - Filter Table.
2. In Table 5(Filter Table), Add the filter rules(icmp block for h4) in higher priority, Add the Match All Flow with the Action GO TO - Forward Table
3. In Table 10(Forward Table), Add the L2 forwarding rules.

RYU OFCTL REST API application

Steps

1. Use the same procedure as Example1 for running mininet and ryu ofctl application.
2. Add the flows as below,

- Add flow in table0, forward all traffic in it to table 5
- Add the flow in table5, to go to table 10
- Add the forwarding flows in table10

> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@table0_flow1.json'*
>
> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@table5_flow1.json'*
>
> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@table10_flow1.json'*
>
> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@table10_flow2.json'*
>
> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@table10_flow3.json'*
>
> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@table10_flow4.json'*
>
> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@table10_arp.json'*

RYU OFCTL REST API application

3. Check the flows and do pingall in mininet.

```
suresh@suresh-vm:~/1$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=163.683s, table=0, n_packets=99, n_bytes=8274, priority=0 actions=goto_table:5
 cookie=0x0, duration=86.453s, table=5, n_packets=97, n_bytes=8134, priority=0 actions=goto_table:10
 cookie=0x0, duration=72.661s, table=10, n_packets=24, n_bytes=2016,
priority=0,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=69.423s, table=10, n_packets=23, n_bytes=1974,
priority=0,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth2"
 cookie=0x0, duration=66.424s, table=10, n_packets=22, n_bytes=1932,
priority=0,dl_dst=00:00:00:00:00:03 actions=output:"s1-eth3"
 cookie=0x0, duration=62.997s, table=10, n_packets=21, n_bytes=1890,
priority=0,dl_dst=00:00:00:00:00:04 actions=output:"s1-eth4"
 cookie=0x0, duration=33.654s, table=10, n_packets=6, n_bytes=252, priority=0,dl_dst=ff:ff:ff:ff:ff:ff
actions=FLOOD
suresh@suresh-vm:~/1$
```

RYU OFCTL REST API application

4. Add a ICMP Drop flow for h4

> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@table5_flow2.json'*

```
suresh@suresh-vm:~/1$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=472.976s, table=0, n_packets=99, n_bytes=8274, priority=0 actions=goto_table:5
 cookie=0x0, duration=3.663s, table=5, n_packets=0, n_bytes=0, priority=100,icmp,nw_dst=10.0.0.4
actions=drop
 cookie=0x0, duration=395.746s, table=5, n_packets=97, n_bytes=8134, priority=0 actions=goto_table:10
 cookie=0x0, duration=381.954s, table=10, n_packets=24, n_bytes=2016,
priority=0,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=378.716s, table=10, n_packets=23, n_bytes=1974,
priority=0,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth2"
 cookie=0x0, duration=375.717s, table=10, n_packets=22, n_bytes=1932,
priority=0,dl_dst=00:00:00:00:00:03 actions=output:"s1-eth3"
 cookie=0x0, duration=372.290s, table=10, n_packets=21, n_bytes=1890,
priority=0,dl_dst=00:00:00:00:00:04 actions=output:"s1-eth4"
 cookie=0x0, duration=342.947s, table=10, n_packets=6, n_bytes=252, priority=0,dl_dst=ff:ff:ff:ff:ff:ff
actions=FLOOD
suresh@suresh-vm:~/1$
```

Now h4 ping traffic will be blocked in FILTER table.

Try pingall from mininet prompt.

Group Table

1. Introduction

The ability for a flow entry to point to a group enables OpenFlow to represent additional methods of forwarding (e.g. select and all).

OpenFlow groups were introduced in OpenFlow 1.1 as a way to perform more complex operations on packets that cannot be defined within a flow alone.

There are different types of groups defined in the OpenFlow specification to serve a variety of purposes

Example usecases :

1) Sniffing/Port mirroring

2) Load balancing

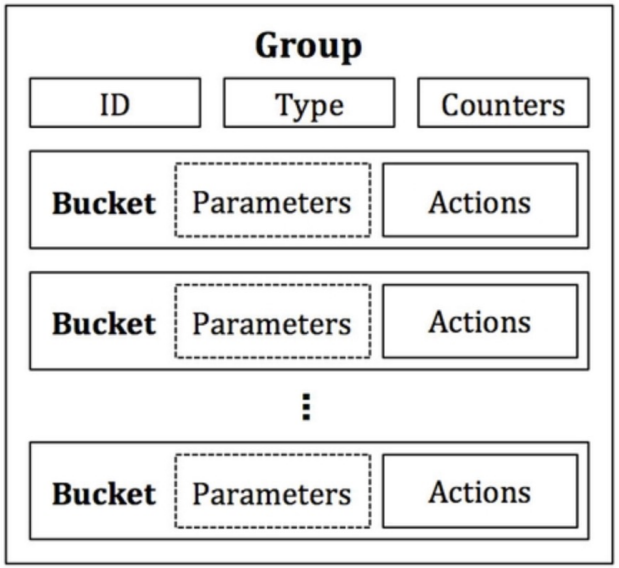
3) fast-failovers to alternative links

ใช้สำหรับอ้างอิงเพิ่มเติม:

https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/7995427/How+to+Work+with+Fast-Failover+OpenFlow+Groups

Group Table

Group table Components



1. Group contains a bucket list
2. Bucket - it contains separate lists of actions, and parameters
3. Each bucket or list of buckets can be applied to entering packets; the exact behavior depends on the group type
4. There are four types of groups – ALL, SELECT, INDIRECT, and FAST-FAILOVER.

Group Table

ALL Group

ALL group, will take any packet received as input and duplicate it to be operated on independently by each bucket in the bucket list. In this way, an ALL group can be used to replicate and then operate on separate copies of the packet defined by the actions in each bucket.

Example usecase: Port Mirror

SELECT Group

SELECT group, which is primarily designed for load balancing

SELECT group has an assigned weight, and each packet that enters the group is sent to a single bucket. The bucket selection algorithm is undefined and is dependent on the switch's implementation;

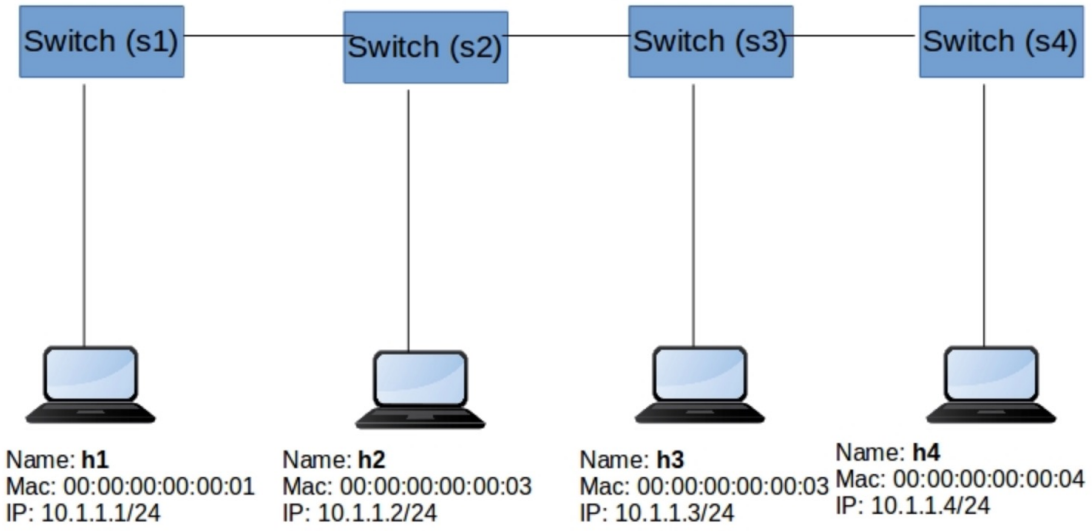
Example usecase: Load Balancer

อ้างอิงเพิ่มเติม: https://docs.openvswitch.org/en/latest/faq/openflow/

Group Table

2. Demo -Sniffer

Objective



Group Table

Make h2 as Sniffer machine. it will sniff all the packets passing via S2.

Steps

1. run the topology in mininiet

```
sudo mn --controller=remote,ip=127.0.0.1 --mac -i 10.1.1.0/24 --switch=ovsk,protocols=OpenFlow13 --
topo=linear,4
```

2. Run the ryu controller application(simple switch and ofctl)

```
ryu-manager ryu.app.simple_switch_13 ryu.app.ofctl_rest
```

3. Do pingall from mininiet

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet>
```

Group Table

4. Check the flows of switch s2

```
sudo ovs-vsctl show
sudo ovs-ofctl -O OpenFlow13 dump-flows s2
sudo ovs-ofctl -O OpenFlow13 dump-groups s2
```

5. Configure the Group in S2

Group table1(Group Table ID 50):

Create a Group table with TYPE=ALL(it means, copy a packet for each bucket. and each bucket will be processed). create two buckets. one bucket will send the packet to Port3, another bucket will send the packet to Port1

Group table2(Group ID 51):

Create a Group table with TYPE=ALL(it means, copy a packet for each bucket. and each bucket will be processed). create two buckets. one bucket will send the packet to Port2, another bucket will send the packet to Port1

Method: POST URI: /stats/groupentry/add

Group Table

Method: POST URI: /stats/groupentry/add

> *curl -X POST http://localhost:8080/stats/groupentry/add -d '@group50.json'*
>
> *curl -X POST http://localhost:8080/stats/groupentry/add -d '@group51.json'*

Check the group tables

```
suresh@suresh-vm:~$ sudo ovs-ofctl -O OpenFlow13 dump-groups s2
OFPST_GROUP_DESC reply (OF1.3) (xid=0x2):
 group_id=50,type=all,bucket=actions=output:"s2-eth1",bucket=actions=output:"s2-eth2"
 group_id=51,type=all,bucket=actions=output:"s2-eth1",bucket=actions=output:"s2-eth3"
suresh@suresh-vm:~$
```

Group Table

6. Configure the Flows

- All the packets received from port3 will be forwarded to Group table1(Group table ID 50)
- All the packets received from port2 will be forwarded to Group table2(Group table ID 51)

> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@flow1.json'*
>
> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@flow2.json'*

Check the flow tables

```
suresh@suresh-vm:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s2
 cookie=0x0, duration=51.164s, table=0, n_packets=8, n_bytes=784, priority=100,in_port="s2-eth3"
actions=group:50
 cookie=0x0, duration=18.320s, table=0, n_packets=6, n_bytes=588, priority=100,in_port="s2-eth2"
actions=group:51
 ....OUTPUT skipped
```

Group Table

Now newly added flows have high priority. So this will take control of forwarding the packets.

6. Testing

Trigger a continuos ping from h1 to h4 and capture traffic in h2 using tcpdump

a) start the xterm for h2

> *mininet> xterm h2*

In the h2 terminal capture tcpdump

> *tcpdump -i any -v*

Group Table

b) continuous ping from h1 to h4

```
mininet> h1 ping h4
PING 10.1.1.4 (10.1.1.4) 56(84) bytes of data.
64 bytes from 10.1.1.4: icmp_seq=1 ttl=64 time=2.15 ms
64 bytes from 10.1.1.4: icmp_seq=2 ttl=64 time=0.080 ms
```

c) check the h2 xterm window, you will observe the h1 to h4 traffic

d)

check the group stats

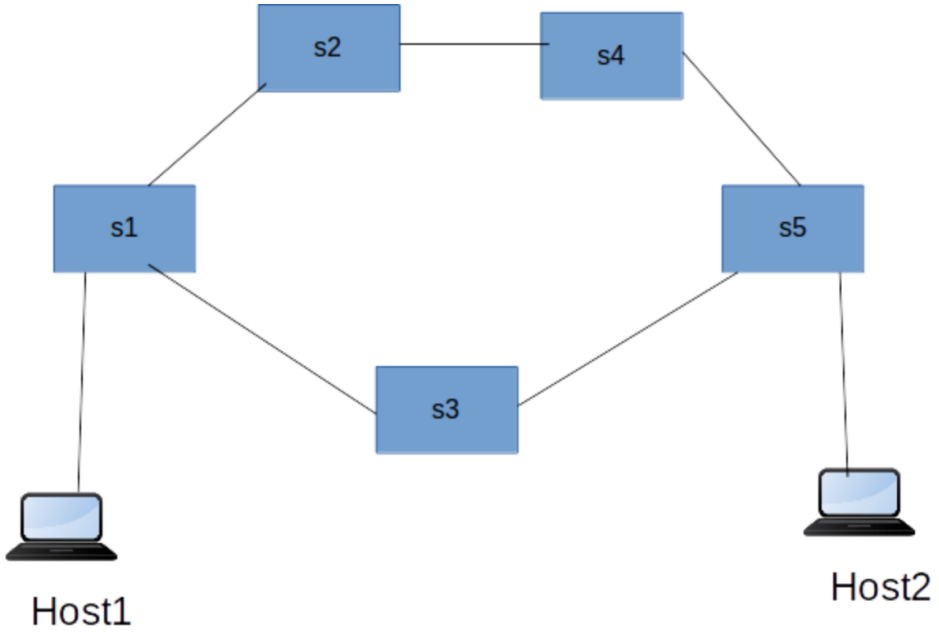
```
suresh@suresh-vm:~$ sudo ovs-ofctl -O OpenFlow13 dump-group-stats s2
OFPST_GROUP reply (OF1.3) (xid=0x4):

group_id=50,duration=1171.522s,ref_count=1,packet_count=223,byte_count=20790,bucket0:packet_count=223,b
yte_count=20790,bucket1:packet_count=223,byte_count=20790

group_id=51,duration=1119.832s,ref_count=1,packet_count=219,byte_count=20510,bucket0:packet_count=219,b
yte_count=20510,bucket1:packet_count=219,byte_count=20510
suresh@suresh-vm:~$
```

Group Table

3. Demo - Multipath Loadbalancer

Objective

Forward the packet to 1 bucket(out of N buckets) and process it. (Load Balancer)



Group Table

Steps

1. run the topology in mininiet

```
sudo python topo.py
```

2. Run the ryu manager ofctl application

```
ryu-manager ryu.app.ofctl_rest
```

3. Add the static ARP Entry in the mininiet hosts

```
mininet> h1 arp -s 192.168.1.2 00:00:00:00:00:02
mininet> h2 arp -s 192.168.1.1 00:00:00:00:00:01
```

Group Table

4. add the flow entries for s2,s3,s4 (it simple, receive one end forward other side)

> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@s2_flow1.json'*
>
> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@s2_flow2.json'*
>
> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@s3_flow1.json'*
>
> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@s3_flow2.json'*
>
> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@s4_flow1.json'*
>
> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@s4_flow2.json'*

Group Table

Verify the flows

```
sudo ovs-ofctl -O OpenFlow13 dump-flows s2
sudo ovs-ofctl -O OpenFlow13 dump-flows s3
sudo ovs-ofctl -O OpenFlow13 dump-flows s4
```

4. Add Group table for S1

Create a Group table with TYPE=SELECT (it means, For each bucket, a bucket will be selected (based on weight - vendor implementation). and the selected bucket will be processed).

create two buckets. one bucket will send the packet to S2 Port, another bucket will send the packet to S3 Port

> *curl -X POST http://localhost:8080/stats/groupentry/add -d '@s1_group50.json'*

verify the group table

```
sudo ovs-ofctl -O OpenFlow13 dump-groups s1
```

Group Table

5. Add Group table for S5

Create a Group table with TYPE=SELECT (it means, For each bucket, a bucket will be selected (based on weight - vendor implementation). and the selected bucket will be processed).

create two buckets. one bucket will send the packet to S4 Port, another bucket will send the packet to S3 Port.

> *curl -X POST http://localhost:8080/stats/groupentry/add -d '@s5_group51.json'*

verify the group table

```
sudo ovs-ofctl -O OpenFlow13 dump-groups s5
```

Group Table

6. Add flows for S1

> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@s1_flow1.json'*
>
> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@s1_flow2.json'*
>
> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@s1_flow3.json'*

7. Add flows for S5

> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@s5_flow1.json'*
>
> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@s5_flow2.json'*
>
> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@s5_flow3.json'*

Group Table

8. Testing

Verify the flows

```
sudo ovs-ofctl -O OpenFlow13 dump-flows s1
sudo ovs-ofctl -O OpenFlow13 dump-flows s2
sudo ovs-ofctl -O OpenFlow13 dump-flows s3
sudo ovs-ofctl -O OpenFlow13 dump-flows s4
sudo ovs-ofctl -O OpenFlow13 dump-flows s5
```

run iperf test between h1 and h5

```
mininet> h2 iperf -u  -s &
mininet> h2 iperf   -s &
mininet> h1 iperf -c h2 -t 60 -P 5 &
```

Check the flows and groups.

Statistics

1. Introduction

We can collect the statistics of flow tables, queues, meter, ports etc. OFCTL REST application provides API to collect the stats. Also RYU in built application simple_monitor_13.py also collect the statistics at regular interval.

Statistics collection is used to identify various traffic monitoring, threat, security applications, Network wide visualization.

OpenFlow Messages:

Some of the Important Statistics Messages are

Flow Statistics Message Aggregate Flow Statistics Message Table Statistics Port Statistics Queue Statistics Group Statistics Meter Statistics

Controller sends the Statistics Request Message. Switch will repond with Statistics Reply Message.

Statistics

2. OFCTL REST API for Statistics

In the Previous sessions, Already we have discussed about the OFCTL REST API for statistics collection.

https://ryu.readthedocs.io/en/latest/app/ofctl_rest.html

Statistics

3. Simple_Monitor Application

This application collects the Flow and PORT Statistics at regular interval and print it.

1) Start the simple mininet topology

```
sudo mn --controller=remote,ip=127.0.0.1 --mac -i 10.1.1.0/24 --topo=single,2
```

2) Run the RYU simple monitor application

```
ryu-manager ryu.app.simple_monitor_13
```

3) Perform TCP Traffic test between h1 and h2

Statistics

4) see the ryu manager terminal, which prints the port stats at regular interval

```
datapath         in-port  eth-dst           out-port packets  bytes
---------------- -------- ----------------- -------- -------- --------
0000000000000001        1 00:00:00:00:00:02        2   701403 33029276718
0000000000000001        2 00:00:00:00:00:01        1   530580 35018712
datapath         port     rx-pkts  rx-bytes rx-error tx-pkts  tx-bytes tx-error
---------------- -------- -------- -------- -------- -------- -------- --------
0000000000000001        1   701414 33029277624        0   530610 35022624        0
0000000000000001        2   530591 35019610        0   701440 33029281104        0
0000000000000001 fffffffe        0        0        0        0        0        0
```

Meter Table

1. Introduction

A switch element that can measure and control the rate of packets. The meter triggers a meter band if the packet rate/byte rate passing through the meter exceeds a predefined threshold. If the meter band drops the packet, it is called a Rate Limiter.

A meter table consists of meter entries, defining per-flow meters. Per-flow meters enable OpenFlow to implement various simple QoS operations, such as rate-limiting.

Meter Table

2. Test Setup

Meter table feature is released in openvswitch 2.10+ versions. Ubuntu 18.10 OS has support the OVS 2.10 in its repo.

So, To test the meter table, Either we need to install OVS 2.10 version in our OS manually. (or) use ubuntu 18.10 OS.

In Ubuntu 18.10 OS, install the openvswitch as below

```
sudo apt-get install openvswitch-switch
```

Meter Table

3. Testing without meter

1. Run the mininet simple topology

> *sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,2*

2. Run the RYU Application

> *ryu-manager ryu.app.simple_switch_13*

3. start the UDP server in h1 and h2

```
mininet> h2 iperf -u -s &
mininet> h1 iperf -u -s &
```

Meter Table

4. Perform the 10Mbps UDP Traffic test from h1 to h2 It means h1 pushing 10Mbps traffic to h2. h2 receives this traffic.

```
mininet> h1 iperf -u -c h2 -b 10m
------------------------------------------------------------
Client connecting to 10.0.0.2, UDP port 5001
Sending 1470 byte datagrams, IPG target: 1176.00 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.1 port 51179 connected with 10.0.0.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  11.9 MBytes  10.0 Mbits/sec
[  3] Sent 8505 datagrams
[  3] Server Report:
[  3]  0.0-10.0 sec  11.9 MBytes  10.0 Mbits/sec   0.000 ms    0/ 8505 (0%)
[  3] 0.00-9.96 sec  30 datagrams received out-of-order
mininet>
```

Meter Table

5. Perform the 10Mbps UDP Traffic test from h2 to h1

It means h2 pushing 10Mbps traffic to h1. h1 receives this traffic.

```
mininet> h2 iperf -u -c h1 -b 10m

[  3] local 10.0.0.2 port 51337 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  11.9 MBytes  10.0 Mbits/sec
[  3] Sent 8505 datagrams
[  3] Server Report:
[  3]  0.0-10.0 sec  11.9 MBytes  10.0 Mbits/sec   0.000 ms    0/ 8505 (0%)
mininet>
```

Now both directions we are able to achieve 10Mbps traffic, as no meter is configured.

Meter Table

4. Testing with meter

Objective

Apply the rate-limit for incoming traffic of host h1 (MAC adddress - 00:00:00:00:00:01).

Steps

1. Run the mininet simple topology

> *sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,2*

2. Run the RYU Application

> *ryu-manager ryu.app.ofctl_rest*

Meter Table

3. Add the Meter

API Details:

https://ryu.readthedocs.io/en/latest/app/ofctl_rest.html#add-a-meter-entry

Configuring 1000Kbps rate limit

> *curl -X POST http://localhost:8080/stats/meterentry/add -d '@addmeter.json'*

4. Add the flows

> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@switch_arp.json'*
>
> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@switch_flow1.json'*
>
> *curl -X POST http://localhost:8080/stats/flowentry/add -d '@switch_flow2.json'*

In the switch_flow1, we are applying the meter rate limit. it means, h1 host is applied with rate limit.

H1 host receives only 1000kbps traffic.

Meter Table

5. Check the flow tables

```
suresh@suresh:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=36.097s, table=0, n_packets=0, n_bytes=0, priority=100,dl_dst=ff:ff:ff:ff:ff:ff
actions=FLOOD
 cookie=0x0, duration=28.766s, table=0, n_packets=0, n_bytes=0, priority=100,dl_dst=00:00:00:00:00:01
actions=meter:1,output:"s1-eth1"
 cookie=0x0, duration=14.650s, table=0, n_packets=0, n_bytes=0, priority=100,dl_dst=00:00:00:00:00:02
actions=output:"s1-eth2"
suresh@suresh:~$
```

6. check the meter table

```
suresh@suresh:~$ sudo ovs-ofctl -O OpenFlow13 dump-meters s1
OFPST_METER_CONFIG reply (OF1.3) (xid=0x2):
meter=1 kbps bands=
type=drop rate=1000
suresh@suresh:~$
```

Meter Table

7. Start the UDP server in h1 and h2

```
mininet> h2 iperf -u -s &
mininet> h1 iperf -u -s &
```

8. Perform the 10Mbps UDP Traffic test from h1 to h2

It means h1 pushing 10Mbps traffic to h2. h2 receives this traffic.

```
mininet> h1 iperf -u -c h2 -b 10m
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
------------------------------------------------------------
Client connecting to 10.0.0.2, UDP port 5001
Sending 1470 byte datagrams, IPG target: 1176.00 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.1 port 45492 connected with 10.0.0.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  11.9 MBytes  10.0 Mbits/sec
[  3] Sent 8505 datagrams
[  3] Server Report:
[  3]  0.0-10.0 sec  11.9 MBytes  10.0 Mbits/sec   0.000 ms    0/ 8505 (0%)
mininet>
```

We have not restricted the h1 outgoing traffic. we restricted only on h1 incoming traffic. hence we see 10mbps going.

Meter Table

9. Perform the 10Mbps UDP Traffic test from h2 to h1

It means h2 pushing 10Mbps traffic to h1. h1 receives this traffic.

This has to be rate limited by the Meter , as this traffic is target to h1(incoming trafffic)

```
mininet> h2 iperf -u -c h1 -b 10m
[  3] Server Report:
[  3]  0.0-10.3 sec  1.39 MBytes  1.14 Mbits/sec   0.000 ms 7512/ 8504 (0%)
```

10. Check the meter table

> *ovs-ofctl -O OpenFlow13 meter-stats s1*

```
suresh@suresh:~$ sudo ovs-ofctl -O OpenFlow13 meter-stats s1

OFPST_METER reply (OF1.3) (xid=0x2):
meter:1 flow_count:1 packet_in_count:19531 byte_in_count:29520582 duration:1093.735s bands:

0: packet_count:15715 byte_count:23761080
```

Flow Manager Application

1. Introduction

https://github.com/martimy/flowmanager The FlowManager is a RYU controller application that gives the user manual control over the flow tables in an OpenFlow network. The user can create, modify, or delete flows directly from the application. The user can also monitor the OpenFlow switches and view statistics.

Flowmanager doesnot have inbuilt switching application. So, if user want switching operations(reactive switching), run switching application(simple switch, l3, l4 switch, .. etc) along with flowmanger.

Flow Manager Application

2. Installation

Just clone or download the flowmanger application from github.

> *git clone https://github.com/martimy/flowmanager*

Flow Manager Application

3. Quick Demo

Demo

1. Start the RYU flow manager application

> *ryu-manager --observe-links ~/flowmanager/flowmanager.py ryu.app.simple_switch_13*

2. Run Mininiet topology

> *sudo mn --controller=remote,ip=127.0.0.1 --mac -i 10.1.1.0/24 --topo=tree,depth=2,fanout=2*

Flow Manager Application

3. Open the flow manager application UI in Google Chrome Browser

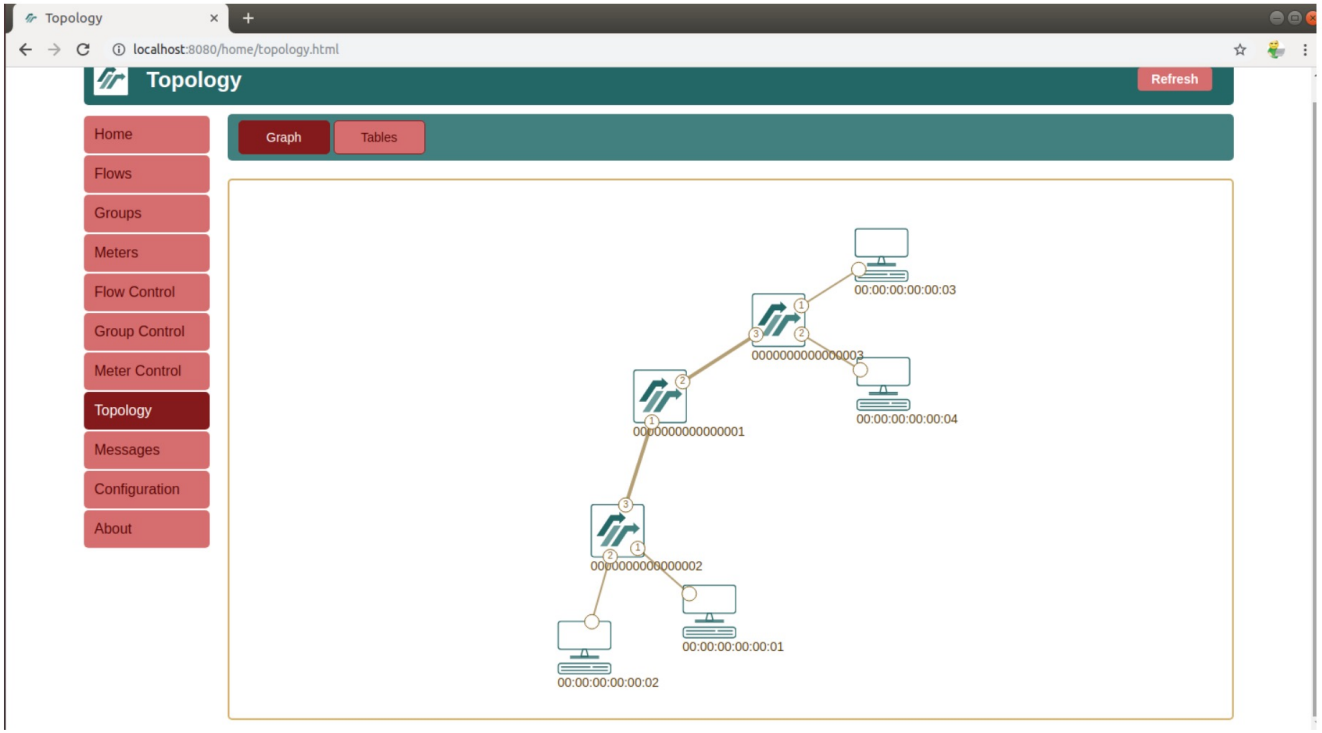
> *http://localhost:8080/home/index.html*

4. do pingall from mininet

> *mininet>pingall*

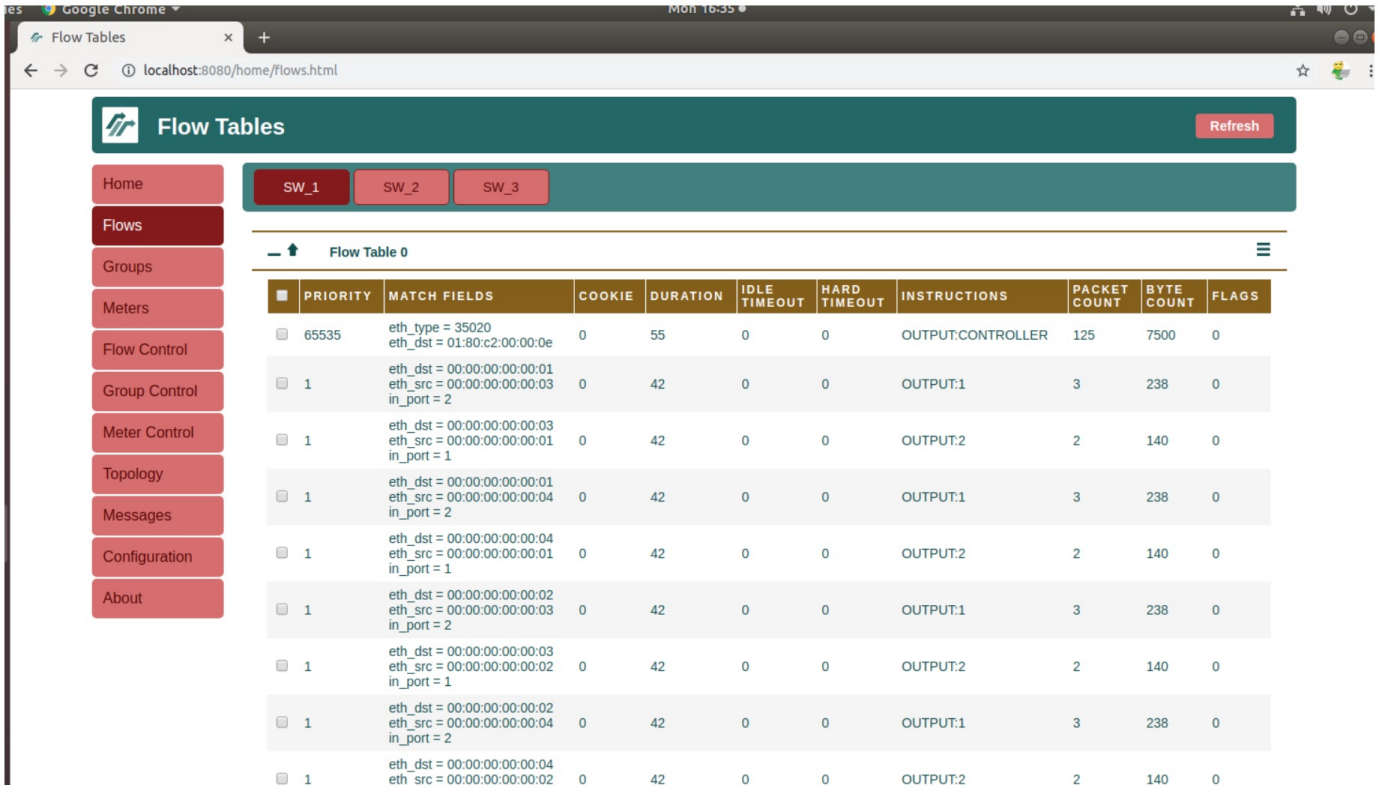
Flow Manager Application

5. Check the topology diagram



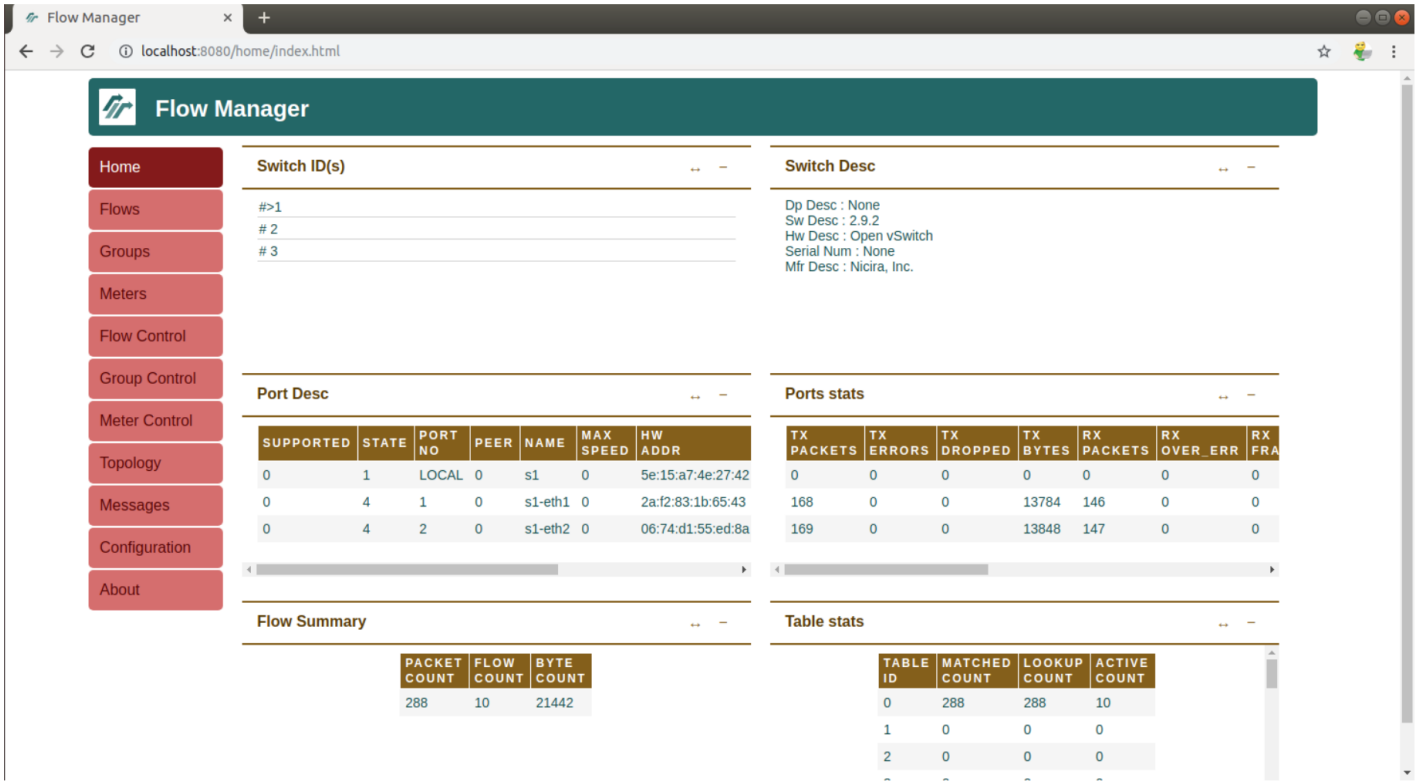
Flow Manager Application

6. Check the Flows



Flow Manager Application

7. Check the Switch & Ports



Flow Manager Application

4. Flow Operations

Flow control tab provides the Add, Modify , Delete Flow operations.

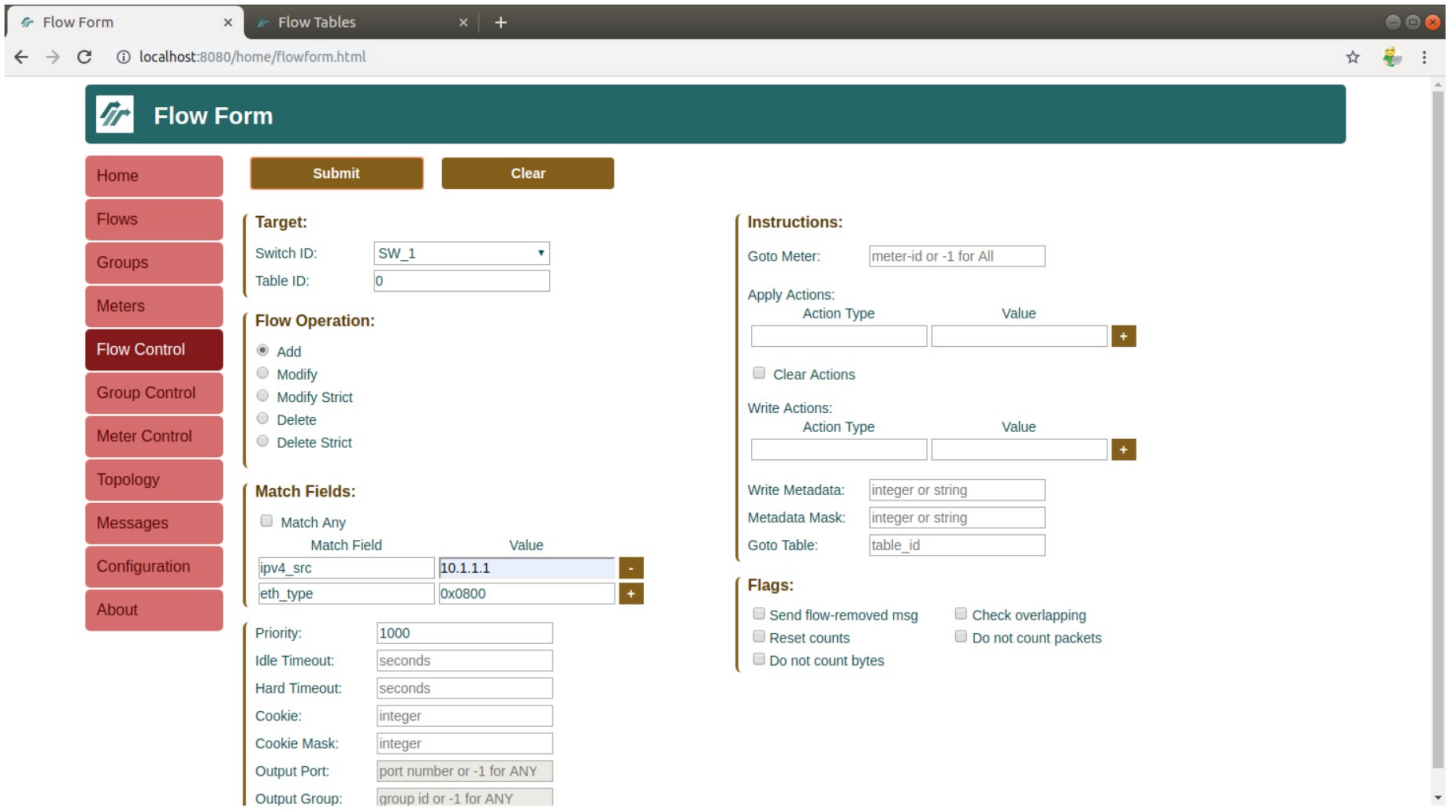
Add a flow

Lets add a flow S1 to drop all packets generated by 10.1.1.1 host.

Select Add Operation -Select Match ipv4_src 10.1.1.1 eth_type 0x0800

- priority 1000
- action
  output port -1

Flow Manager Application

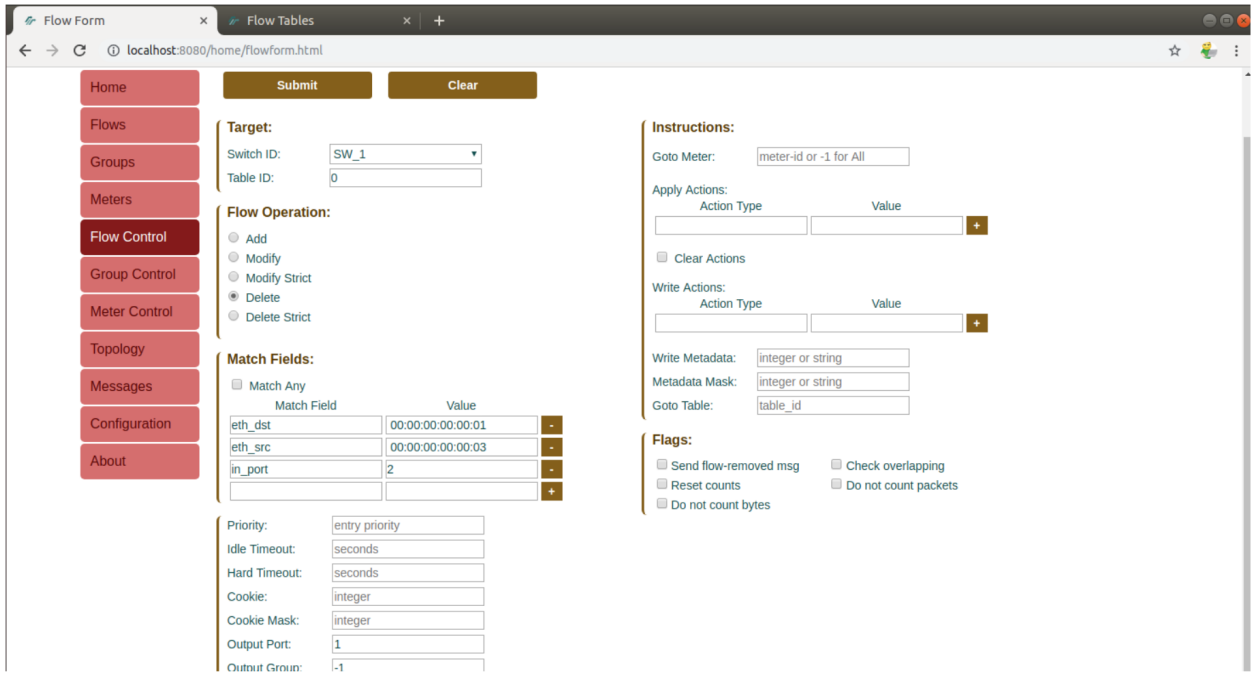


Now check the flows.

Flow Manager Application

Delete a Flow with exact match

- Select the Delete operation
- Provide the Match fields (all fields)
- Provide the output Port
- submit it

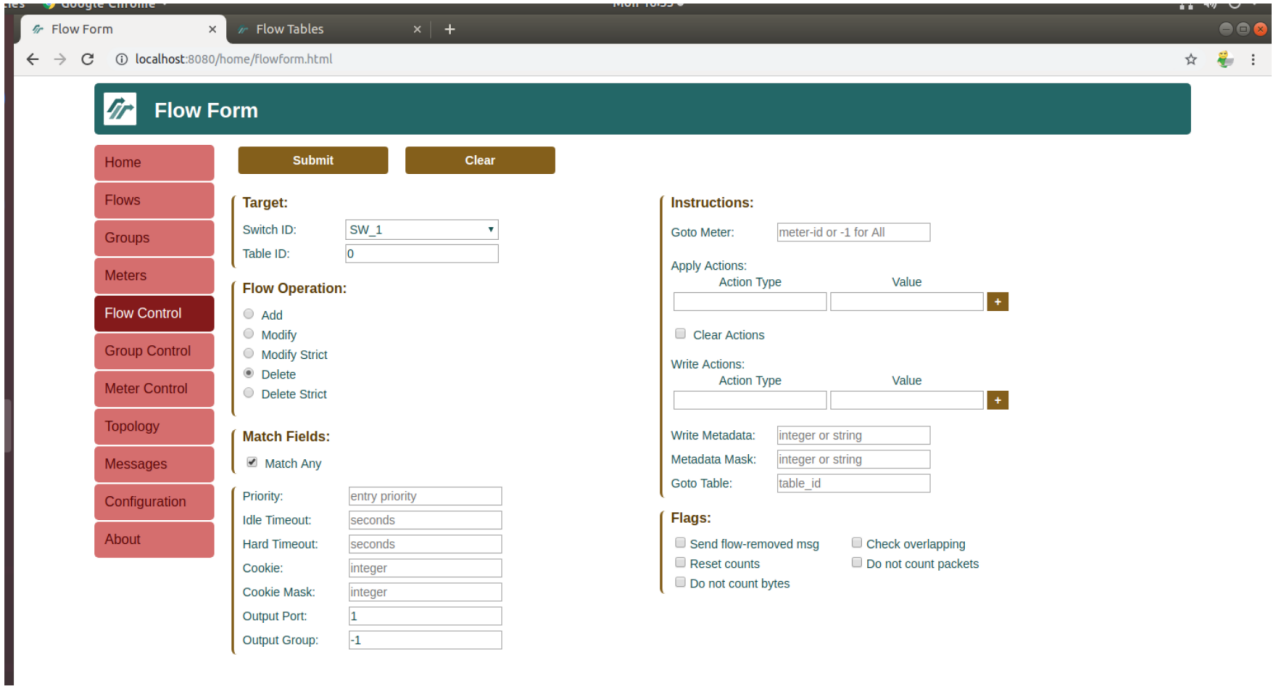


Now check the flows.

Flow Manager Application

Delete flows with only output port number

- Select the Delete operation
- Select Match Any
- Provide the output Port
- submit it



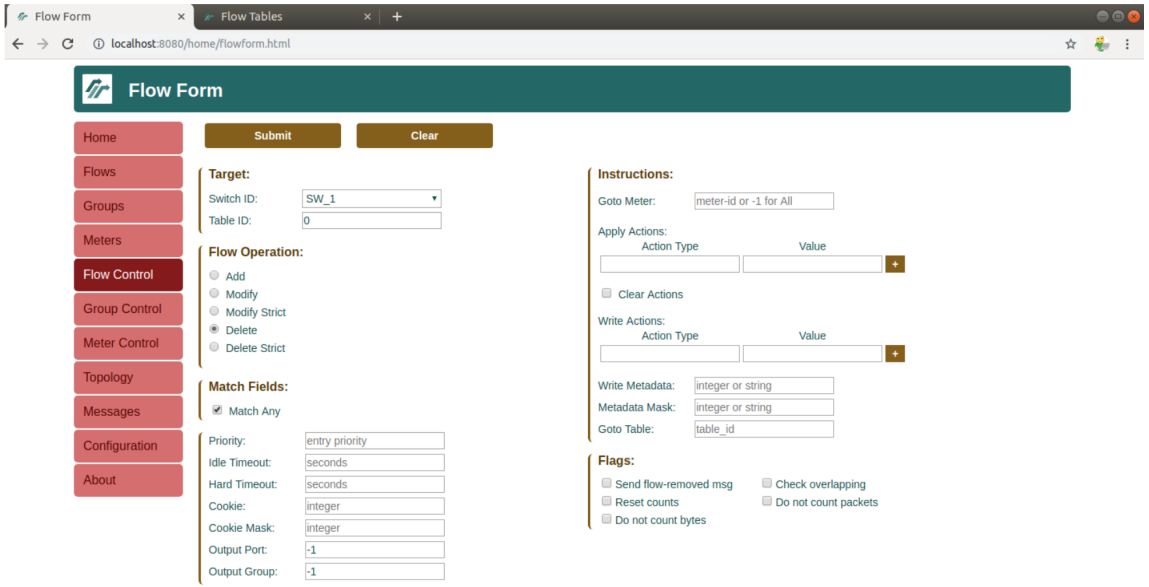
Now check the flows.

Flow Manager Application

Delete all flows

Select the Delete operation

- Select Match Any
- Provide the output Port as (-1)
- submit it



Now check the flows.

Table Miss entry also will be removed. So , we need to destroy the topology and start again to continue the experiment.

Homework

- **RYU OFCTL REST API application**
  - Add a flow based on tuple of 5 Params(src ip, dst ip, protocol, src port, dst port)
  - Add a full fledged Access Control List (Application protocol ex:HTTP,DNS, DHCP etc)
  - Build MPLS forwarding table
  - Soruce Nat & Destination Nat exercises.