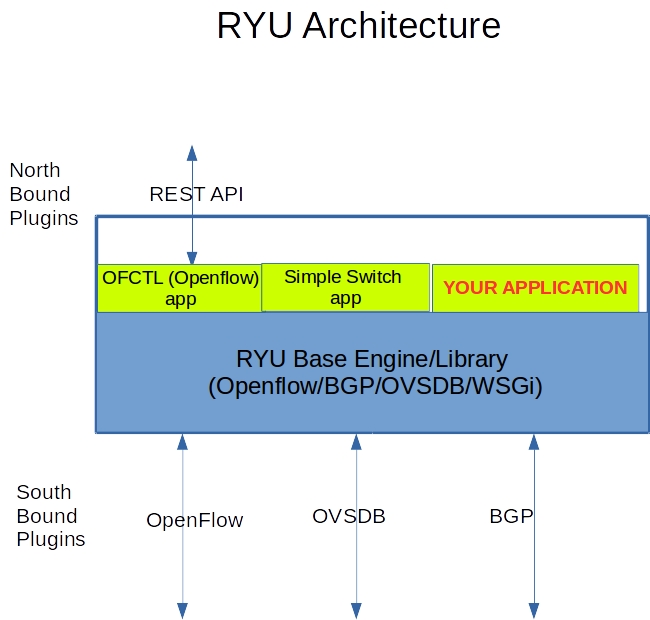
**CCGC5002- Lab 4-1 Introduction to Ryu**

Compile solutionsinto a single word/pdf file and upload in the Lab 4 folder under Assignments tab of Blackboard. Lab 4 has got 2 parts – Lab 4-1 Introduction to Ryu and Lab 4-2 OFCTL REST

**1. Introduction**

Ryu is a component-based software defined networking framework. Ryu provides software components with well defined API that make it easy for developers to create new network management and control applications. Ryu supports various protocols for managing network devices, such as OpenFlow, OVSDB, BGP. About OpenFlow, Ryu supports fully 1.0, 1.2, 1.3, 1.4, 1.5 and Nicira Extensions. All of the code is freely available under the Apache 2.0 license.



**2. How to run RYU applications.**

**run the ryu application:**

*ryu-manager ryu.app.*

In built applications are available in

<https://github.com/osrg/ryu/tree/master/ryu/app>

Some of the applications are ,

1. simple\_switch
2. simple\_monitor
3. ofctl\_rest
4. rest\_qos
5. rest\_firewall
6. rest\_router

*ryu-manager ryu.app.simple\_switch\_13*

Example:

tanvir@tanvir-vm:~$ ryu-manager ryu.app.simple\_switch\_13

loading app ryu.app.simple\_switch\_13

loading app ryu.controller.ofp\_handler

instantiating app ryu.app.simple\_switch\_13 of SimpleSwitch13

instantiating app ryu.controller.ofp\_handler of OFPHandler

​

**Check the Openflow port status**

RYU Manager listens on openflow ports(6653) are in listening state.

*netstat -ap | grep 6653*

tanvir@tanvir-vm:~$ netstat -ap | grep 6653

(Not all processes could be identified, non-owned process info

will not be shown, you would have to be root to see it all.)

tcp       0     0 0.0.0.0:6653           0.0.0.0:\*               LISTEN     19190/python

tanvir@tanvir-vm:~$

**How to stop the ryu controller**

CTRL + C (Kill the Process)

**How to run your (custom developed) applications.**

RYU application is a python script.

ryu-manager <python-file-name>

​

Example:

*ryu-manager l3\_switch.py*

**How to run your multiple applications.**

RYU can run multiple applications in a single initiation.

ryu-manager <application1> <application2>

​

Example:

*ryu-manager ryu.app.simple\_switch\_13 ryu.app.ofctl\_rest*

## 3. RYU Controller command line options

**To know all the available options**

*ryu-manager --help*

**To enable the debug logs:**

*ryu-manager --verbose*

**To use custom openflow port number**

*ryu-manager --ofp-tcp-listen-port 6634*

Example:

ryu:

​

ryu-manager --ofp-tcp-listen-port 6634 ryu.app.simple\_switch\_13

​

Mininet:

​

sudo mn --controller=remote,ip=127.0.0.1:6634 --switch=ovsk,protocols=OpenFlow13 --topo=linear,4

**To use topology discovery**

*ryu-manager --observe-links*

## 4. Reactive/Proactive Flows

**Reactive Flows:**

* When the new packet enters in the switch, if it doesnot match on the existing flows, Switch sends it to the controller.
* controller inspect the packet, and build the logic
* install the flow for that session(match) in the switch

Packet IN /Packet Out

**Proactive Flows:**

* OpenFlow controller will install flow tables ahead of time for all traffic matches.

## 5. OpenFlow Applications

**Applications are Part of SDN Controller**

* Most of the RYU Applications are this category. example , simple\_switch application, monitor application.
* To develop this application, we should know the RYU Python API, and RYU Programming guidelines.
* Most of the academic projects will be developed in this type.

**Application sits externally, and communicate with SDN Controller thru North bound plugin**

* User using NorthBound interfaces(REST API) to add the flows ,
* Packet In /Packet Out will not be considered as the flow control is handled externally by the user or external application.
* No Packet generation capability(Packetout)

## 6. Basic Openvswitch Commands

​

ovs-vsctl show

​

ovs-ofctl -O OpenFlow13 dump-flows <bridgename>

​

ovs-ofctl -O OpenFlow13 show <bridgename>

​

## 7. Switching applications

### A. Simple Proactive Hub Application

Install the Openflow flow in the switch which performs FLOOD action, when switch connects to the controller.

**Testing**

1. Run Mininet topology

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

tanvir@tanvir-vm:~/sdn$ sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,4

[sudo] password for tanvir:

\*\*\* Creating network

\*\*\* Adding controller

Unable to contact the remote controller at 127.0.0.1:6653

Unable to contact the remote controller at 127.0.0.1:6633

Setting remote controller to 127.0.0.1:6653

\*\*\* Adding hosts:

h1 h2 h3 h4

\*\*\* Adding switches:

s1

\*\*\* Adding links:

(h1, s1) (h2, s1) (h3, s1) (h4, s1)

\*\*\* Configuring hosts

h1 h2 h3 h4

\*\*\* Starting controller

c0

\*\*\* Starting 1 switches

s1 ...

\*\*\* Starting CLI:

mininet>

​

1. Run RYU hub application

*ryu-manager hub.py*

tanvir@tanvir-vm:~/sdn$ ryu-manager hub.py

loading app hub.py

loading app ryu.controller.ofp\_handler

instantiating app ryu.controller.ofp\_handler of OFPHandler

instantiating app hub.py of SimpleSwitch13

1. Check the flows

*sudo ovs-ofctl -O OpenFlow13 dump-flows s1*

tanvir@tanvir-vm:~/sdn$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1

[sudo] password for tanvir:

cookie=0x0, duration=16.055s, table=0, n\_packets=5, n\_bytes=350, priority=0 actions=FLOOD

tanvir@tanvir-vm:~/sdn$

1. Perform Ping between the hosts in mininet

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>

1. Watch the flows again

tanvir@tanvir-vm:~/sdn$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1

cookie=0x0, duration=654.092s, table=0, n\_packets=72, n\_bytes=5040, priority=0 actions=FLOOD

tanvir@tanvir-vm:~/sdn$

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*take screenshot of the above screen.

### B. Simple Switch Application (in built)

Simple Switch Application is RYU inbuit basic switching application works in reactive model.

* Install the Table Miss entry to the switch
* When the packet comes to Switch, it matches with Table Miss Entry, then Switch send it to the Controller(PACKET IN message)
* Controller look the src mac of the packet and updates in its db(port to mac mapping)
* Controller look the destination mac of the packet, and decides on the output port .
* Controller send the packet to switch (PACKET OUT message)
* Controller add the flow using (FLOW Modificcation message), here match field is based on MAC address.

**Testing**

1. Run Mininet topology

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

tanvir@tanvir-vm:~/sdn$ sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,4

[sudo] password for tanvir:

\*\*\* Creating network

\*\*\* Adding controller

Unable to contact the remote controller at 127.0.0.1:6653

Unable to contact the remote controller at 127.0.0.1:6633

Setting remote controller to 127.0.0.1:6653

\*\*\* Adding hosts:

h1 h2 h3 h4

\*\*\* Adding switches:

s1

\*\*\* Adding links:

(h1, s1) (h2, s1) (h3, s1) (h4, s1)

\*\*\* Configuring hosts

h1 h2 h3 h4

\*\*\* Starting controller

c0

\*\*\* Starting 1 switches

s1 ...

\*\*\* Starting CLI:

mininet>

​

1. Run RYU simple switch application

*ryu-manager ryu.app.simple\_switch\_13*

tanvir@tanvir-vm:~/sdn$ ryu-manager ryu.app.simple\_switch\_13

loading app ryu.app.simple\_switch\_13

loading app ryu.controller.ofp\_handler

instantiating app ryu.app.simple\_switch\_13 of SimpleSwitch13

instantiating app ryu.controller.ofp\_handler of OFPHandler

packet in 1 00:00:00:00:00:03 33:33:00:00:00:02 3

packet in 1 00:00:00:00:00:04 33:33:00:00:00:02 4

packet in 1 00:00:00:00:00:01 33:33:00:00:00:02 1

packet in 1 00:00:00:00:00:02 33:33:00:00:00:02 2

1. Check the switch & flows

Table Miss entry to be present

tanvir@tanvir-vm:~/sdn$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1

cookie=0x0, duration=39.737s, table=0, n\_packets=8, n\_bytes=560, priority=0 actions=CONTROLLER:65535

tanvir@tanvir-vm:~/sdn$

1. Do h1 to h2 ping from mininet prompt

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=18.5 ms

64 bytes from 10.0.0.2: icmp\_seq=2 ttl=64 time=0.672 ms

64 bytes from 10.0.0.2: icmp\_seq=3 ttl=64 time=0.119 ms

64 bytes from 10.0.0.2: icmp\_seq=4 ttl=64 time=0.148 ms

^C

--- 10.0.0.2 ping statistics ---

4 packets transmitted, 4 received, 0% packet loss, time 3027ms

rtt min/avg/max/mdev = 0.119/4.869/18.540/7.896 ms

mininet>

​

1. Check the flows again

tanvir@tanvir-vm:~/sdn$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1

cookie=0x0, duration=35.152s, table=0, n\_packets=5, n\_bytes=434, priority=1,in\_port="s1-eth2",dl\_src=00:00:00:00:00:02,dl\_dst=00:00:00:00:00:01 actions=output:"s1-eth1"

cookie=0x0, duration=35.140s, table=0, n\_packets=4, n\_bytes=336, priority=1,in\_port="s1-eth1",dl\_src=00:00:00:00:00:01,dl\_dst=00:00:00:00:00:02 actions=output:"s1-eth2"

cookie=0x0, duration=139.207s, table=0, n\_packets=19, n\_bytes=1302, priority=0 actions=CONTROLLER:65535

tanvir@tanvir-vm:~/sdn$

\*\*\*\*\*\*\*\*\*\*\*\*\*\*take screenshot of the above screen.

1. Look the Priority, Match and Action field

Priority:

*priority=1*

Match:

*in\_port="s1-eth2",dl\_src=00:00:00:00:00:02,dl\_dst=00:00:00:00:00:01*

Action:

*output:"s1-eth1"*

### C. Simple Switch with L3 Match Application

This exercise is same as Simple Switch Application, except Match is based on IP address.

1. Run Mininet topology

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

1. Run RYU l3switch application

*ryu-manager l3\_switch.py*

1. Check the switch & flows
2. Do h1 to h2 ping from mininet prompt
3. Check the flows again

tanvir@tanvir-vm:~/sdn$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1

cookie=0x0, duration=10.369s, table=0, n\_packets=1, n\_bytes=98, priority=1,ip,nw\_src=10.0.0.1,nw\_dst=10.0.0.2 actions=output:"s1-eth2"

cookie=0x0, duration=10.361s, table=0, n\_packets=1, n\_bytes=98, priority=1,ip,nw\_src=10.0.0.2,nw\_dst=10.0.0.1 actions=output:"s1-eth1"

cookie=0x0, duration=17.720s, table=0, n\_packets=10, n\_bytes=644, priority=0 actions=CONTROLLER:65535

tanvir@tanvir-vm:~/sdn$

1. Look the Priority, Match and Action field

Priority:

*priority=1*

Match:

*ip,nw\_src=10.0.0.1,nw\_dst=10.0.0.2*

Action:

*output:"s1-eth2"*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*take screenshot of the above screen.

### D. Simple Switch with L4 Match Application

This exercise is same as Simple Switch Application, except Match is based on IP address,IP Protocol, src and dst Port

1. Run Mininet topology

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

1. Run RYU l4switch application

*ryu-manager l4\_switch.py*

1. Check the switch & flows
2. Do h1 to h2 ping from mininet prompt
3. start iperf tcp server in h2

mininet> h2 iperf -s &

1. Run iperf client in h1 connecting to h2

mininet> h1 iperf -c h2

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

Client connecting to 10.0.0.2, TCP port 5001

TCP window size: 85.3 KByte (default)

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

[ 3] local 10.0.0.1 port 59010 connected with 10.0.0.2 port 5001

[ ID] Interval Transfer Bandwidth

[ 3] 0.0-10.0 sec 32.8 GBytes 28.1 Gbits/sec

mininet>

1. Check the flows again

tanvir@tanvir-vm:~/sdn$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1

cookie=0x0, duration=131.163s, table=0, n\_packets=1, n\_bytes=98, priority=1,icmp,nw\_src=10.0.0.1,nw\_dst=10.0.0.2 actions=output:"s1-eth2"

cookie=0x0, duration=131.154s, table=0, n\_packets=1, n\_bytes=98, priority=1,icmp,nw\_src=10.0.0.2,nw\_dst=10.0.0.1 actions=output:"s1-eth1"

cookie=0x0, duration=85.890s, table=0, n\_packets=773576, n\_bytes=35217018256, priority=1,tcp,nw\_src=10.0.0.1,nw\_dst=10.0.0.2,tp\_src=59010,tp\_dst=5001 actions=output:"s1-eth2"

cookie=0x0, duration=85.875s, table=0, n\_packets=668305, n\_bytes=44108130, priority=1,tcp,nw\_src=10.0.0.2,nw\_dst=10.0.0.1,tp\_src=5001,tp\_dst=59010 actions=output:"s1-eth1"

cookie=0x0, duration=137.765s, table=0, n\_packets=30, n\_bytes=1996, priority=0 actions=CONTROLLER:65535

tanvir@tanvir-vm:~/sdn$

1. Look the Priority, Match and Action field

Priority:

*priority=1*

Match:

*tcp,nw\_src=10.0.0.1,nw\_dst=10.0.0.2,tp\_src=59010,tp\_dst=5001*

Action:

*output:"s1-eth2"*

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*take screenshot of the above screen.

### E. Simple Switch with flow expiry (flow timeouts configured)

This exercise is same as Simple Switch Application with idle\_timeout and hard\_timeout. So the flow will be removed/expired after certain time.

1. Run Mininet topology

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

1. Run RYU flow timeout application

ryu-manager flow\_timeout.py

1. Do pingall from mininet prompt

mininet> pingall

1. Check the flows continuously

tanvir@tanvir-vm:~/Desktop/sdn/ryu\_apps$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1

cookie=0x0, duration=1.681s, table=0, n\_packets=1, n\_bytes=98, idle\_timeout=10, hard\_timeout=30, priority=1,in\_port="s1-eth1",dl\_src=00:00:00:00:00:01,dl\_dst=00:00:00:00:00:02 actions=output:"s1-eth2"

cookie=0x0, duration=1.670s, table=0, n\_packets=1, n\_bytes=98, idle\_timeout=10, hard\_timeout=30, priority=1,in\_port="s1-eth2",dl\_src=00:00:00:00:00:02,dl\_dst=00:00:00:00:00:01 actions=output:"s1-eth1"

cookie=0x0, duration=1.660s, table=0, n\_packets=1, n\_bytes=98, idle\_timeout=10, hard\_timeout=30, priority=1,in\_port="s1-eth1",dl\_src=00:00:00:00:00:01,dl\_dst=00:00:00:00:00:03 actions=output:"s1-eth3"

cookie=0x0, duration=1.655s, table=0, n\_packets=1, n\_bytes=98, idle\_timeout=10, hard\_timeout=30, priority=1,in\_port="s1-eth3",dl\_src=00:00:00:00:00:03,dl\_dst=00:00:00:00:00:01 actions=output:"s1-eth1"

cookie=0x0, duration=1.646s, table=0, n\_packets=1, n\_bytes=98, idle\_timeout=10, hard\_timeout=30, priority=1,in\_port="s1-eth1",dl\_src=00:00:00:00:00:01,dl\_dst=00:00:00:00:00:04 actions=output:"s1-eth4"

1. Look the idle\_timeout, hard\_timeout value
2. Check the flows again after 10 secs(idle\_timeout time) . Flows will be expired and not available in the switch.

tanvir@tanvir-vm:~/Desktop/sdn/ryu\_apps$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1

cookie=0x0, duration=116.277s, table=0, n\_packets=47, n\_bytes=3794, priority=0 actions=CONTROLLER:65535

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*take screenshot of the above screen.

## 8. Controller Connection Failure

Openvswitch support two mode of operation upon Controller connection fails A. Standalone Mode B. Secure Mode

**A. Standalone mode:**

* Openvswitch act like an ordinary MAC-learning switch(traditional switch).
* Openvswitch will take over responsibility for setting up flows (when no message has been received from the controller for three times the inactivity probe interval)
* In the background, Openvswtitch retry connecting to the controller, when it succees it switch it to the openflow mode.

**B. Secure Mode:**

* In this mode, Openvswitch will try to connect forever. It will not set up flows on its own when the controller connection fails.

**How to configure**:

you can check the current configuration, using

*ovs-vsctl show*

To configure:

*ovs-vsctl set-fail-mode standalone ovs-vsctl set-fail-mode secure*

Example:

sudo ovs-vsctl set-fail-mode s1 standalone

### Demo

1. Start the Mininet Topology

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

1. Run the RYU SDN Controller with flowtimeout application.

ryu-manager flow\_timeout.py

1. Check the Switch mode

sudo ovs-vsctl show

1. Do continuous ping from h1 to h2

mininet> h1 ping h2

1. Stop the Controller
2. After flows are expired, the ping stopped. No data traffic will be passed in the switch.
3. Start the controller again.
4. Flows will be installed, and Ping will continue
5. Configure the switch in "standalone mode"

sudo ovs-vsctl set-fail-mode s1 standalone

1. Stop the controller again
2. After the flows are expired, the switch will move to standalone mode and continue to pass the traffic.