Software Defined Network
SDN

Ch 3

OpenFlow Theory & Ryo Controller - Basics

OpenFlow Theory

Introduction

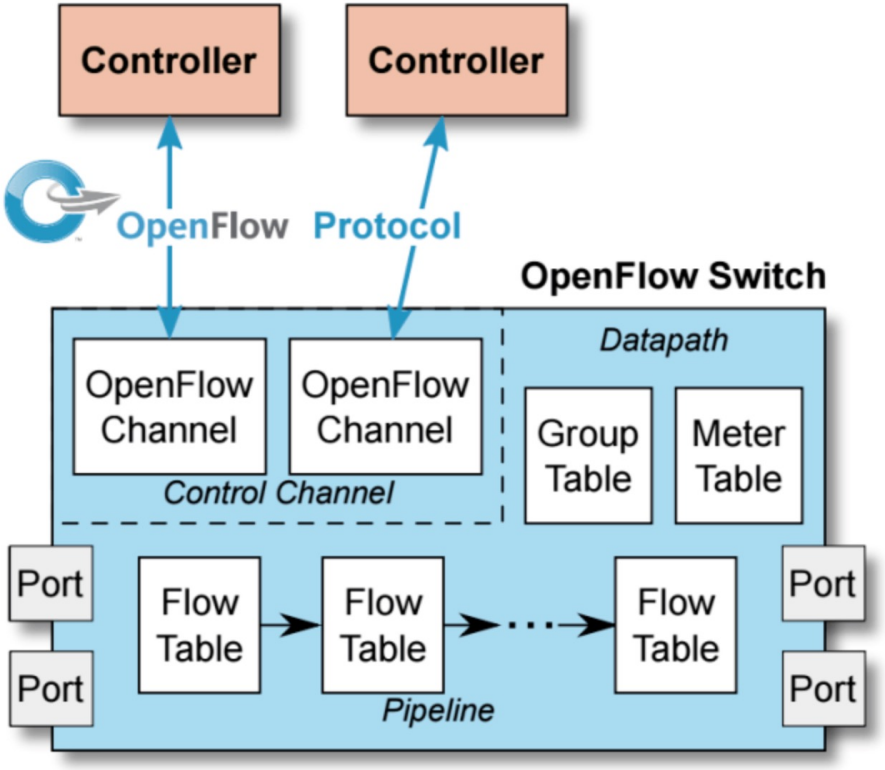
The OPENFLOW specification covers the components and the basic functions of the switch, and the OpenFlow switch protocol to manage an OpenFlow switch from a remote OpenFlow controller.

Openflow Version Details:

Openflow 1.1 Openflow 1.2 Openflow 1.3 Openflow 1.4 Openflow 1.5

Most widley used: 1.3

Switch Components



Switch Components

- The switch communicates with the controller and the controller manages the switch via the OpenFlow switch protocol.
- Using the OpenFlow switch protocol, the controller can add, update, and delete flow entries in flow tables, both reactively (in response to packets) and proactively.
- Each flow table in the switch contains a set of flow entries; each flow entry consists of match fields, counters, and a set of instructions to apply to matching packets.
- Matching starts at the first flow table and may continue to additional flow tables of the pipeline
- Flow entries match packets in priority order, with the first matching entry in each table being used.

Switch Components

- If a matching entry is found, the instructions associated with the specific flow entry are executed.
- If no match is found in a flow table, the outcome depends on configuration of the table-miss flow entry: for example, the packet may be forwarded to the controllers over the OpenFlow channel, dropped, or may continue to the next flow table
- Actions included in instructions describe packet forwarding, packet modification and group table processing.
- Flow entries may forward to a port. This is usually a physical port, but it may also be a logical port defined by the switch(such as link aggregation groups, tunnels or loopback interfaces) or a reserved port defined by this specification.

OpenFlow Channel

- The OpenFlow channel is the interface that connects each OpenFlow Logical Switch to an OpenFlow controller. Through this interface, the controller configures and manages the switch, receives events from the switch, and sends packets out the switch.
- The Control Channel of the switch may support a single OpenFlow channel with a single controller, or multiple OpenFlow channels enabling multiple controllers.
- The OpenFlow channel is usually encrypted using TLS, but may be run directly over TCP
- Default Port number : 6653

OpenFlow Flow Table

A flow table entry is identified by its match fields and priority: the match fields and priority taken together identify a unique flow entry in a specific flow table.

A flow table consists of flow entries.

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags

Table 1: Main components of a flow entry in a flow table.

OpenFlow Flow Table

Example Flows with MAC Match

```
 cookie=0x0, duration=4.742s, table=0, n_packets=2, n_bytes=196, priority=1,in_port="s1-
eth2",dl_src=00:00:00:00:11:12,dl_dst=00:00:00:00:11:11 actions=output:"s1-eth1"
 cookie=0x0, duration=4.738s, table=0, n_packets=1, n_bytes=98, priority=1,in_port="s1-
eth1",dl_src=00:00:00:00:11:11,dl_dst=00:00:00:00:11:12 actions=output:"s1-eth2"
 cookie=0x0, duration=5.781s, table=0, n_packets=29, n_bytes=3102, priority=0 actions=CONTROLLER:65535
```

Example Flows with IP Match

```
 cookie=0x0, duration=12.927s, table=0, n_packets=2, n_bytes=196,
priority=1,ip,nw_src=192.168.1.1,nw_dst=192.168.1.2 actions=output:"s1-eth2"
 cookie=0x0, duration=12.918s, table=0, n_packets=2, n_bytes=196,
priority=1,ip,nw_src=192.168.1.2,nw_dst=192.168.1.1 actions=output:"s1-eth1"
 cookie=0x0, duration=12.959s, table=0, n_packets=37, n_bytes=3844, priority=0 actions=CONTROLLER:65535
```

Example Flows with TCP/UDP Ports Match

```
suresh@suresh-vm:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=3.933s, table=0, n_packets=238752, n_bytes=11069190464,
priority=1,tcp,nw_src=192.168.1.2,nw_dst=192.168.1.1,tp_src=37304,tp_dst=5001 actions=output:"s1-eth1"
 cookie=0x0, duration=3.906s, table=0, n_packets=192421, n_bytes=12699810,
priority=1,tcp,nw_src=192.168.1.1,nw_dst=192.168.1.2,tp_src=5001,tp_dst=37304 actions=output:"s1-eth2"
 cookie=0x0, duration=31.495s, table=0, n_packets=43, n_bytes=4309, priority=0 actions=CONTROLLER:65535
suresh@suresh-vm:~$
```

OpenFlow Flow Table

match fields: to match against packets. These consist of the ingress port and packet headers, and optionally other pipeline fields such as metadata specified by a previous table

priority: matching precedence of the flow entry.

counters: updated when packets are matched.

instructions: to modify the action set or pipeline processing.

timeouts: maximum amount of time or idle time before flow is expired by the switch.

cookie: opaque data value chosen by the controller. May be used by the controller to filter flow entries affected by flow statistics, flow modification and flow deletion requests. Not used when processing packets.

flags: flags alter the way flow entries are managed, for example the flag OFPFF_SEND_FLOW_REM triggers flow removed messages for that flow entry.

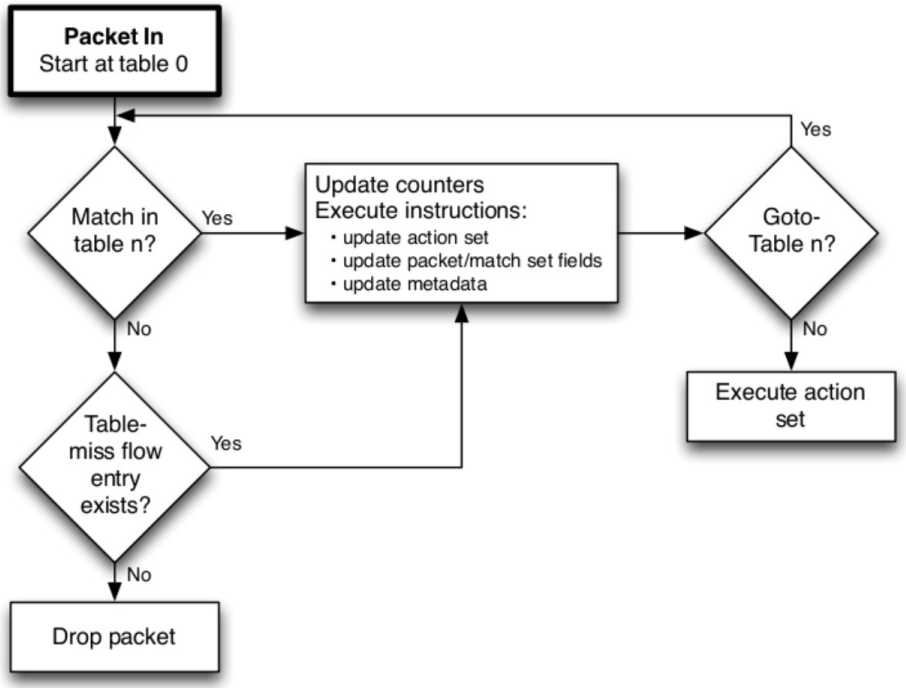
The flow entry that wildcards all match fields (all fields omitted) and has priority equal to 0 is called the table-miss flow entry.

The table-miss flow entry must support at least sending packets to the controller using the CONTROLLER reserved port.

OpenFlow Flow Table

Openflow Matching

On receipt of a packet, an OpenFlow Switch performs the functions shown as below.



Flowchart detailing packet flow through an OpenFlow switch.

OpenFlow Flow Table

```
Switch input port. */
Switch physical input port. *
Ethernet destination address. */
Ethernet source address. */
Ethernet frame type. */
VLAN id. */
VLAN priority. */
IP DSCP (6 bits in ToS field). */
IP ECN (2 bits in ToS field). */
IP protocol. */
IPv4 source address. */
IPv4 destination address. */
TCP source port. */
TCP destination port. */
UDP source port. */
UDP destination port. */
SCTP source port. */
SCTP destination port. */
ICMP type. */
ICMP code. */
ARP opcode. */
ARP source IPv4 address. */
ARP target IPv4 address. */
ARP source hardware address. */
ARP target hardware address. */
IPv6 source address. */
```

OpenFlow Flow Table

```
IPv6 destination address. */
IPv6 Flow Label */
ICMPv6 type. */
ICMPv6 code. */
Target address for ND.
Source link-layer for ND. */
Target link-layer for ND. */
MPLS label. */
MPLS TC. */
MPLS BoS bit. */
PBB I-SID. */
Logical Port Metadata. */
IPv6 Extension Header pseudo-field */
```

OpenFlow Flow Table

- Match fields come in two types, header match fields and pipeline match fields.
- Header match fields are match fields matching values extracted from the packet headers. Most header match fields map directly to a specific field in the packet header defined by a datapath protocol
- All header match fields have different size, prerequisites and masking capability
- Pipeline match fields are match fields matching values attached to the packet for pipeline processing and not associated with packet headers, such as META_DATA, TUNNEL_ID. Refer :

OpenFlow Flow Table

Prerequisties Example:

If you want to include the SRC IP Address in the match, Prerequisties is ETH_TYPE should be 0X0800. It means, you need to include the ETH_TYPE match also.

Instructions

Each flow entry contains a set of instructions that are executed when a packet matches the entry. These instructions result in changes to the packet, action set and/or pipeline processing.

Apply-Actions action(s):

- Applies the specific action(s) immediately, without any change to the Action Set.
- This instruction may be used to modify the packet between two tables or to execute multiple actions of the same type.
- The actions are specified as a list of actions

OpenFlow Flow Table

Clear-Actions:

- Clears all the actions in the action set immediately.

Goto-Table next-table-id:

- Indicates the next table in the processing pipeline. The table-id must be greater than the current table-id.

Write-Actions action(s):

- Merges the specified set of action(s) into the current action set

Meter meter id:

- Direct packet to the specified meter.

OpenFlow Flow Table

Action Set and Actions:

Action set contains set of actions.

Example actions are:

- Output port no
- Group group id
- Drop
- Set-Queue queue id
- Push-Tag/Pop-Tag ethertype (VLAN, MPLS, PBP)
- Set-Field field type value
- Change-TTL (IP TTL, MPLS TTL)

OpenFlow Flow Table

Flow Removal

Flow entries are removed from flow tables in two ways, either at the request of the controller or via the switch flow expiry mechanism.

Flow expiry:

Each flow entry has an idle_timeout and a hard_timeout associated with it

Example:

```
 cookie=0x0, duration=7.619s, table=0, n_packets=3, n_bytes=238, idle_timeout=10, hard_timeout=30,
priority=1,in_port="s1-eth2",dl_src=00:00:00:00:11:12,dl_dst=00:00:00:00:11:11 actions=output:"s1-eth1"
 cookie=0x0, duration=7.605s, table=0, n_packets=2, n_bytes=140, idle_timeout=10, hard_timeout=30,
priority=1,in_port="s1-eth1",dl_src=00:00:00:00:11:11,dl_dst=00:00:00:00:11:12 actions=output:"s1-eth2"
 cookie=0x0, duration=8.652s, table=0, n_packets=33, n_bytes=3527, priority=0 actions=CONTROLLER:65535
```

OpenFlow Flow Table

Hard_timeout If the hard_timeout field is non-zero, the switch must note the flow entry's arrival time, as it may need to evict the entry later. A non-zero hard_timeout field causes the flow entry to be removed after the given number of seconds, regardless of how many packets it has matched.

idle_timeout If the idle_timeout field is non-zero, the switch must note the arrival time of the last packet associated with the flow, as it may need to evict the entry later. A non-zero idle_timeout field causes the flow entry to be removed when it has matched no packets in the given number of seconds. The switch must implement flow expiry and remove flow entries from the flow table when one of their timeouts is exceeded.

Counters

statistics are maintained by the openflow switch as below,

- Per flow entry
- Per flow table
- Per Switch Port
- Per Queue
- Per Group
- Per Group Bucket
- Per Meter
- Per Meter Band

OpenFlow Flow Table

Openflow Messages

Three types of messages.

1. Controller to Switch

Controller-to-switch messages are initiated by the controller and used to directly manage or inspect the state of the switch

- Feature Request
- Packet Out
- Modify Flow Table
- Modify Group Table
- Modify Meter Table
- OpenFlow Switch Description Request
- OpenFlow Port Description Request
- Openflow Statistics Request(Flow, Port, Flowtable, Aggregate, Group, Meter, Queue)
- Role Request
- Barrier Request

OpenFlow Flow Table

2. Asynchronous

Asynchronous messages are initiated by the switch and used to update the controller about network events and changes to the switch state

- Packet In
- Flow Removed

3. Symmetric

Symmetric messages are initiated by either the switch or the controller and sent without solicitation.

- Hello Message
- Echo Message

OpenFlow Flow Table

Message transcation - during the Topology Setup

1. Hello
2. Feature request/Response
3. Switch/Port Description Request/Response
4. Modify Flow Entry (To install table Miss entry)
5. Packet IN (Switch to Controller)
6. Packet Out (Controller to Switch)
7. Modify Flow Entry (Install a flow)
8. Echo

Hello Message:

Hello messages are exchanged between the switch and controller upon connection startup.

- Switch sends Openflow Hello Message(includes version number) to the Controller
- Controller responds with the Hello Message if version is supported.
- Failure Case(Version MisMatch): If different Openflow Version is user between the Controller and Switch, Hello Message will fail.
- You will see similar error msg in the controller.

```
Error:
unsupported version 0x1. If possible, set the switch to use one of the versions [3]
```

OpenFlow Flow Table

Echo Message:

Echo request/reply messages can be sent from either the switch or the controller, and must return an echo reply. They are mainly used to verify the liveness of a controller-switch connection, and may as well be used to measure its latency or bandwidth.(default: 5sec interval)

A. Switch sends Echo Request to the Controller.

B. Controller responds back with Echo Reply.

Features Request/Reply:

- The controller may request the identity and the basic capabilities of a switch by sending a features request;
- The switch must respond with a features reply that specifies the identity and basic capabilities(datapath ID, buffers, number of tables, statistics) of the switch.
- This is commonly performed upon establishment of the OpenFlow channel.

OpenFlow Flow Table

Packet-in:

Transfer the control of a packet to the controller. For all packets forwarded to the CONTROLLER reserved port using a flow entry or the table-miss flow entry, a packet-in event is always sent to controllers

Packet-out Message:

These are used by the controller to send packets out of a specified port on the switch, and to forward packets received via Packet-in messages. Packet-out messages must contain a full packet or a buffer ID referencing a packet stored in the switch.

The message must also contain a list of actions to be applied in the order they are specified; an empty list of actions drops the packet.

Modify Flow Entry Message:

Modifications to a flow table from the controller are done with the OFPT_FLOW_MOD message:

- To add, remove, modify the flow in the switch, controller using this message.
- Controller Sends the Flow Modification message to the switch with this important params (Command, Match, Instruction, action.)
- Command: ADD, MODIFY, MODIFY_STRICT, DELETE, DELETE_STRICT

OpenFlow Ports

Physical ports:

The OpenFlow physical ports are switch defined ports that correspond to a hardware interface of the switch. In the Virtualized environment it represents the virutal interface.

Example: "s1-eth1"

Logical ports:

Logical ports are higher level abstractions that may be defined in the switch using non-OpenFlow methods (e.g. link aggregation groups, tunnels, loopback interfaces).

Example: "vxlan0"

Reserved ports:

The OpenFlow reserved ports are defined by this specification. They specify generic forwarding actions such as sending to the controller, flooding, or forwarding using non-OpenFlow methods, such as “normal” switch processing. FLOOD, ALL, CONTROLLER, IN PORT, LOCAL, NORMAL,

Example: FLOOD

Important Take Aways

1) Openflow version should match between the switch and Controller

2) Our Controller Application(our RYU project/exercise) should process Packet IN (Message), to build the Switching/Routing logic.

3) Our Controller Application(our RYU project/exercise) should use Flow Modifcation message to add/modify/delete the flows in the switch.

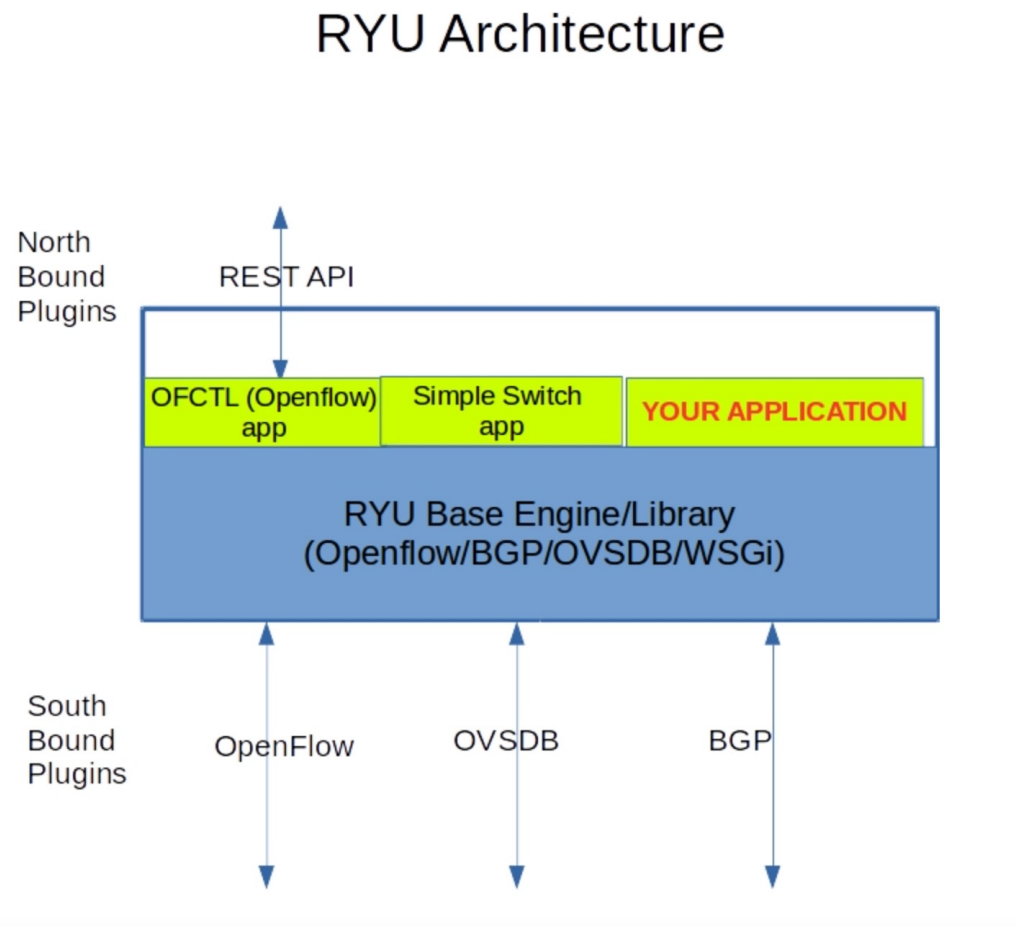
4) Our Controller Application(our RYU project/exercise) should use Flow Stats, Port Stats request message to get the statistics(Packets Sent/Received , etc) of the flows, Ports .

RYU Controller - Basics

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.

RYU Controller - Basics



RYU Controller - Basics

How to run RYU applications

run the ryu application:

```
ryu-manager ryu.app.application-name
```

Example:

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

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 Controller - Basics

> ryu-manager ryu.app.simple_switch_13

Example:

```
suresh@suresh-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

```
suresh@suresh-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/py
suresh@suresh-vm:~$
```

RYU Controller - Basics

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

RYU Controller - Basics

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

RYU Controller - Basics

RYU Controller command line options

To know all the available options

> ryu-manager --help

To enable the debug logs:

> ryu-manager --verbose application_name

To use custom openflow port number

> ryu-manager --ofp-tcp-listen-port 6634 application_name

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

RYU Controller - Basics

To use topology discovery

```
ryu-manager --observe-links application_name
```

Example:

```
ryu-manager --observe-links ryu.app.simple_switch_13
```

RYU Controller - Basics

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.

RYU Controller - Basics

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
```

RYU Controller - Basics

```
suresh@suresh-vm:~/sdn$ sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,p
[sudo] password for suresh:
*** 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>
```

RYU Controller - Basics

2. Run RYU hub application

> ryu-manager hub.py

```
suresh@suresh-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
```

3. Check the flows

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

```
suresh@suresh-vm:~/sdn$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
[sudo] password for suresh:
 cookie=0x0, duration=16.055s, table=0, n_packets=5, n_bytes=350, priority=0 actions=FLOOD
suresh@suresh-vm:~/sdn$
```

RYU Controller - Basics

4. 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>
```

5. Watch the flows again

```
suresh@suresh-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
suresh@suresh-vm:~/sdn$
```

RYU Controller - Basics

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)

RYU Controller - Basics

Basic Openvswitch Commands

```
ovs-vsctl show

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

ovs-ofctl -O OpenFlow13 show <bridgename>
```

RYU Controller - Basics

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.

RYU Controller - Basics

Testing

1. Run Mininet topology

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

```
suresh@suresh-vm:~/sdn$ sudo mn --controller=remote,ip=127.0.0.1 --mac --
switch=ovsk,protocols=OpenFlow13 --topo=single,4
[sudo] password for suresh:
*** 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>
```

RYU Controller - Basics

2. Run RYU simple switch application

> ryu-manager ryu.app.simple_switch_13

```
suresh@suresh-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
```

RYU Controller - Basics

3. Check the switch & flows

Table Miss entry to be present

```
suresh@suresh-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
suresh@suresh-vm:~/sdn$
```

4. 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>
```

RYU Controller - Basics

5. Check the flows again

```
suresh@suresh-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
suresh@suresh-vm:~/sdn$
```

6. 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"

RYU Controller - Basics

Simple L3 Switch

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
```

2. Run RYU l3switch application

```
ryu-manager l3_switch.py
```

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

```
suresh@suresh-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
 cookie=0x0, duration=10.361s, table=0, n_packets=1, n_bytes=98, priority=1,ip,nw_src=10
 cookie=0x0, duration=17.720s, table=0, n_packets=10, n_bytes=644, priority=0 actions=CO
suresh@suresh-vm:~/sdn$
```

RYU Controller - Basics

6. 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"
```

RYU Controller - Basics

Simple L4 Switch

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

2. Run RYU l4switch application

> ryu-manager l4_switch.py

3. Check the switch & flows

4. Do h1 to h2 ping from mininet prompt

5. start iperf tcp server in h2

```
mininet> h2 iperf –s &
```

RYU Controller - Basics

6. 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>
```

7. Check the flows again

```
suresh@suresh-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
suresh@suresh-vm:~/sdn$
```

RYU Controller - Basics

6. 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"
```

RYU Controller - Basics

Simple Switch with flow expiry

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

2. Run RYU flow timeout application

```
ryu-manager flow_timeout.py
```

3. Do pingall from mininet prompt

```
mininet> pingall
```

RYU Controller - Basics

4. Check the flows continuously

```
suresh@suresh-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"
```

5. Look the idle_timeout, hard_timeout value
6. Check the flows again after 10 secs(idle_timeout time) . Flows will be expired and not available in the switch.

```
suresh@suresh-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
```

RYU Controller - Basics

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.

RYU Controller - Basics

How to configure:

you can check the current configuration, using

> ovs-vsctl show

To configure:

> ovs-vsctl set-fail-mode switch-name standalone ovs-vsctl set-fail-mode switch-name secure

Example:

```
sudo ovs-vsctl set-fail-mode s1 standalone
```

RYU Controller - Basics

Demo

1. Start the Mininet Topology

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

2. Run the RYU SDN Controller with flowtimeout application.

```
ryu-manager flow_timeout.py
```

Copy

3. Check the Switch mode

```
sudo ovs-vsctl show
```

4. Do continuous ping from h1 to h2

```
mininet> h1 ping h2
```

RYU Controller - Basics

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

```
sudo ovs-vsctl set-fail-mode s1 standalone
```

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