RYU SDN Crash Course - Book

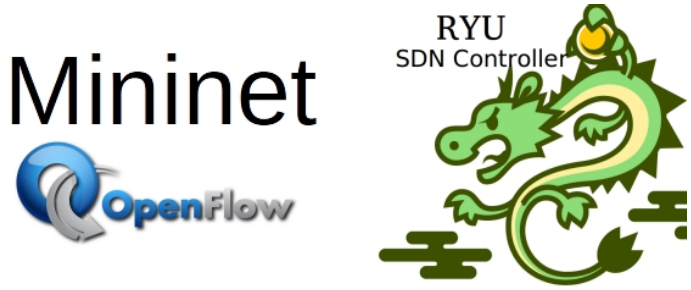
KNET Solutions (Online Training Centre),

http://knetsolutions.in

knetsolutions2@gmail.com

Phone/WhatsApp: +919445042007

Academic SDN Mini Project Training.



1. Introduction

This Book is (**RYU SDN CRASH COURSE Course book**), prepared by KNET Solutions. This Book is used as Course material for UDEMY SDN Crash Course.

The Course is available in Udemy . The Course link is UDEMY SDN CRASH COURSE

This Course covers SDN Basics, OpenFlow Theory, Mininet Basics, Mininet Programming, RYU Controller Basics, RYU Controller built-in applications, RYU Controller Programming, Writing your own sdn application in RYU Controller, Multicontroller environments etc.

Some book contents (IMAGE/Text) are copied from Freely available resources from internet / RFCs/ Opensource materials. Thanks to the original authors.

This Book is free to use.

Diarmuid O'Briain made an extensive RYU Tutorials, RYU Tutorial

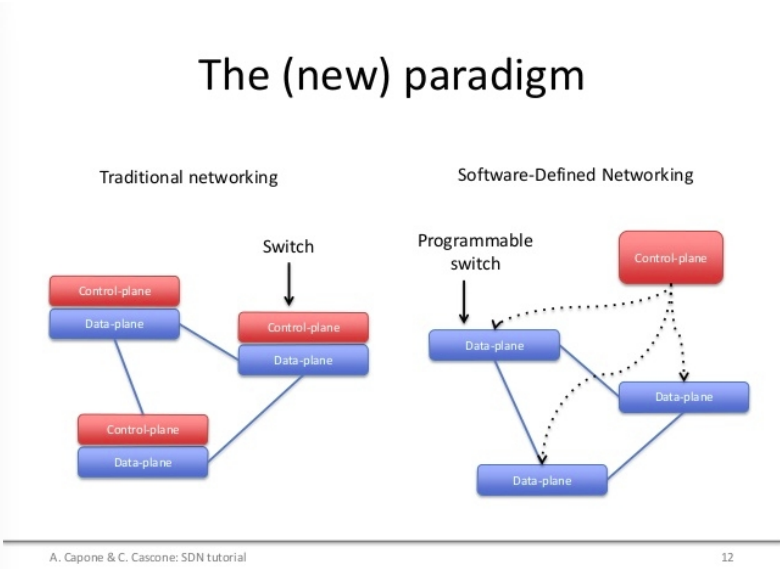
---

2. SDN Overview

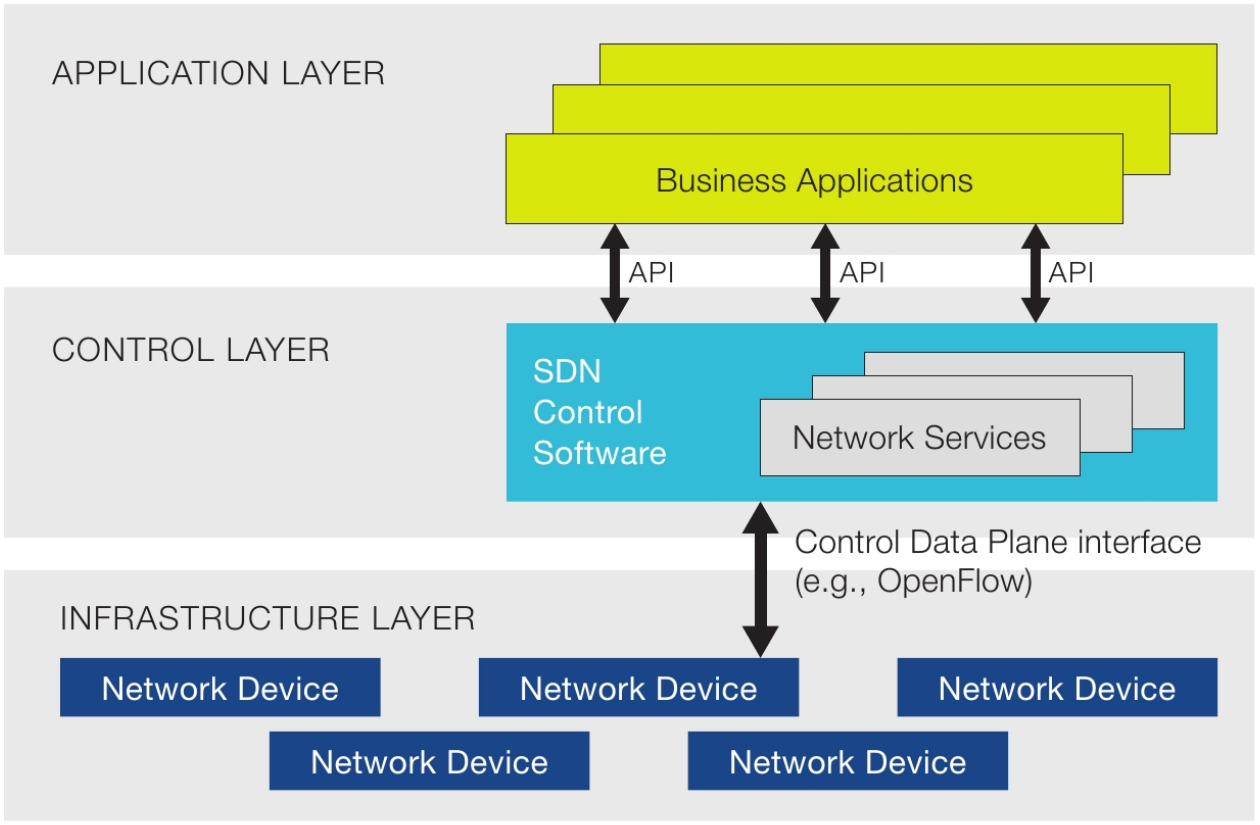
SDN Introduction

SDN suggests to centralize network intelligence in one network component by disassociating the forwarding process of network packets (data plane) from the routing process (control plane). The control plane consists of one or more controllers which are considered as the brain of SDN network where the whole intelligence is incorporated.

Enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. The OpenFlow protocol is a foundational element for building SDN solutions.



SDN Architecutre



SDN architectures generally have three components or groups of functionality:

The SDN networking devices control the forwarding and data processing capabilities for the network. This includes forwarding and processing of the data path. Example: Openvswitch (openflow protocol)

SouthBound Interface:

Southbound interface is the connection between the controller and the physically networking hardware

SDN Controller

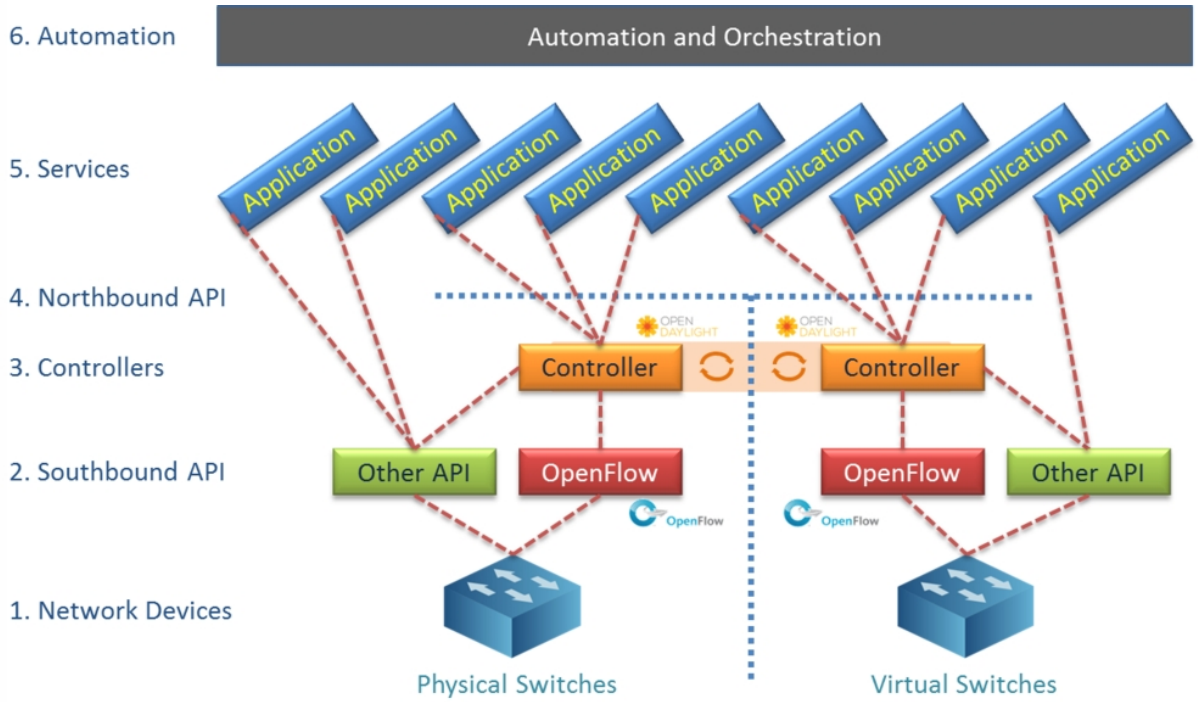
The SDN Controller is a logical entity that receives instructions or requirements from the SDN Application layer and relays them to the networking components. The controller also extracts information about the network from the hardware devices and communicates back to the SDN Applica with an abstract view of the network, including statistics and events about what is happening.

Example:
RYU, OpenDayLight, ONOS, FloodLight etc

SDN Applications:

SDN Applications are programs that communicate behaviors and needed resources with the SDN Controller via application programming interface (APIs). These applications could include networking management, analytics, or business applications used to run large data centers. This is the place for research, innovations, new ideas etc.

A Northbound interface is defined as the connection between the controller and applications.



---

3. SDN Test bed Installation

Setup the SDN Test environment to practice Openflow usecases with RYU SDN Controller.

There are two option.

- Download the prebuilt SDN Test bed VM image(ubuntu 20.04 desktop) and use it
- Setup the SDN Test bed by your self(Fresh Installation on ubuntu 20.04)

Option1: Prebuilt VM Image

This is the easiest option(for new comer).

Prebuilt VM Image (OVA Format), Size 3.3GB can be download from this below link,

https://drive.google.com/file/d/1_eC9O6EW7fh5YUp_LJMQ3_gIyFXBIc8k/view?usp=sharing

username : test password : test

This can be imported in **Oracle VirtualBox**. Just double click the OVA file it will be imported automatically.

- Follow this youtube video https://youtu.be/93lM4OLyytE
- https://docs.oracle.com/cd/E26217_01/E26796/html/qs-import-vm.html

This should be imported in **VMWARE** also.

Option2: Fresh Installation

If you want to setup the testbed by yourself follow this

Requirements:

OS: Ubuntu 20.04

CPU: 2 Cores +

RAM: 4GB +

HDD: 15GB+

If you are using Windows or other OS, you can install Ubuntu 20.04 as a Virtual Machine(VM) using Virtual Box .

You can download the ISO installer from the below link,

https://www.ubuntu.com/download/desktop

Please run the below commands in UBUNTU Terminal

```
sudo apt update
sudo apt install python3 python3-pip xterm iperf hping3 net-tools wireshark apache2-utils curl
sudo apt install mininet
sudo pip3 install ryu
sudo pip3 install mininet
sudo cp /usr/bin/python3 /usr/bin/python
ryu-manager --version
sudo mn --version
```

To check the python version:

> python3 --version or python --version

```
Python 3.8.5
```

To verify :

> ovs-vsctl --version

```
ovs-vsctl (Open vSwitch) 2.13.1
DB Schema 8.2.0
```

Testing

Open 4 Terminals:

1. In Terminal1,

```
sudo wireshark
```

And start the capture for "loopback" or "any" interface.

2. In Terminal2,

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

3. In Terminal3,

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

you will get the mininet prompt. In mininet prompt, type pingall comamnd

> pingall

Logs:

```
suresh@suresh-vm:~$ sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,4
*** Creating network
*** Adding controller
Connecting to remote controller at 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> 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>
```

4. In Terminal 4,

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

Logs:

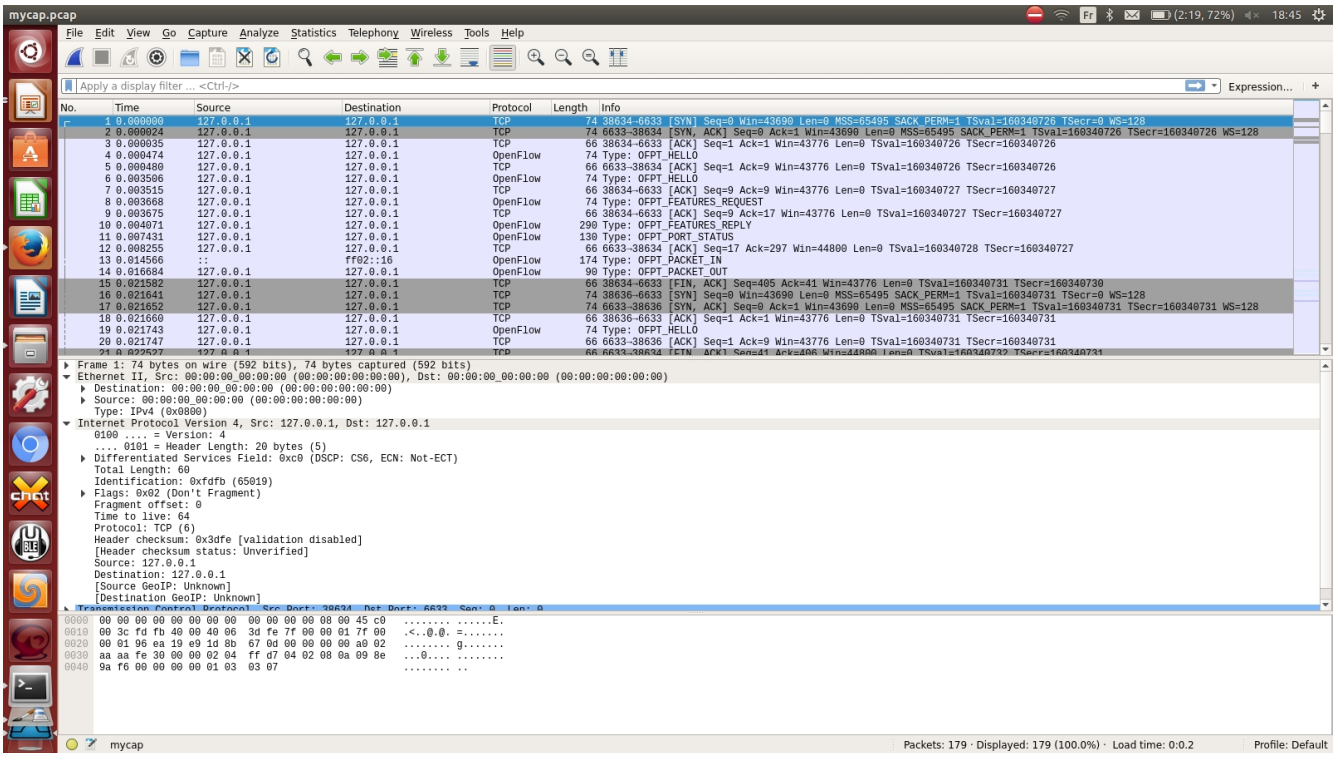
```
suresh@suresh-vm:~$ sudo ovs-vsctl show
[sudo] password for suresh:
a315e8b4-dd3f-42f6-b84a-e967e02660a4
    Bridge "s1"
        Controller "tcp:127.0.0.1:6653"
            is_connected: true
        Controller "ptcp:6654"
        fail_mode: secure
        Port "s1-eth4"
            Interface "s1-eth4"
        Port "s1"
            Interface "s1"
                type: internal
        Port "s1-eth1"
            Interface "s1-eth1"
        Port "s1-eth3"
            Interface "s1-eth3"
        Port "s1-eth2"
            Interface "s1-eth2"
    ovs_version: "2.13.1"
suresh@suresh-vm:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=115.430s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:
 cookie=0x0, duration=115.421s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:
 cookie=0x0, duration=115.410s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:
 cookie=0x0, duration=115.404s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:
 cookie=0x0, duration=115.391s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:
 cookie=0x0, duration=115.380s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:
 cookie=0x0, duration=115.370s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:
 cookie=0x0, duration=115.368s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:
 cookie=0x0, duration=115.361s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:
 cookie=0x0, duration=115.359s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:
 cookie=0x0, duration=115.346s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth4",dl_src=00:00:00:0
 cookie=0x0, duration=115.344s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:
```

```
 cookie=0x0, duration=164.572s, table=0, n_packets=58, n_bytes=4276, priority=0 actions=CONTROLLER:65535
suresh@suresh-vm:~$
```

5. check the openflow messages in wireshark

Stop the Wireshark capture,

In the filter type "openflow_v4" to see the OPENFLOW Messages.



---

4. Networking Concepts

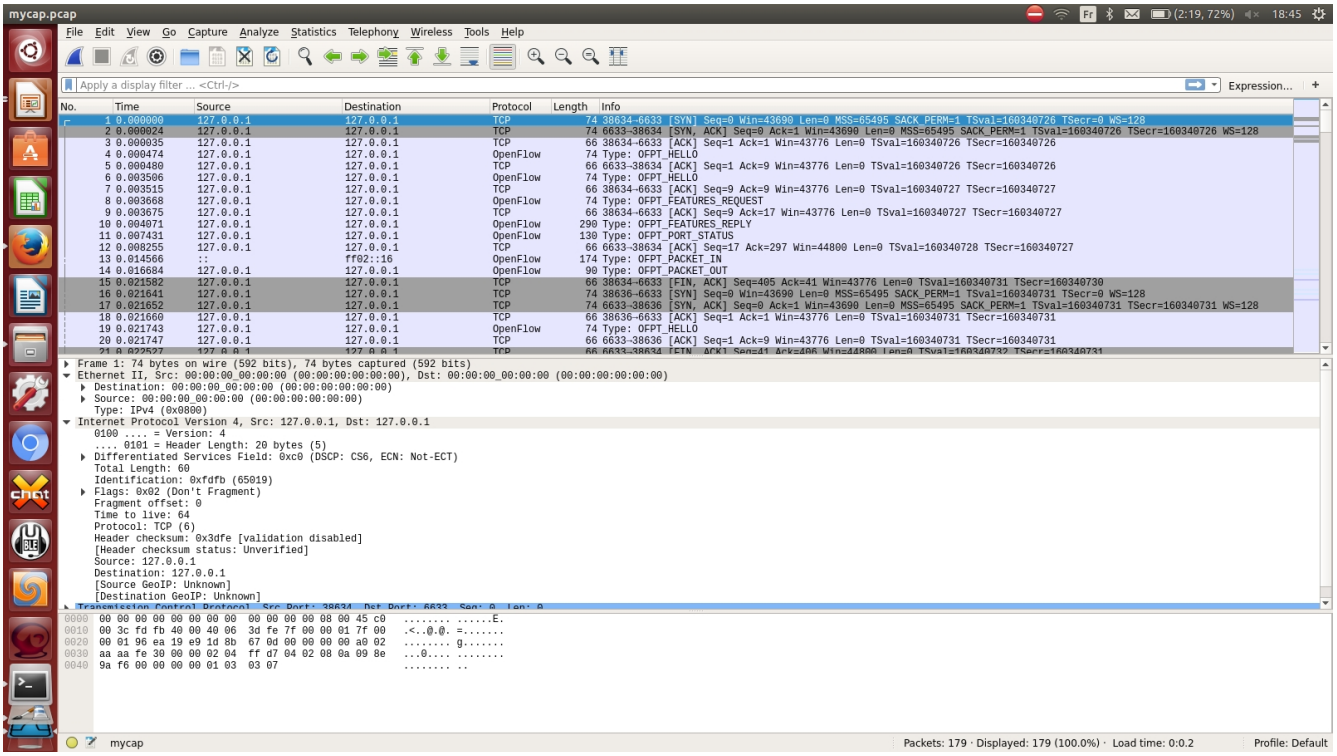
TCP/IP Layers

TCP/IP	OSI Model	Protocols
Application Layer	Application Layer	DNS, DHCP, FTP, HTTPS, IMAP, LDAP, NTP, POP3, RTP, RTSP, SSH, SIP, SMTP, SNMP, Telnet, TFTP
	Presentation Layer	JPEG, MIDI, MPEG, PICT, TIFF
	Session Layer	NetBIOS, NFS, PAP, SCP, SQL, ZIP
Transport Layer	Transport Layer	TCP, UDP
Internet Layer	Network Layer	ICMP, IGMP, IPsec, IPv4, IPv6, IPX, RIP
Link Layer	Data Link Layer	ARP, ATM, CDP, FDDI, Frame Relay, HDLC, MPLS, PPP, STP, Token Ring
	Physical Layer	Bluetooth, Ethernet, DSL, ISDN, 802.11 Wi-Fi

Network Packet Capture & Protocol Analyzer

Wireshark

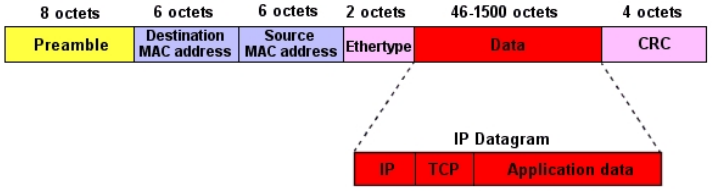
- Traffic Capture & Network protocol analyzer
- GUI
- Capture the Traffic from the interface
- Open the capture file and analyze it



Ethernet (Layer2) Concepts

- Ethernet is Layer2 Protocol,
- Majorly used as LAN connectivity
- Interface (eth0, eth1, ….)
- MAC Address is L2 Address, Associated with the interface, its 8 bytes (HWaddr 02:42:ac:11:00:02)

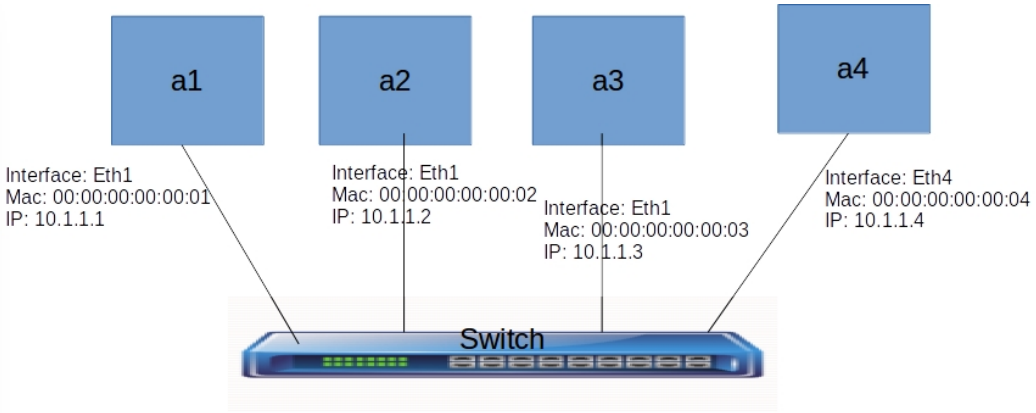
Ethernet Frame




```
Destination Mac address :  Target machine MAC address
Source Mac address:         Source machine MAC address
Ethertype:   Next layer protocol
                0X0800 -  IP
                0X0806 -  ARP
                0X86DD  - IPv6
```

Neighbour Discovery

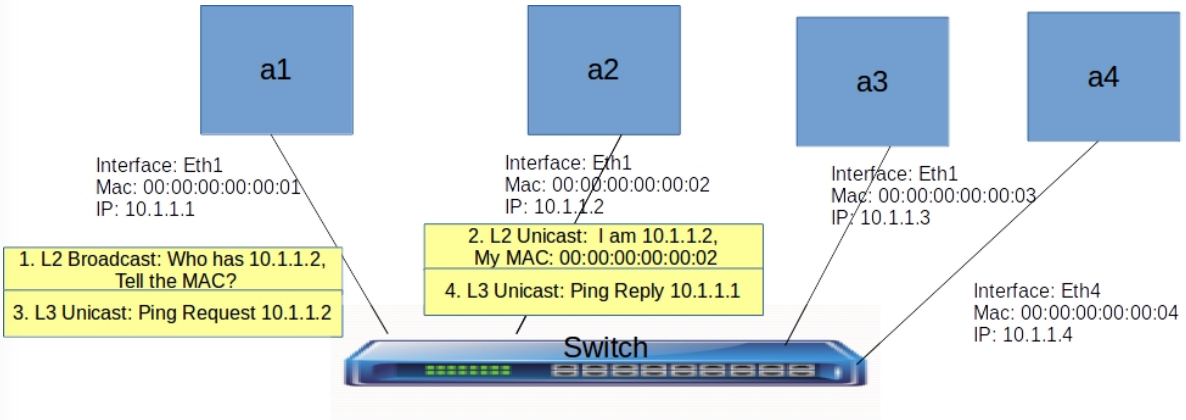
The MAC address is how machines on a subnet communicate. When machine A sends packets to another machine on its subnet, it sends it using the MAC address. When sending a packet to a machine on the public Internet, the packet is sent to the MAC address of the router interface that is the default gateway. IP addresses are used to figure out the MAC address to send to using ARP.



ARP Basics

ARP stands for Address Resolution Protocol. When you try to ping an IP address on your local network, say 192.168.1.1, your system has to turn the IP address 192.168.1.1 into a MAC address. This involves using ARP to resolve the address, hence its name.

Systems keep an ARP look-up table where they store information about what IP addresses are associated with what MAC addresses.



1. When trying to send a packet to an IP address, the system will first consult this table to see if it already knows the MAC address. If there is a value cached, ARP is not used.
2. If the IP address is not found in the ARP table, the system will then send a broadcast packet to the network using the ARP protocol to ask "who has 192.168.1.1".
3. Because it is a broadcast packet, it is sent to a special MAC address that causes all machines on the network to receive it.
4. Any machine with the requested IP address will reply with an ARP packet that says "I am 192.168.1.1", and this includes the MAC address which can receive packets for that IP.

ARP Table

List ARP Table

```
arp -a
```

Delete ARP entry

```
arp -d 10.1.1.2
```

ARP Demo

1. Start the mininet topology (linear topology , not connected to SDN Controller)

```
sudo python traditional_switch.py
```

2. in mininet console, run 'links' command to identify the link names

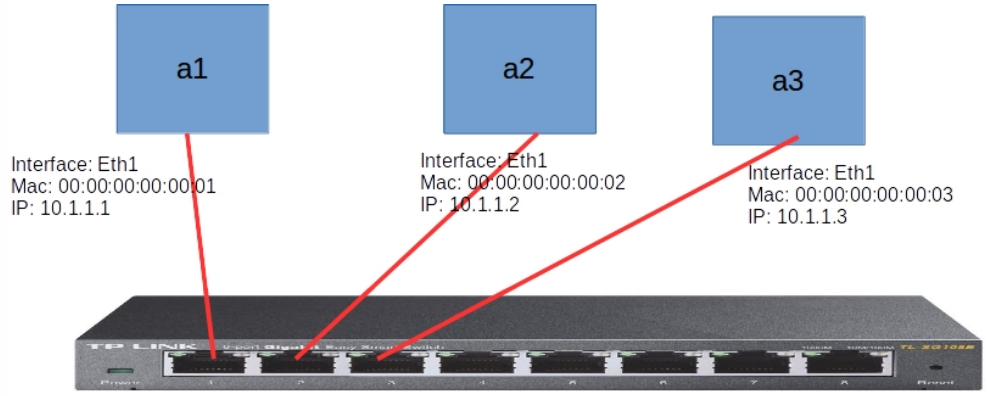
```
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s1-eth2 (OK OK)
h3-eth0<->s1-eth3 (OK OK)
h4-eth0<->s1-eth4 (OK OK)
mininet>
```

6. check the ARP entries in h1 and h2

```
mininet>h1 arp -a
```

Traditional L2 Switch

A network switch (MAC bridge) is a computer networking device that connects devices together on a computer network by using packet switching to receive, process, and forward data to the destination device.



Connections:
PORT1 – Connected to a1
PORT2 – Connected to a2
PORT3 - Connected to a3

MAC Table:
PORT1 – 00:00:00:00:00:01
PORT2 – 00:00:00:00:00:02
PORT3 - 00:00:00:00:00:03

Control Plane

Learns the MAC Address from the incoming packet and populate the MAC Table

Data Plane

- Receives the Packet
- Read the Destination MAC from the Packet
- Look up the MAC Table for the destination Port
- Forward the Packet to the destination Port

Openvswitch is softswitch, which works as normal switch(traditional) as well as SDN(openflow)switch

Demo

Same steps as Neighbour discovery Demo

Lets run OVS commands to check the switch

```
sudo ovs-vsctl   --version
sudo ovs-vsctl show
sudo ovs-appctl fdb/show s1
sudo ovs-dpctl dump-flows
```

How it works

1. Traditional Switch have built in Control Plane + Data Plane.
2. Mac Table(a.k.a Forwarding table) is Empty when the switch starts.

3. Control Plane updates the Mac Table with the MAC and the PORT Number. This information is extracted from incoming Packet.
4. Control Plane keeps on building/updating the Mac Table.
5. When the packet arrives, Switch Data plane looks the Mac table, if the destination MAC matches in the Mac table, it forwards the packet to the respective Port.

SDN Switch(Openflow Switch)

1. Run the Linear Mininet Topology,

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

2. Start Wireshark Capture

3. RYU L3 Application

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

4. In the mininet cli, ping h1 to h2.

```
mininet>h1 ping h2
```

5. Check the OpenFlow flows

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

How it works

1. Switch is configured with SDN Controller IP and Openflow protocol version.
2. Switch establishes the communication with SDN Controller.
3. SDN Controller installs the default Openflow rule (TABLE MISS ENTRY) in the switch Flow table.
4. TABLE MISS Entry openflow rule matches with all the packets and send it to the CONTROLLER. The priority is lowest in the table(0)
5. When the Host Data Packet arrives in the Switch, It will be matched with TABLE MISS ENTRY , and the packet will be sent it to Controller (PACKET IN Message)
6. Controller receives the packet, and build the Switch Logic with the packets.
7. Controller adds the OPENFLOW flows to the switch.
8. Now, Switch data path is built with flows. So next time, when the Packet arrives it will be matched with the Flow table and forward the packet to respective port.

---

5. Mininet

Basic Operations

1. To check the mininet version

```
mn --version
```

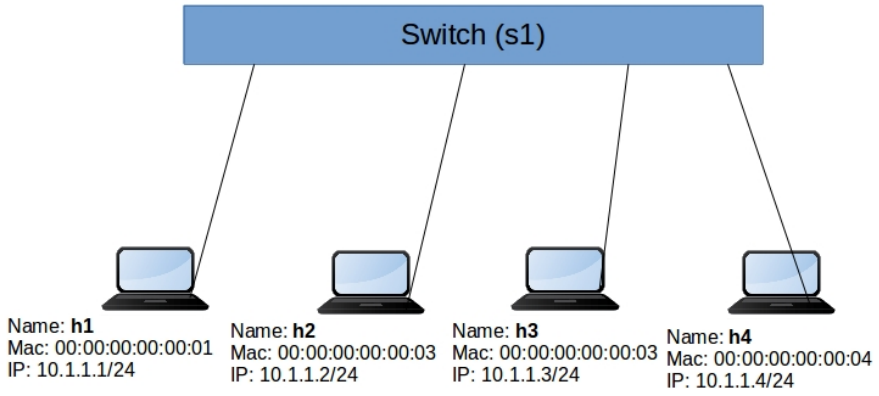
2. To clean up the existing ovs bridges and namespaces

Note: sometime we mistakenly closed the mininet shell, or mininet crashed. But the topology components will continue to exists. To clean such stuff, cleanup command is used.

```
mn -c
```

3. Our First Topology (Single)

Topology with Single Switch and 4 Nodes.



RYU SDN Controller

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

Mininet Topology

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

options	Description
--controller	type of controller local/remote and remote controller ip.
--mac	mac address starts with 00:00:00:00:00:01
-i	IP Subnets for the Topology
--switch	Switch type (ovsk - openvswitch kernel module), and openflow version.
--topo	topology type(linear,minimal,reversed,single,torus,tree) and params.

Once you are done, make sure "exit" the mininet shell,

> exit

Example:

```
mininet> exit
*** Stopping 1 controllers
c0
*** Stopping 4 links
....
*** Stopping 1 switches
s1
*** Stopping 4 hosts
h1 h2 h3 h4
*** Done
completed in 2.604 seconds
suresh@suresh-vm:~$
```

4. Mininet Basic Shell Commands

Informative commands

```
help
dump
net
links
```

Action commands

```
pingall
```

Execute the commands in HOST/Node:

Option1:

we can login to each host using 'xterm' command

> xterm h1

```
mininet>xterm h1
```

It will open a xterm terminal for the host. Now we can execute the command inside that terminal

Option2:

we can directly execute from the mininet shell.

```
mininet><hostname> command
```

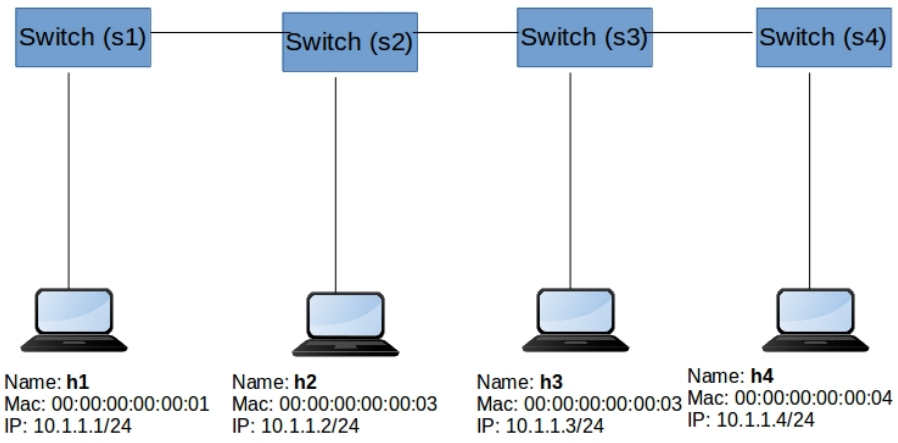
Example:

```
mininet>h1 ifconfig
mininet>h1 ping h2
mininet>h1 ip route
```

we can use either method to run the Traffic tests or executing the commands.

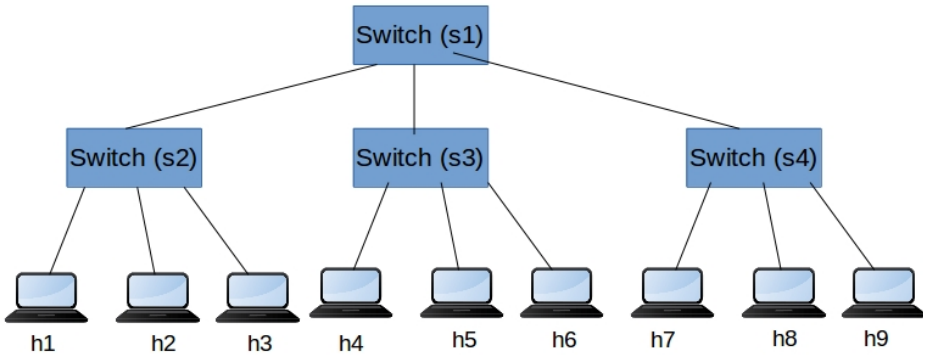
5. Linear Topology

linear topology (where each switch has one host, and all switches connect in a line)



6. Tree Topology

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




```
sudo mn --controller=remote,ip=127.0.0.1 --mac -i 10.1.1.0/24 --topo=tree,depth=2,fanout=3
```

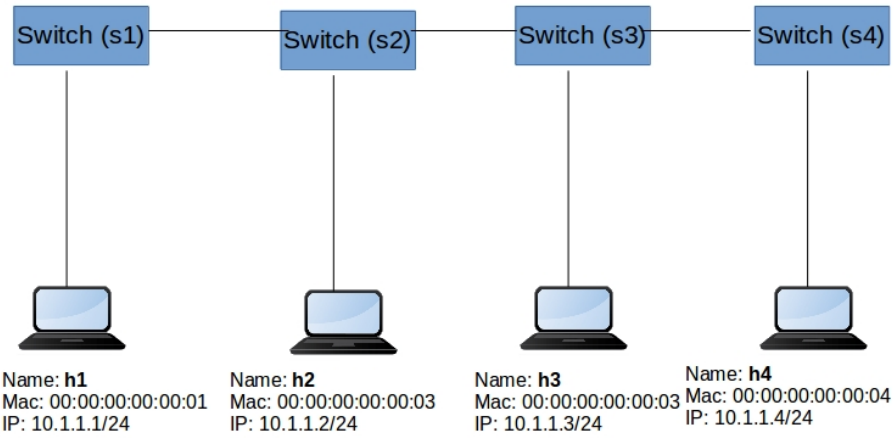
fanout : each switch is connected to these many childs depth : depth of the tree

---

Running Traffic Tests

1. TCP/UDP Traffic Tests

a. Setup the Topology with xterms options (open terminal for each Node)




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

b. TCP Traffic Test between h1 to h4

Run IPERF TCP Server in h4

```
iperf -s
```

-s means server mode

RUN IPERF TCP Client in h1

```
iperf -c 10.1.1.4 -i 10 -t 30
iperf -c 10.1.1.4 -i 10 -b 10m -t 30
iperf -c 10.1.1.4 -i 10 -P 10 -t 30
```

-b means bandwidth 10m means 10Mbps

-P means parallel connections

c. UDP Traffic Test between h1 to h4

Run IPERF UDP Server in h4

```
iperf -u -s
```

-u means udp

RUN IPERF UDP Client in h1

```
iperf -u -c 10.1.1.4 -b 10m -i 10 -t 30
iperf -u -c 10.1.1.4 -b 10m -i 10 -P 10 -t 30
```

-b means bandwidth 10m means 10Mbps

HTTP Traffic Tests

Run Python Simple Web Server in h4

```
python -m SimpleHTTPServer 80
```

From H1, Access the Web server

```
curl http://10.1.1.4/
```

curl utility used as web client to access the web server.

If we want to simulate the 1000s users accessing the web server on the same time (load), we can use ab(apache bench) tool

```
ab -n 500 -c 50 http://10.1.1.4/
```

-c 50 means parallel request per second (50 Requests per second)

-n 500 means total request for this test (500 requests)

Apache bench tool have lot more options , More details can be found from the below link

https://httpd.apache.org/docs/2.4/programs/ab.html

Writing Custom Topology in Mininet

mininet exposes the python API. We can create a custom topologies using the python API with few lines of code.

How to write Custom Topology in Mininet

Steps are below.

1. Import the python required libraries

2.

```
from mininet.topo import Topo
from mininet.net import Mininet

class SingleSwitchTopo(Topo):
   def build(self):
       s1 = self.addSwitch('s1')

       h1 = self.addHost('h1')
       h2 = self.addHost('h2')
       h3 = self.addHost('h3')
       h4 = self.addHost('h4')

       self.addLink(h1, s1)
       self.addLink(h2, s1)
       self.addLink(h3, s1)
       self.addLink(h4, s1)
```

Important Topology definition APIs:

```
addSwitch
addHost
addLink
```

3. Run the Topology as below,

- Create the Topology object
- Create the Mininet with Topology object
- Start the Mininet

```
if __name__ == '__main__':
    topo = SingleSwitchTopo()
    c1 = RemoteController('c1', ip='127.0.0.1')
    net = Mininet(topo=topo, controller=c1)
    net.start()
```

How to run

1. start the RYU SDN Controller

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

2. Run the Mininet topology file

```
sudo python <topology file name>
```

3. Perform your tests/operation etc.

---

6. Openflow Theory

This document text/diagram is copied from the Openflow 1.3 specifications

https://www.opennetworking.org/software-defined-standards/specifications/

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

An OpenFlow Logical Switch consists of one or more flow tables and a group table, which perform packet lookups and forwarding, and one or more OpenFlow channels to an external controller (Figure 1).

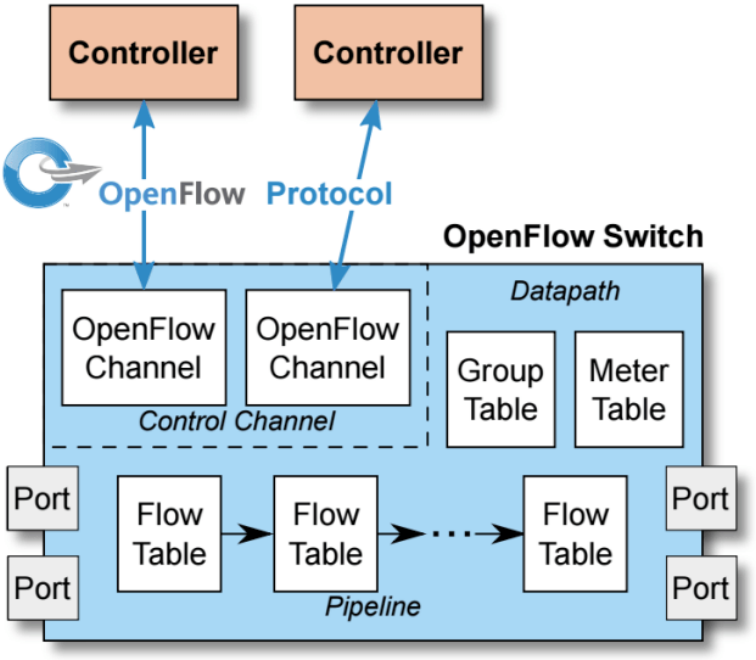


Figure 1: Main components of an OpenFlow switch.

- 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.
- 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.

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
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,
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
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
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
 cookie=0x0, duration=3.906s, table=0, n_packets=192421, n_bytes=12699810, priority=1,tcp,nw_src=192.168.1.1,nw_dst=19
 cookie=0x0, duration=31.495s, table=0, n_packets=43, n_bytes=4309, priority=0 actions=CONTROLLER:65535
suresh@suresh-vm:~$
```

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 Matching

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

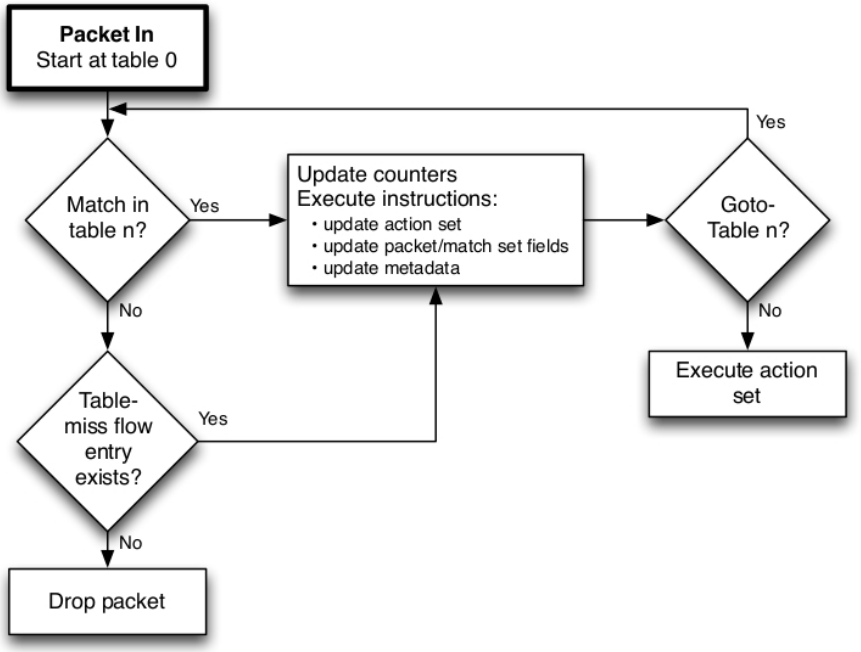


Figure 3: Flowchart detailing packet flow through an OpenFlow switch.

The switch starts by performing a table lookup in the first flow table, and based on pipeline processing, may perform table lookups in other flow tables

Match Fields

Reference : Page 62 of the specification

```
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. */
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 */
```

- 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 : Page 66.

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.

Masking example:

You can match the Source IP against with Subnet. Refer page 64 for detailed table.

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

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.

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)

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=
cookie=0x0, duration=7.605s, table=0, n_packets=2, n_bytes=140, idle_timeout=10, hard_timeout=30, priority=1,in_port=
cookie=0x0, duration=8.652s, table=0, n_packets=33, n_bytes=3527, priority=0 actions=CONTROLLER:65535
```

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 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

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

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]
```

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.

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.

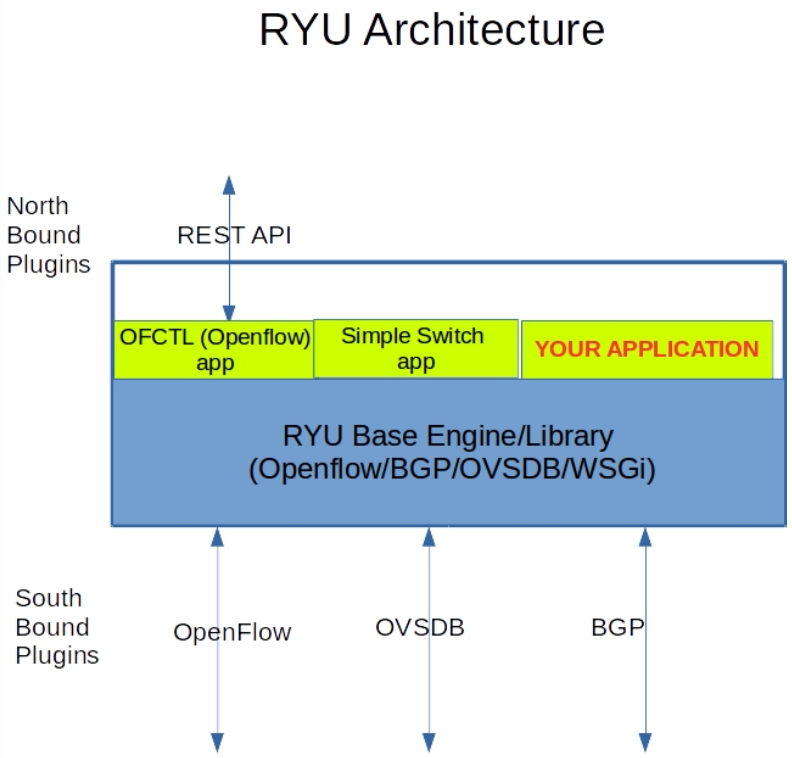
4) Our Controller Application(our RYU project/exercise) should use Flow Stats, Port Stats request message to get the statistics(Packets Sent/Received , etc) of the flows, Ports .

---

7. 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.



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-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/python
suresh@suresh-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

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
```

To use topology discovery

> ryu-manager --observe-links application_name

Example:

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

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.

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
```

```
suresh@suresh-vm:~/sdn$ sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=singl
[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>
```

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$
```

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$
```

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)

Basic Openvswitch Commands

```
ovs-vsctl show

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

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

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

```
suresh@suresh-vm:~/sdn$ sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=singl
[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>
```

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
```

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>
```

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=00:00:00:00:00:0
 cookie=0x0, duration=35.140s, table=0, n_packets=4, n_bytes=336, priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:0
 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"
```

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=10.0.0.1,nw_dst=10.0.0.2 actions
 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
 cookie=0x0, duration=17.720s, table=0, n_packets=10, n_bytes=644, priority=0 actions=CONTROLLER:65535
suresh@suresh-vm:~/sdn$
```

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"
```

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 &
```

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 acti
 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 acti
 cookie=0x0, duration=85.890s, table=0, n_packets=773576, n_bytes=35217018256, priority=1,tcp,nw_src=10.0.0.1,nw_dst=1
 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
 cookie=0x0, duration=137.765s, table=0, n_packets=30, n_bytes=1996, priority=0 actions=CONTROLLER:65535
suresh@suresh-vm:~/sdn$
```

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"

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
```

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="
 cookie=0x0, duration=1.670s, table=0, n_packets=1, n_bytes=98, idle_timeout=10, hard_timeout=30, priority=1,in_port="
 cookie=0x0, duration=1.660s, table=0, n_packets=1, n_bytes=98, idle_timeout=10, hard_timeout=30, priority=1,in_port="
 cookie=0x0, duration=1.655s, table=0, n_packets=1, n_bytes=98, idle_timeout=10, hard_timeout=30, priority=1,in_port="
 cookie=0x0, duration=1.646s, table=0, n_packets=1, n_bytes=98, idle_timeout=10, hard_timeout=30, priority=1,in_port="
```

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
```

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

2. Run the RYU SDN Controller with flowtimeout application.

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

3. Check the Switch mode

```
sudo ovs-vsctl show
```

4. Do continuous ping from h1 to h2

```
mininet> h1 ping h2
```

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.

---

8. RYU Programming - Basics

Ryu applications are just Python scripts so you can save the file as (.py) on any location.

Components

ryu-manager :

- main executable

ryu.base.app_manager:

- The central management of Ryu applications
- Load Ryu applications
- Provide contexts to Ryu applications

ryu.ofproto:

- OpenFlow wire protocol encoder and decoder:

ryu.controller:

- openflow controller implementation
- generates openflow events

ryu.packet:

- all packet parsing libraries

Events:

RYU Application works based on Events. RYU Controller emits the events for the Openflow Messages received. This can be handled by the RYU Applicaltions.

Example Events:

ofp_event.EventOFPSwitchFeatures ofp_event.EventOFPPacketIn ofp_event.EventOFPFlowStatsReply

https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html

Simple Switch application is a base application to refer. https://github.com/osrg/ryu/blob/master/ryu/app/simple_switch_13.py

---

Closer look on Simple_Switch_13 application

Lets have quick Recap on the OpenFlow Message transcations between Controller and Switch.

1. Hello Message
2. Feature Request/Response Message
3. Flow Modification message to install Table Miss Entry
4. Packet In Message
5. Packet Out Message
6. Flow Modification Message to Install the Flows

we can classify the program in few important parts.

1.Import the base classes / library

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
```

```
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types
```

2. Application Class (derived from app_manager)

```
class SimpleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
```

3. Important openflow events handled

Features_Response_Received

PacketIn_Message_Received

we can handle this event, in our application as below,

```
    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        .....skipped......


    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        # If you hit this you might want to increase
        # the "miss_send_length" of your switch
        if ev.msg.msg_len < ev.msg.total_len:
            self.logger.debug("packet truncated: only %s of %s bytes",
                              ev.msg.msg_len, ev.msg.total_len)

        .....skipped......
```

In the switch_features_handler event, we add the TABLE MISS Entry . So we get the packet in messages.

In the packet_in_handler event, we process the PACKETs and parse the src_mac and dst_mac address and build the switching logic and install the flow.

Inserting(Adding) a flow

Flow table consists of Match, Actions and Counters.

A. Create a Match

Reference:

https://github.com/osrg/ryu/blob/master/ryu/ofproto/ofproto_v1_2_parser.py#L3403

This match is with no match fields. it means matching all the packets.

```
    match = parser.OFPMatch()
```

This matches with in_port, eth_dst and eth_src field.

```
    match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
```

B. Create a Actions

https://github.com/osrg/ryu/blob/master/ryu/ofproto/ofproto_v1_2_parser.py#L1252

Below action is send it to CONTROLLER(Reserved port).

```
actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,ofproto.OFPCML_NO_BUFFER)
```

Below action , send it to port number 1.

```
out_port = 1
actions = [parser.OFPActionOutput(out_port)]
```

C. Create a Instruction list with Actions

https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html?#ryu.ofproto.ofproto_v1_3_parser.OFPInstructionActions

```
inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,actions)]
```

D. Send OFP Flow Modifiation meassage for creating a new flow

https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html?#ryu.ofproto.ofproto_v1_3_parser.OFPFlowMod

Most of the parameter are default. table_id, timeouts, cookies etc.

```
mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                priority=priority, match=match,
                                instructions=inst)
```

In Below example, explicitly specify tableid, timeouts.

```
mod = parser.OFPFlowMod(datapath=datapath, table_id=10,
                        buffer_id=buffer_id,idle_timeout=10,
                        hard_timeout=30, priority=priority,
                        match=match, instructions=inst)
```

---

9. Developing Switching Applications

we use simple_switch_13 application as base application, and will build hub, L3, L4 Match applications

---

HUB Application

Logic

Create a Flow, all Matches with FLOOD Action

No need of TABLE MISS Entry.

So, we can simply modify the TABLE MISS Entry action as FLOOD to achieve the HUB Operations.

Code changes

In the Switch Features handler, modify the action as FLOOD.

```
actions = [parser.OFPActionOutput(port=ofproto.OFPP_FLOOD)]
```

We will never get Packet_in event , So we can remove those routines.

Save this file as hub.py

Demo

1. Run Mininet topology

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

2. Run RYU hub application

```
ryu-manager hub.py
```

3. Check the openvswitch flows

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

4. do pingall from mininet.

Switch Application With L3 Match

Logic

- simple_switch_13 application as a base application and switch learning process is same
- But Flow will be based on Layer3 Match (src ip and destination ip) instead of src_mac and dst_mac

Code changes

1. include the IP library

```
from ryu.lib.packet import ipv4
```

2. Populate the Match based on IP.

- Check the packet is IP Packet, then decode the srcip and dstip from the packet header
- Populate the Match based on srcip and dstip.

```
    # check IP Protocol and create a match for IP
    if eth.ethertype == ether_types.ETH_TYPE_IP:
        ip = pkt.get_protocol(ipv4.ipv4)
        srcip = ip.src
        dstip = ip.dst
        match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP,
                                ipv4_src=srcip,
                                ipv4_dst=dstip
                                )
```

Demo

1. Run Mininet topology

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

2. Run RYU hub application

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

3. do pingall from mininet.

4. Check the openvswitch flows

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

Switch Application With L4 Match

Logic

- simple_switch_13 application as a base application and switch learning process is same
- But Flow will be based on Layer4 Match (src ip and destination ip, protocol and port number) instead of src_mac and dst_mac
- In the ethernet packet, check the ether frame type field is IP.
- if it is IP, load the IP Packet from the packet, extract the srcip , and dst ip from IP Packet.
- Check the IP Protocol(ICMP or TCP or UDP)
- If it is ICMP Prepare the openflow match with IP Src, IP dst and Protocol.
- If it is TCP, extract the tcp src port and tcp dst port fied Prepare the openflow match with IP Src, IP dst and Protocol, TCP Src Port and TCP dst port..
- If it is UDP Extract the udp src port and tcp dst port fied Prepare the openflow match with IP src, IP dst , protcol, udp src and udp dst port.

Code changes

1. include the header

```
from ryu.lib.packet import in_proto
from ryu.lib.packet import ipv4
from ryu.lib.packet import icmp
from ryu.lib.packet import tcp
from ryu.lib.packet import udp
```

2. Populate the Match

```
            # check IP Protocol and create a match for IP
            if eth.ethertype == ether_types.ETH_TYPE_IP:
                ip = pkt.get_protocol(ipv4.ipv4)
                srcip = ip.src
                dstip = ip.dst
                protocol = ip.proto

                # if ICMP Protocol
                if protocol == in_proto.IPPROTO_ICMP:
                    match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP, ipv4_src=srcip, ipv4_dst=dstip, ip_proto

                #  if TCP Protocol
                elif protocol == in_proto.IPPROTO_TCP:
                    t = pkt.get_protocol(tcp.tcp)
                    match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP, ipv4_src=srcip, ipv4_dst=dstip, ip_proto

                #  If UDP Protocol
                elif protocol == in_proto.IPPROTO_UDP:
                    u = pkt.get_protocol(udp.udp)
                    match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP, ipv4_src=srcip, ipv4_dst=dstip, ip_proto
```

Demo

1. Run Mininet topology

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

2. Run RYU hub application

```
ryu-manager l4_switch.py
```

3. do pingall from mininet.

4. Check the openvswitch flows

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

Flow Timeout

Idle Timeout:

If the flow is Idle(no packets hitting this flow) for the specified time(Idle Time), the flow will be expired.

Hard Timeout:

Flow entry must be expired in the specified number of seconds regardless of whether or not packets are hitting the entry

In simple_switch.py, the default timeout values(for idle and hard) are not set(0). it means, the flows are permanent. it will never expiry.

Code changes

1. In the add_flow function, include idle, hard parameters with default value as 0.
2. OFPFlowMod API include the idle_timeout,hard_timeout parameters.

```
    mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                idle_timeout=idle, hard_timeout=hard, priority=priority, match=match,
                                instructions=inst)
```

3. Call the add_flow function with idle, hard value

Demo

1. Run Mininet topology

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

2. Run RYU hub application

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

3. do pingall from mininet.
4. Check the openvswitch flows

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

verify the timeout values in the flow,

5. Check the flows again after the timeout seconds.

---

10. Statistics Collection

Openflow protocol provides extensive statistics, such as,

- flow statistics
- flow aggregate statistics
- port statistics
- group table statistics
- meter statistics

User can collect these statistics from the openflow switch and use it(its used for many applications)

User needs to send the statistics request meassge(Example: OpenFlow FlowStatistics Request Message). Switch reply with statistics respone message.

In this tutorial, we are going to write a RYU Code to display the statistics.

RYU Thread API

RYU comes with inbuilt thread implementation(hub library.)

Import the library

```
from ryu.lib import hub

    def myfunction(self):
        self.logger.info(" started new thread")
        i = 0
        while True:
            hub.sleep(30)
            self.logger.info("printing %d", i)
            i = i + 1
```

Note: its a continuous run, so we use hub.sleep() to sleep for a time.

To initiate a thread

```
hub.spawn(self.myfunction)
```

Demo

> ryu-manager thread.py

Example:

```
$ ryu-manager thread1.py
loading app thread1.py
loading app ryu.controller.ofp_handler
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app thread1.py of SimpleSwitch13
started new thread
printing 0
printing 1
printing 2
```

Flow Statistcs

Measure the flows statistics or utilization in regular interval(10second) and print it.

Logic

- Use Ryu Thread(HUB) mechanism,
- In the Thread, Send Openflow Flow Statistics Request message at regular interval
- Flow Statistics Response will be received as event, and display the bytes & packets counter values.

Coding

1. Use Simple Switch application as base application

2. we require datapaths object of all the switches for generating the stats request message. hence we store the datapaths in the dictionary.

In the init routine, we declare it.

```
self.datapaths = {}
```

In the switch features handler function, we store the the datapath object in the datapaths dictionary.

```
self.datapaths[datapath.id] = datapath
```

3. Include RYU Thread library

```
from ryu.lib import hub
```

3. Start the thread in the init routine

```
self.monitor_thread = hub.spawn(self.monitor)
```

The thread function to be placed in side the RYU manager application class.

In the thread function, we process the datapaths(switches) dictionary, and generate the flowstats request message(without any match filter). It means we query the stats of all the flows in the switch.

```
 def monitor(self):
    self.logger.info("start flow monitoring thread")
    while True:
        hub.sleep(10)
        for datapath in self.datapaths.values():
            ofp = datapath.ofproto
            ofp_parser = datapath.ofproto_parser
            req = ofp_parser.OFPFlowStatsRequest(datapath)
            datapath.send_msg(req)
```

Refer:

https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html#ryu.ofproto.ofproto_v1_3_parser.OFPFlowStatsRequest

5. Flow Stats Response handling routine.

```
@set_ev_cls([ofp_event.EventOFPFlowStatsReply,
                ], MAIN_DISPATCHER)
    def stats_reply_handler(self, ev):
        for stat in ev.msg.body:
            self.logger.info("Flow details:  %s ",stat)
            self.logger.info("byte_count:  %d ", stat.byte_count)
            self.logger.info("packet_count:  %d ", stat.packet_count)
```

Reference:

https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html#ryu.ofproto.ofproto_v1_3_parser.OFPFlowStatsReply

Demo

1. start the mininet single topology

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

2. run our application.

```
ryu-manager flow_stats.py
```

3. ping h1 to h2 in mininet

```
mininet> h1 ping h2
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=64 time=6.33 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=64 time=0.501 ms
64 bytes from 10.1.1.2: icmp_seq=3 ttl=64 time=0.112 ms
```

4. In the ryu console, we can see the output

```
Flow details:  OFPFlowStats(byte_count=18480,cookie=0,duration_nsec=20000000,duration_sec=206,flags=0,hard_timeout=0,i
byte_count:  18480
packet_count:  192
```

Aggregate Flow Stats

Measure Total number flows in regular interval(10second) and print it.

Logic

- Use Ryu Thread(HUB) mechanism,
- In the Thread, Send Openflow Aggregate Flow Statistics Request message at regular interval
- Aggregate Flow Statistics Response will be received as event, and display the bytes & packets counter values.

Coding same as flow statistics example for thread stuff. But we send Aggregate flow stats request.

```
    def monitor(self):
        self.logger.info("start flow monitoring thread")
        while True:
            hub.sleep(10)
            for datapath in self.datapaths.values():
                ofp = datapath.ofproto
                ofp_parser = datapath.ofproto_parser
                cookie = cookie_mask = 0
                match = ofp_parser.OFPMatch()
                req = ofp_parser.OFPAggregateStatsRequest(datapath, 0,
                                                  ofp.OFPTT_ALL,
                                                  ofp.OFPP_ANY,
                                                  ofp.OFPG_ANY,
                                                  cookie, cookie_mask,
                                                  match)
                datapath.send_msg(req)
```

Reference:

https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html#ryu.ofproto.ofproto_v1_3_parser.OFPAggregateStatsRequest

Aggregate Flow Stats Response handling routine.

```
    @set_ev_cls([ofp_event.EventOFPAggregateStatsReply,
                 ], MAIN_DISPATCHER)
    def stats_reply_handler(self, ev):
        result = ev.msg.body
        self.logger.info('AggregateFlowStats  %s', result)
        self.logger.info('FlowCount  %d', result.flow_count)
```

Reference:

https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html#ryu.ofproto.ofproto_v1_3_parser.OFPAggregateStatsReply

demo

1. start the mininet single topology

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

2. run our application.

```
ryu-manager agg_flow_stats.py
```

3. ping h1 to h2 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>
```

4. In the ryu console, we can see the output

Result:

```
AggregateFlowStats  OFPAggregateStats(packet_count=296,byte_count=24752,flow_count=13)
FlowCount  13
```

Port Statistics

This is similar to previous two exercises

we should use port stats request message.

```
req = ofp_parser.OFPPortStatsRequest(datapath)
```

For Port Stats response,

```
    @set_ev_cls([ofp_event.EventOFPPortStatsReply,
                 ], MAIN_DISPATCHER)
    def stats_reply_handler(self, ev):
        for stat in ev.msg.body:
            self.logger.info("Port statistics :  %s ", stat)
```

Demo same procedure as last two exercises

---

11. Configuration File

At various sceanarios, user need to provide input to the Application instead of hardcoded in the program. For example, user can provide INTERVAL interval in the statistics applications.

RYU supports configuration file input, which will be parsed by the RYU Application and we can use it as a variable.

Configuration File

step1:

Enter the configuration in a file as below format,

> cat params.conf

```
[DEFAULT]
INTERVAL = 10
```

Here we specificed the INTERVAL as 10.

step2:

In the Ryu application,

Include the library for configuration apis

```
from ryu import cfg
```

Read the Configuration in the init routine of the application

step3:

When you run the ryu application, you should use --config-file filename options as below,

```
self.CONF = cfg.CONF
self.CONF.register_opts([
            cfg.IntOpt('INTERVAL', default=10, help = ('Monitoring Interval'))]
            )
self.logger.info("Interval value %d ",self.CONF.INTERVAL)
```

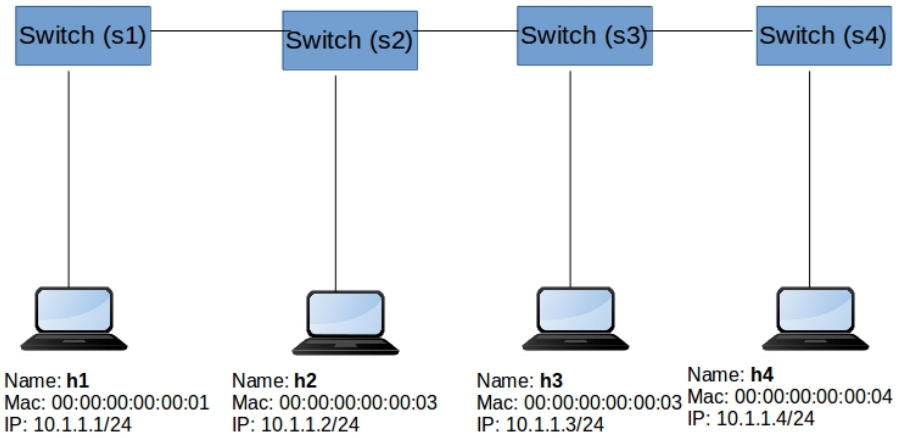
Repeat the same demo flow statistics procedure.

---

12. Group Table

Please refer the group table theory/concepts from PART2 . In this guide, we are going to see, programming methods/api for for group table.

Sniffer Program



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

Logic

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

Proactively We create group tables and flows in S2.

Coding

Use simple_switch_13 as base code

1. Creat a group table create function.

```
    def send_group_mod(self, datapath):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # Hardcoding the stuff, as we already know the topology diagram.
        # Group table1
        # Receiver port2, forward it to port1 and Port3
```

```
        actions1 = [parser.OFPActionOutput(1)]
        actions2 = [parser.OFPActionOutput(3)]
        buckets = [parser.OFPBucket(actions=actions1),
                   parser.OFPBucket(actions=actions2)]
        req = parser.OFPGroupMod(datapath, ofproto.OFPGC_ADD,
                                 ofproto.OFPGT_ALL, 50, buckets)
        datapath.send_msg(req)

        # Group table2
        # Receive Port3, forward it to port1 and Port2
        actions1 = [parser.OFPActionOutput(1)]
        actions2 = [parser.OFPActionOutput(2)]
        buckets = [parser.OFPBucket(actions=actions1),
                   parser.OFPBucket(actions=actions2)]
        req = parser.OFPGroupMod(datapath, ofproto.OFPGC_ADD,
                                 ofproto.OFPGT_ALL, 51, buckets)
        datapath.send_msg(req)
```

Here we use OFPGroupMod API for creation groups. refer the API for more details https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html#ryu.ofproto.ofproto_v1_3_parser.OFPGroupMod

2. In Switch features handler function, call the Group table creation function and add flows for switch S2,

```
    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        .........skipped......
        .........skipped......
        # switch s2
        if datapath.id == 2:
            # add group tables
            self.send_group_mod(datapath)
            actions = [parser.OFPActionGroup(group_id=50)]
            match = parser.OFPMatch(in_port=2)
            self.add_flow(datapath, 10, match, actions)
            # entry 2
            actions = [parser.OFPActionGroup(group_id=51)]
            match = parser.OFPMatch(in_port=3)
            self.add_flow(datapath, 10, match, actions)
```

Demo

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 sniffer.py
```

3. Do pingall from mininiet

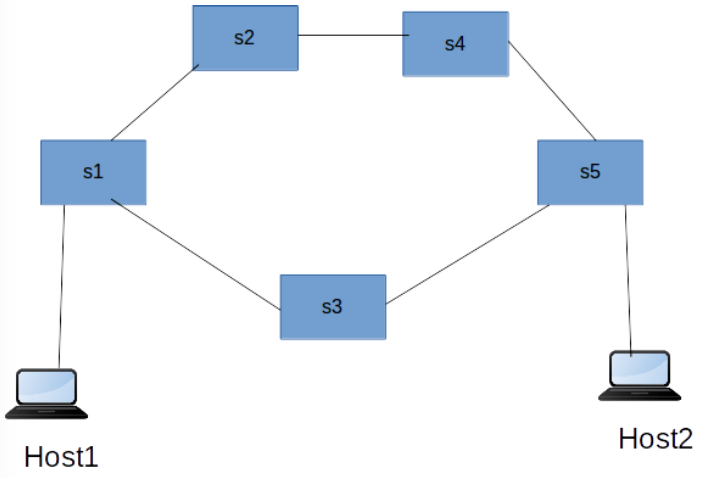
```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet>
```

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
sudo ovs-ofctl -O OpenFlow13 dump-group-stats s2
```

Loadbalancer

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



Logic

In Switch 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

In Switch 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

Proactively We create group tables and flows in S1 and S5.

Coding

Use simple_switch_13 as base code

1. Creat a group table create function.

```
    def send_group_mod(self, datapath):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # Hardcoding the stuff, as we already know the topology diagram.
        # Group table1
        # Receiver port3 (host connected), forward it to port1(switch) and Port2(switch)
        LB_WEIGHT1 = 30 #percentage
        LB_WEIGHT2 = 70 #percentage

        watch_port = ofproto_v1_3.OFPP_ANY
        watch_group = ofproto_v1_3.OFPQ_ALL

        actions1 = [parser.OFPActionOutput(1)]
        actions2 = [parser.OFPActionOutput(2)]
        buckets = [parser.OFPBucket(LB_WEIGHT1, watch_port, watch_group, actions=actions1),
                   parser.OFPBucket(LB_WEIGHT2, watch_port, watch_group, actions=actions2)]

        req = parser.OFPGroupMod(datapath, ofproto.OFPGC_ADD,
                                 ofproto.OFPGT_SELECT, 50, buckets)
        datapath.send_msg(req)
```

Here we use OFPGroupMod API for creation groups. refer the API for more details https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html#ryu.ofproto.ofproto_v1_3_parser.OFPGroupMod

2. In Switch features handler function, call the Group table creation function and add flows for switch S1 and S5,

```
        # switch s1
        if datapath.id == 1:
            # add group tables
            self.send_group_mod(datapath)
            actions = [parser.OFPActionGroup(group_id=50)]
            match = parser.OFPMatch(in_port=3)
            self.add_flow(datapath, 10, match, actions)

            #add the return flow for h1 in s1.
            # h1 is connected to port 3.
            actions = [parser.OFPActionOutput(3)]
            match = parser.OFPMatch(in_port=1)
            self.add_flow(datapath, 10, match, actions)

            actions = [parser.OFPActionOutput(3)]
            match = parser.OFPMatch(in_port=2)
            self.add_flow(datapath, 10, match, actions)

       # switch s5
        if datapath.id == 5:
        # add group tables
            self.send_group_mod(datapath)
            actions = [parser.OFPActionGroup(group_id=50)]
            match = parser.OFPMatch(in_port=3)
            self.add_flow(datapath, 10, match, actions)


            #add the return flow for h2 in s4.
            # h2 is connected to port 3.
            actions = [parser.OFPActionOutput(3)]
            match = parser.OFPMatch(in_port=1)
            self.add_flow(datapath, 10, match, actions)

            actions = [parser.OFPActionOutput(3)]
            match = parser.OFPMatch(in_port=2)
            self.add_flow(datapath, 10, match, actions)
```

Demo

1. run the topology in mininiet

```
sudo python lb_topo.py
```

2. 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
```

3. Run the ryu manager

```
ryu-manager loadbalancer.py
```

4. 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.

Further experiments:

- Multipath load balancing with specified percentage
- Dynamically do the load balancing depends on the traffic condition.
- Failover

References

- openflow 1.3 specification documemnt, 5.6 Group table
- http://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html
- https://www.quora.com/Why-dose-the-Openflow-protocol-exist-the-group-table-And-what-the-relationship-between-pipeline-and-group-table
- http://www.muzixing.com/pages/2014/11/07/load-balancemultipath-application-on-ryu.html

---

13. Packet Generation

RYU comes with packent library (consists of plenty of protocols), we can use this library to parse the packets or generate the packets.

There are many usecase around this.

Packet Parsing Capbility

We can use the packet library to parse the PACKET IN Message and identify which application and use it.

1. Include the relevant packet library

```
from ryu.lib.packet import ipv4
```

2. In the Packet In handler,

Get the Packet Handler object

```
pkt = packet.Packet(msg.data)
```

Now we can get the Protocol headers from the pkt

To get the ethernet header

```
eth = pkt.get_protocols(ethernet.ethernet)
self.logger.info("Ethernet Header %s ",eth)
```

To Get the IP Header

```
ip = pkt.get_protocol(ipv4.ipv4)
self.logger.info("IPv4 Header %s ",eth)
```

To Get the TCP header

```
t = pkt.get_protocol(tcp.tcp)
self.logger.info("TCP Header %s ",eth)
```

To get the UDP header

```
u = pkt.get_protocol(udp.udp)
```

```
self.logger.info("UDP Header %s ",eth)
```

Further we can get the Protocol header details, by processing it

```
u = pkt.get_protocol(udp.udp)
self.logger.info("UDP Header %s ",eth)
self.logger.info ("UDP Src port %d Destination Port %d ",u.src_port, u.dst_port)
```

RYU can handle these many protocols

https://github.com/osrg/ryu/tree/master/ryu/lib/packet

Packet Generation Capbility

In SDN, SDN Controller can generate a packet and send it. There are many usecases/applications require this capability.

Router SDN Controller needs to have ARP Request & Response capability for neighbour discovery

Proxy Application (ARP Proxy, DNS Proxy) Proxy application need to handle the request of the specific proxy application protocol(such as DNS, ARP etc) and respond it.

Simple ARP Proxy application

Objective This application answers the ARP requests(instead of forwarding the ARP broadcasts to all hosts)

This Application act as ARP Proxy. It captures the ARP Packets, builds the ARP response and send it to the Src.

Logic

1. MAC with IP Table will be hardcoded in the code.
2. In the Packet IN handler, check the packet is ARP Packet.
3. If it is ARP Packet,
   - In the ARP Request Packet, decode the destination IP from ARP header
   - Get the MAC address of the equivalent IP address
   - Build the ARP Response packet
   - Send it to the Received PORT.
4. If not ARP, normal switching routine.

Coding

1. include the lib

```
from ryu.lib.packet import arp
```

2. define your table with MAC address and IP Address.(ARP Table)

```
arp_table = {"10.1.1.1": "00:00:00:00:00:01",
             "10.1.1.2": "00:00:00:00:00:02",
             "10.1.1.3": "00:00:00:00:00:03",
             "10.1.1.4": "00:00:00:00:00:04"
             }
```

3. Write your arp proxy function

```
    def arp_process(self, datapath, eth, a, in_port):
        r = arp_table.get(a.dst_ip)
        if r:
            self.logger.info("Matched MAC %s ", r)
            arp_resp = packet.Packet()
            arp_resp.add_protocol(ethernet.ethernet(ethertype=eth.ethertype,
                                  dst=eth.src, src=r))
            arp_resp.add_protocol(arp.arp(opcode=arp.ARP_REPLY,
```

```
                            src_mac=r, src_ip=a.dst_ip,
                            dst_mac=a.src_mac,
                            dst_ip=a.src_ip))

        arp_resp.serialize()
        actions = []
        actions.append(datapath.ofproto_parser.OFPActionOutput(in_port))
        parser = datapath.ofproto_parser
        ofproto = datapath.ofproto
        out = parser.OFPPacketOut(datapath=datapath, buffer_id=ofproto.OFP_NO_BUFFER,
                            in_port=ofproto.OFPP_CONTROLLER, actions=actions, data=arp_resp)
        datapath.send_msg(out)
        self.logger.info("Proxied ARP Response packet")
```

4. In the packet handler function, if it is arp packet, call the arp_proxy function.

```
    # Check whether is it arp packet
    if eth.ethertype == ether_types.ETH_TYPE_ARP:
        self.logger.info("Received ARP Packet %s %s %s ", dpid, src, dst)
        a = pkt.get_protocol(arp.arp)
        self.arp_process(datapath, eth, a, in_port)
        return
```

Demo

1. Run Mininet topology

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

2. Run RYU hub application

```
ryu-manager arp_proxy.py
```

3. Check the openvswitch flows

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

4. start the tcpdump capture in h2

```
xterm h2
tcpdump -i any -vvv
```

5. do ping h1 to h2 from mininet

```
mininet>h1 ping h2
```

6. Check the tcpdump traces in h2 you will not the ARP Request, Controller respond with ARP Reply.

Further Experiments

- Implement DNS Proxy Application
- Implement Simple Gateway / router Application

---

14. Topology Discovery

RYU has Topology discovery mechnism. RYU uses LLDP(Link Layer Discovery Protocol) to discover the switches and links. Topology discovery feature will be enabled with **"--observe-links"** option.

Topology discovery feature is used in many applications such as identify the shortest path, avoid loops, multipath with load balancing etc.

How it works

Switch & Link Discovery:

- RYU Controller generates LLDP Packet(datapath id, port number) from each switch and each port number.
- LLDP Packet reaches the other end of the link (its another switch port).
- This packet will send it to the controller
- Controller decodes the packet and learn the Switch, and Link information from the LLDP Packet (sender switch details(datapath ID, Port number) and received switch details((datapath ID, Port number)))

Host discovery:

Host discovery is based on the incoming packet. Usually the host machines are keep sending some broadcast/multicast packets(some services will trigger this). RYU Controller uses these packets to discover the hosts.

Coding

Logic:

Topology discovery process may take few seconds to complete it . So we use RYU thread and print the topology information after 10seconds. We assume within 10 seconds, the topology discovery process will be completed.

1. Include the header

```
from ryu.topology.api import get_switch, get_link, get_host
```

2. Initiate a thread

```
        hub.spawn(self.myfunction)
```

3. Printing the topology discovery information

```
    def myfunction(self):
        self.logger.info("started new thread")
        hub.sleep(10)

        switch_list = get_switch(self.topology_api_app, None)
        self.switches = [switch.dp.id for switch in switch_list]
        links_list = get_link(self.topology_api_app, None)
        self.links = [(link.src.dpid, link.dst.dpid, {'port': link.src.port_no}) for link in links_list]
        host_list = get_host(self.topology_api_app, None)
        self.hosts = [(host.mac, host.port.dpid, {'port': host.port.port_no}) for host in host_list]
        self.logger.info("*********Topology Information*************")
        self.logger.info("Switches %s", self.switches)
        self.logger.info("Links %s", self.links)
        self.logger.info("Hosts %s", self.hosts)
```

we use three APIs **get_switch, get_link, get_host** . These api provides the discovered switch,link and host details in the list of objects.

This below line, we are processing the switch list, and get only switch dp id.

```
        self.switches = [switch.dp.id for switch in switch_list]
```

Simiar to other link and host.

Demo

1. Run Mininet Linear Topology

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

2. Start the Wireshark Capture with loopback interface

3. Run ryu controller with "--observe-links" options

```
ryu-manager --observe-links topology_discovery.py

*********Topology Information*************
Switches [1, 2, 3, 4]
Links [(2, 3, {'port': 3}), (3, 2, {'port': 2}), (3, 4, {'port': 3}), (2, 1, {'port': 2}), (4, 3, {'port': 2}), (1, 2,
Hosts [('00:00:00:00:00:03', 3, {'port': 1}), ('00:00:00:00:00:02', 2, {'port': 1}), ('00:00:00:00:00:01', 1, {'port':
```

5. Analyze the Wireshark traces(LLDP Packets)

References

- https://en.wikipedia.org/wiki/Link_Layer_Discovery_Protocol
- https://vlkan.com/blog/post/2013/08/06/sdn-discovery/
- https://github.com/Ehsan70/RyuApps/blob/master/TopoDiscoveryInRyu.md

---

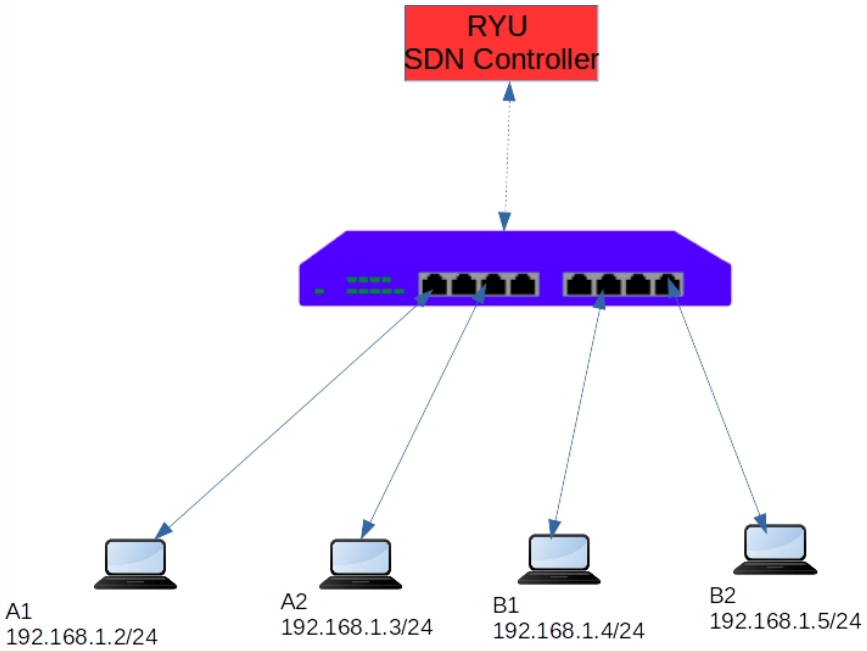
15. Multicontrollers

Problem Statement

1. Controller failure in SDN Network is major failure affects the entire network topology

2. Various reasons for controller failure - security threats, hardware /software issues, manual errors etc.

3. Using Single Controller in SDN Network is high risk, high possibility for failure, not scalable and fault tolerant.



Demo - Single Controller

1. Running Mininet topology with Linear topology.

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

2. Running a ryu controller with simple switch application(idle timeout/hard timeout to 30seconds)

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

3. Pingall.

4. Stop the controller

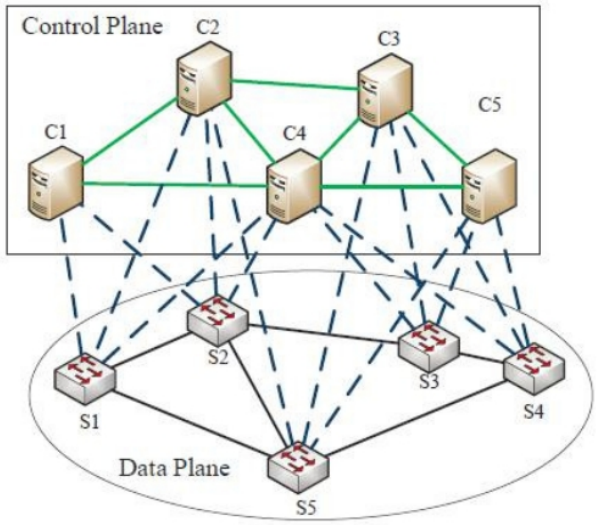
5. Check the flows

Observation: Once the controller is stopped, topology works till the flows are timeout(hard) after that, the entire topology datapath is not working. hosts cannot reach other hosts.

Its a single point failure.

Cluster of SDN Controllers:

Openflow specification supports the Multiple controllers environment. User can configure the multiple controllers in the switch.



Controller Roles

Equal Role

It means, all the controllers configured in the switch have full control to update/modify the flows.

Switch must send the PACKET IN Message to all the Controllers. Also switch process the PACKET OUT, FLOW UPDATES etc from all the controllers.

Master Role

It means, MASTER controller will be responsible for managing the switch openflow dataplane. Switch will send the control messages to only MASTER controller.

Slave Role

Slave plays backup role for MASTER. it also receives the HELLO and KEEPALIVE Message. But it cannot send and receive the Control message.

4.Openflow ROLE Change Message:

ROLE_REQUEST and RESPONSE:

When the controller comes up, it will send the ROLE REQUEST with ROLE MASTER, other controller should send a role as SLAVE. Switch will communicate with MASTER.

Demo - Equal role

1. controller1

```
ryu-manager --ofp-tcp-listen-port 6653 flow_timeout.py
```

2. controller2

```
ryu-manager --ofp-tcp-listen-port 6654 flow_timeout.py
```

```
sudo python topology.py
```

4. Start the wireshark capture for traces

5. pingall

Note : we can see the packet in comes from both controller

Demo- Master/Slave Role

Here one controller going to act as Master Role, another controller going to act as SLAVE or BACKUP role.

1. Start the wireshark capture for traces in loopback interface

2. Mininet topology

```
sudo python topology.py
```

3. start the master controller

```
ryu-manager --ofp-tcp-listen-port 6653 l3switch_master.py
```

4. start the backup controller

```
ryu-manager --ofp-tcp-listen-port 6654 l3switch_slave.py
```

5. Check the traces in controller terminals. you can see only MASTER terminal you can see packet in traces.

6. pingall

only MASTER Controller manages the switches

7. Analyze the ROLE Messages in the Wireshark traces.

Data Synchronization

In Multicontroller environement(specifically Master/slave), data(intelligence built by the controller. For example mac_to_port structure) needs to be synchronized across the controller.

There are many mechnisms available such as DB, InMemoryDB, Message Brokers etc.

Further experiments

- Use InmemoryDB or MessageBroker for DataSynchronization
- Implement Master Election Algorithm/ Failure Detection /Autofailover
- Distributed Master/Slaves

---

16. REST API

In this guide, we will see how we can include REST API support in our application. So far, we use only application class(app_manager.RyuApp) For REST API support, We have to define two class in the application.

application (app_manager.RyuApp)

- Our default code
- REST API endpoints declaration

Controller (ControllerBase) - Here we need to implement the REST API handling function.

Sample Program

In this exercise, included dummy REST API which returns hardcoded value.

1. Include the WSGI library.

```
from ryu.app.wsgi import WSGIApplication
from ryu.app.wsgi import ControllerBase
from ryu.app.wsgi import Response
```

2. In the Application class, include wsgi(WebServer) context

```
    _CONTEXTS = {
                'wsgi': WSGIApplication,
                }
```

3.

defines the controller name, mapper class

```
        # REST handler
        wsgi = kwargs['wsgi']
        mapper = wsgi.mapper
        wsgi.registory['DeviceController'] = self.data
```

4. declare the REST API

```
        # API linkages
        uri = "/apps"
        mapper.connect('stats', uri,
                       controller=DeviceController, action='get_apps',
                       conditions=dict(method=['GET']))
```

Here we declare the REST API endpoint and methods.

- specify which controller it connects .
- which function will be triggererd

5. Define the controller class

```
class DeviceController(ControllerBase):

    def __init__(self, req, link, data, **config):
        super(DeviceController, self).__init__(req, link, data, **config)
        self.data = data
    #self.data["apps"]
    def get_apps(self, req, **kwargs):
        apps = self.data["apps"]
        body = json.dumps(apps)
        return Response(content_type='application/json', body=body)
```

Here we defined the api functions.

Demo As its a dummy application, we dont require mininet

1. run the ryu application

```
ryu-manager rest_simple.py
```

2. GET Request to the API

```
curl localhost:8080/apps
```

References

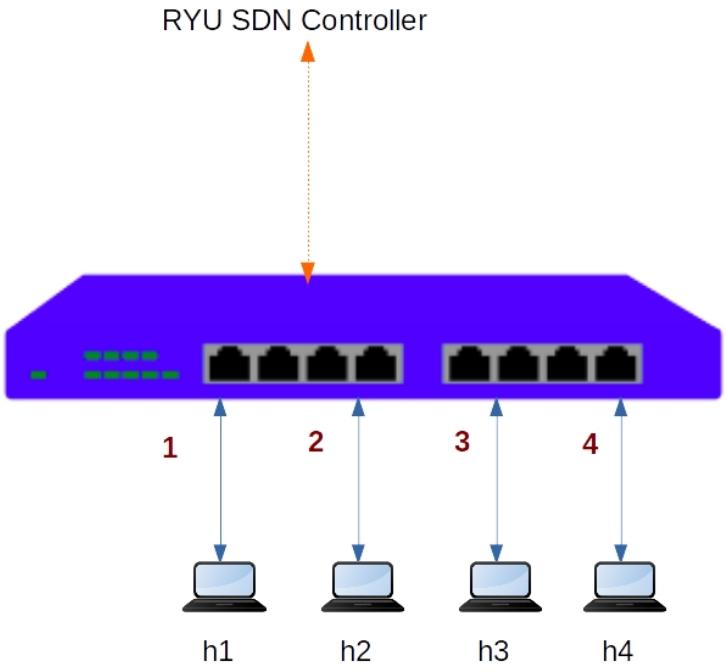
https://osrg.github.io/ryu-book/en/html/rest_api.html

17. Traffic Tests - Part1

Usually we need to perform various traffic tests between the hosts to test our network topology & SDN application performance.

Some of the test includes:

1. TCP Tests between two hosts
2. UDP Tests
3. UDP Tests with different Packet size (64 bytes, 1024 bytes etc), different packet rate (50 pkts/s, 100 pkts/s etc)
4. Variable Traffic rate (10Pkts/s for first 60s, then 100Pkts/s for rest of the test)
5. Multiple/Parallel Streams/Sessions
6. VoIP Traffic Test
7. Video Streaming mp4 video file will be used as video source. and will generate the video stream using VLC, and will be received by client using VLC Video Player.



TCP Tests

IPERF is widely accepted traffic generation tool to perform TCP Tests.

- TCP tests generally used for measure the bandwidth.
- Latency, Jitter, Packet loss cannot be measured using TCP Tests.

For all our traffic tests, we will use Simple Topology and l4 switch application.

1. Create the topology

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

2. Run the RYU Controller l4 switch applicaion.

```
ryu-manager l4_switch.py
```

3. In mininet shell, do ping h1 to h2.

```
mininet>h1 ping -c 5 h2
```

4.check the flows

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

5. understand the Host & Switch Port numbers.

```
mininet>links
mininet>ports
```

6. To check the switch statistics

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

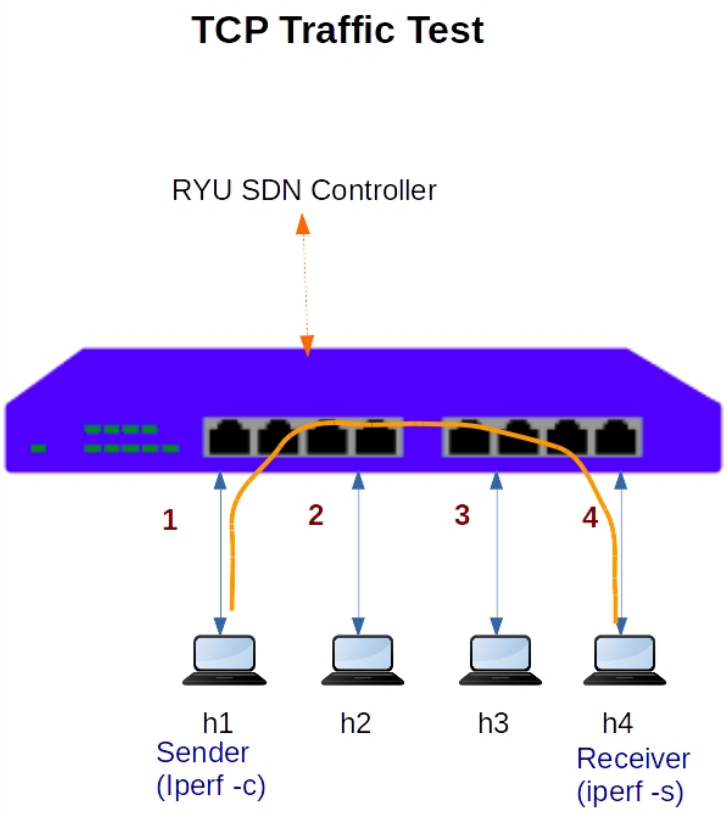
Now all set to run our traffic tests.

A. Traffic test from h1 to h4

Objective: Generate TCP Traffic from h1 to h4. (Measure bandwidth from h1 to h4)

h1 is sender.

h4 is receiver.



1. Start IPERF Server in h4

```
h4 iperf -s &
```

2. Start the IPERF Client in h1 and connecting to h4

```
h1 iperf -c h4
```

3. Analyze the results by flows

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

we observe traffic in both directions.

h1 to h4 traffc is very high (in Gbps). This is data traffic. h4 to h1 traffic is very less. This is TCP Acknowledge traffic.

4. Analyze the results by ports.

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

h1 --------port1, port4--------h4

Forward traffic:

- h1 transimits. port1 receives.
- port4 transmits, h4 receives.

Acknowlege:

- h4 transmits. port4 receives.
- port1 transmits. h1 receives.

B. Bidirectional Traffic test h1 to h4(sequentially).

1. Start IPERF Server in h4

```
h4 iperf -s &
```

2. Start the IPERF Client in h1 and connecting to h4

```
h1 iperf -c h4 -r
```

Its sequential test , once h1 to h4 traffic test is completed it will start h4 to h1 traffic test.

C. Bidirectional Traffic test h1 to h4(parallel).

1. Start IPERF Server in h4

```
h4 iperf -s &
```

2. Start the IPERF Client in h1 and connecting to h4

```
h1 iperf -c h4 -d
```

Its parallel test , both direction h1 to h4 as well h4 to h1 traffic tests started.

D. Traffic test from h1 to h4 with Multiple Sessions.

1. Start IPERF Server in h4

```
h4 iperf -s &
```

2. Start the IPERF Client in h1 and connecting to h4

```
h1 iperf -c h4 -P 5
```

UDP Tests with IPERF

UDP Tests are quite flexible in the nature of playing around packet sizes(64 bytes) and number of packets(10 Packets/s) you want to send. It means, user can control the bandwidth of UDP Traffic.

- Bandwidth, Latency, Jitter, Packet loss can be measured using UDP Tests.

IPERF doesnot provide provide much flexibility in UDP Tests.

1. Start IPERF UDP Server in h4

```
h4 iperf -u -s &
```

2. Start the IPERF Client in h1 and connecting to h4 and generate bandwidth of 10Mbsp

```
h1 iperf -u -c h4 -b 10m
```

UDP Tests with MGEN

Multi-Generator (MGEN) is open source traffic gen software

https://downloads.pf.itd.nrl.navy.mil/docs/mgen/mgen.html

Some important features

- variable packet rate
- pattern(Periodic, possion, burst, jitter)
- logs
- scriptable

It doesnot give test report (similar to iperf). we need to process the logs and prepare it.

Installation

```
sudo apt-get install mgen
```

A. Simple MGEN UDP Test (Variable Packet rate)

Objective:

we want to generate 10 packets per second for first 0th to 100 seconds. then will send 100 packets per secondfor 100th to 200 seconds. packet size is 1024 bytes.

receive.mgn

```
0.0 LISTEN UDP 5000-5001
```

send.mgn

```
0.0 ON 1 UDP SRC 5001 DST 10.1.1.3/5001 PERIODIC [10 1024]
100.0 MOD 1 PERIODIC [100 1024]
200.0 OFF 1
```

How to run mgen:

The default/minimum syntax is below,

> mgen input script name

Testing

1. Start MGEN Receiver in h3

```
h3 mgen input receive.mgn output mgenlog.txt &
```

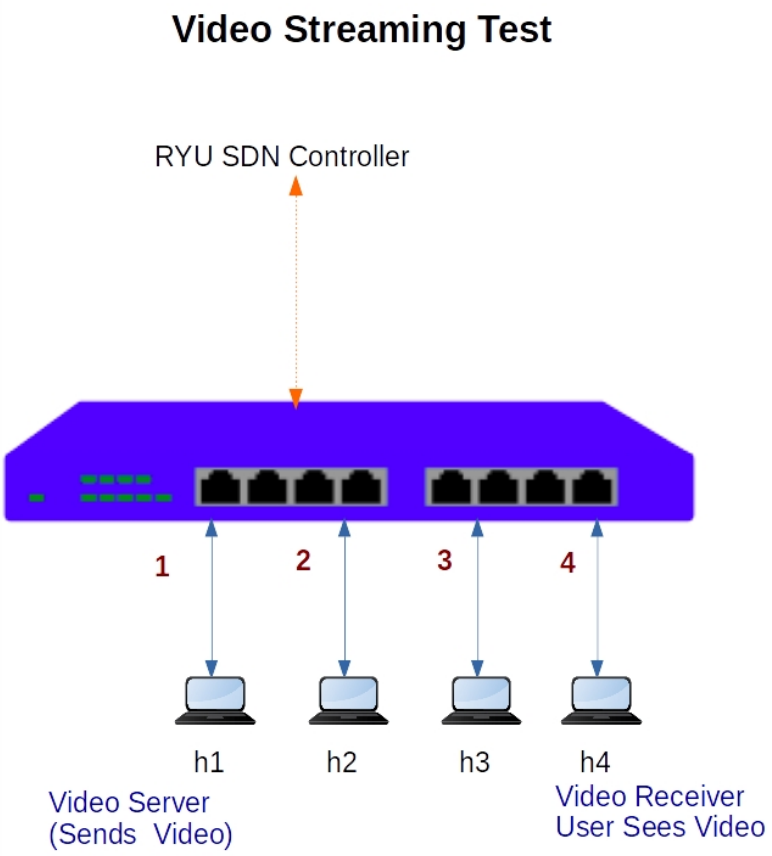
2. Start the MGEN sender in h1

```
h1 mgen input send.mgn &
```

we use VLC media player(https://www.videolan.org/index.html) for Video Streaming testing.

installation procedure

```
sudo apt-get install vlc vlc-nox
```



Testing

1. Create the topology

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

2. Run the RYU Controller l4 switch applicaion.

```
ryu-manager l4_switch.py
```

3. Download the sample video from internet (or use your own video file)

4. h4 is a Video receiver. Start the video receiver in h4 xterm shell

```
mininet> xterm h4
su - user-name
vlc rtp://@10.0.0.4:9001
```

5. h1 is video sender. Start the video streaming sender in h1 xterm shell

```
mininet> xterm h1
su - user-name
```

```
cvlc  /home/sureh/cat3.mp4  --sout '#rtp{proto=udp,mux=ts,dst=10.0.0.4,port=9001}'
```

VOIP Tests

VOIP Traffic is UDP Stream. This can be simulated thru IPERF UDP Tests.

Objective 1. Test the single Voip call 64Kbps 2. Test parallel voip calls

Testing

1. Create the topology

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

2. Run the RYU Controller l4 switch applicaion.

```
ryu-manager l4_switch.py
```

A. Single 64Kbps VOICE CALL Test

start the 64Kbps VOIP Traffic Test between h1 to h4 for 60 seconds ,and vice versa.

1) Run the IPERF UDP server in h4

In mininet CLI

```
mininet>h4 iperf --server --udp --len 300 --tos 184 -fk --interval 5 &
```

Here we are starting udp server, and setting the TOS field to 184

2) Run the IPERF UDP Client in h1

In mininet CLI

```
h1 iperf -c 10.1.1.4 --udp --len 300 --bandwidth 67000 --dualtest --tradeoff --tos 184 -fk --interval 5 --time 60 --li
```

B. Multiple Parallel calls VOIP calls test

1) Run the IPERF UDP server in h4

```
mininet>h4 iperf --server --udp --len 300 --tos 184 -fk --interval 5 --parallel 4
```

2) Run the IPERF UDP Client in h1

```
mininet>iperf -c 10.1.1.4 --udp --len 300 --bandwidth 67000 --dualtest --tradeoff --tos 184 -fk --interval 5 --time 60
```

References

1. http://wiki.innovaphone.com/index.php?title=Howto:Network_VoIP_Readiness_Test
2. https://www.videolan.org/index.html

---

18. Traffic Tests with DITG

D-ITG (Distributed Internet Traffic Generator) is a platform capable to produce traffic at packet level accurately replicating appropriate stochastic processes for both IDT (Inter Departure Time) and PS (Packet Size) random variables (exponential, uniform, cauchy, normal, pareto, ...).

D-ITG supports both IPv4 and IPv6 traffic generation and it is capable to generate traffic at network, transport, and application layer.

Once installed, Quick verification

```
suresh@suresh-Latitude-6430U:~$ ITGSend
ITGSend version 2.8.1 (r1023)
Compile-time options: sctp dccp bursty multiport

Missing argument!!!
Try ITGSend -h or --help for more information
```

Quick Start

ITGRecv command for receiver

ITGSend command for sender

ITGSend/Recv generate the logs. which needs to be ananlzed by **ITGDec** command to see the result.

UDP

1. start the ryu l4_switch app

```
ryu-manager l4_switch.py
```

2. run the mininet topology

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

3. start the receiver (or server) on h2

```
xterm h2
ITGRecv -l /tmp/receiver.log
```

Here the log file is stored in /tmp/receiver.log, which can be decoded via ITGDec.

4. start the sender

```
h1 ITGSend -T UDP -a h2 -c 100 -C 10 -t 15000
```

The options are explained below,

```
-t 15000   is test duration in milliseconds  (15 seconds)

-a h2    target ip

-c 100   packet size

-C 10    packet per second (pps)

-T UDP   Protocol - UDP/TCP/
```

5. verify the flows:

You can observe 1 UDP flows and 2 TCP flows(server port 9000). Those TCP flows are used for communicating between ITGSend and ITGRecv for control information.

```
$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=4.876s, table=0, n_packets=5, n_bytes=358, priority=1,tcp,nw_src=10.1.1.1,nw_dst=10.1.1.2,tp_src
 cookie=0x0, duration=4.874s, table=0, n_packets=3, n_bytes=206, priority=1,tcp,nw_src=10.1.1.2,nw_dst=10.1.1.1,tp_src
 cookie=0x0, duration=4.871s, table=0, n_packets=45, n_bytes=6390, priority=1,udp,nw_src=10.1.1.1,nw_dst=10.1.1.2,tp_s
 cookie=0x0, duration=47.834s, table=0, n_packets=19, n_bytes=1554, priority=0 actions=CONTROLLER:65535
(ry) suresh@suresh-Latitude-6430U:~/projects/ditg$
```

5. To Check the result

```
cd /tmp

ITGDec receiver.log

(ry) suresh@suresh-Latitude-6430U:~/projects/ditg$ ITGDec /tmp/receiver.log
ITGDec version 2.8.1 (r1023)
Compile-time options: sctp dccp bursty multiport
\------------------------------------------------------------
Flow number: 1
From 10.1.1.1:39783
To    10.1.1.2:8999
------------------------------------------------------------
Total time               =     14.923949 s
Total packets            =           150
Minimum delay            =      0.000057 s
Maximum delay            =      0.001506 s
Average delay            =      0.000162 s
Average jitter           =      0.000040 s
Delay standard deviation =      0.000118 s
Bytes received           =         15000
Average bitrate          =      8.040767 Kbit/s
Average packet rate      =     10.050959 pkt/s
Packets dropped          =             0 (0.00 %)
Average loss-burst size  =      0.000000 pkt
------------------------------------------------------------

_____________________________________________________________
***************  TOTAL RESULTS   ******************
_____________________________________________________________
Number of flows          =             1
Total time               =     14.923949 s
Total packets            =           150
Minimum delay            =      0.000057 s
Maximum delay            =      0.001506 s
Average delay            =      0.000162 s
Average jitter           =      0.000040 s
Delay standard deviation =      0.000118 s
Bytes received           =         15000
Average bitrate          =      8.040767 Kbit/s
Average packet rate      =     10.050959 pkt/s
Packets dropped          =             0 (0.00 %)
Average loss-burst size  =             0 pkt
Error lines              =             0
------------------------------------------------------------
```

The result contain two parts. First part is flow specific,

we ran only one UDP flow, this is mapped with FLOW Number. if we have multiple flows, we can see each flow result.

Second Part is , as a Whole receiver summary.

6. Stop the ITGRecv on h2

UDP with three flows(Multiflow mode).

Syntax is below

```
ITGSend <script_file> [log_opts]
```

1. Create a script file /tmp/send.sh

```
-T UDP -a 10.1.1.2 -c 100 -C 30 -t 15000
-T UDP -a 10.1.1.2 -c 500 -C 20 -t 20000
-T UDP -a 10.1.1.2 -c 1024 -C 10 -t 25000
```

Here we generate 3 flows

1st flow generates 30 pkts/per second, with packet size(payload) 100 bytes for 15 seconds

2nd flow generates 20 pkts/per second, with packet size(payload) 500 bytes for 20 seconds

3rd flow generates 10 pkts/per second, with packet size(payload) 1024 bytes for 25 seconds

2. start the receiver (or server) on h2

```
xterm h2
ITGRecv -l /tmp/receiver1.log
```

Here the log file is stored in /tmp/receiver.log, which can be decoded via ITGDec.

3. start the sender

```
h1 ITGSend  /tmp/send.sh

Logs:
mininet> h1 ITGSend /tmp/send.sh
ITGSend version 2.8.1 (r1023)
Compile-time options: sctp dccp bursty multiport
Started sending packets of flow ID: 1
Started sending packets of flow ID: 2
Started sending packets of flow ID: 3
Finished sending packets of flow ID: 1

Finished sending packets of flow ID: 2

Finished sending packets of flow ID: 3
```

4. verify the flows:

you can see 3 UDP flows and 2 TCP Flows (control channel)

5. stop the receiver in h2

6. result verification

we see 3 flows, each flow represents the one which we sent.

```
$ ITGDec /tmp/receiver1.log
ITGDec version 2.8.1 (r1023)
Compile-time options: sctp dccp bursty multiport
/------------------------------------------------------------
Flow number: 1
From 10.1.1.1:58990
To    10.1.1.2:8999
------------------------------------------------------------
Total time               =     14.984081 s
Total packets            =           449
Minimum delay            =      0.000056 s
Maximum delay            =      0.004455 s
Average delay            =      0.000173 s
Average jitter           =      0.000041 s
Delay standard deviation =      0.000207 s
Bytes received           =         44900
Average bitrate          =     23.972107 Kbit/s
Average packet rate      =     29.965134 pkt/s
Packets dropped          =             0 (0.00 %)
Average loss-burst size  =      0.000000 pkt
------------------------------------------------------------
------------------------------------------------------------
Flow number: 2
From 10.1.1.1:39348
To    10.1.1.2:8999
------------------------------------------------------------
Total time               =     19.994380 s
Total packets            =           400
Minimum delay            =      0.000056 s
Maximum delay            =      0.004048 s
Average delay            =      0.000174 s
Average jitter           =      0.000053 s
Delay standard deviation =      0.000199 s
```

```
Bytes received           =        200000
Average bitrate          =     80.022486 Kbit/s
Average packet rate      =     20.005622 pkt/s
Packets dropped          =             0 (0.00 %)
Average loss-burst size  =      0.000000 pkt
------------------------------------------------------------
------------------------------------------------------------
Flow number: 3
From 10.1.1.1:60097
To    10.1.1.2:8999
------------------------------------------------------------
Total time               =     24.938133 s
Total packets            =           235
Minimum delay            =      0.000058 s
Maximum delay            =      0.003840 s
Average delay            =      0.000189 s
Average jitter           =      0.000054 s
Delay standard deviation =      0.000247 s
Bytes received           =        240640
Average bitrate          =     77.195835 Kbit/s
Average packet rate      =      9.423320 pkt/s
Packets dropped          =            15 (6.00 %)
Average loss-burst size  =      3.000000 pkt
------------------------------------------------------------


****************  TOTAL RESULTS   ******************

Number of flows          =             3
Total time               =     24.953222 s
Total packets            =          1084
Minimum delay            =      0.000056 s
Maximum delay            =      0.004455 s
Average delay            =      0.000177 s
Average jitter           =      0.000056 s
Delay standard deviation =      0.000214 s
Bytes received           =        485540
Average bitrate          =    155.664066 Kbit/s
Average packet rate      =     43.441284 pkt/s
Packets dropped          =            15 (1.36 %)
Average loss-burst size  =      0.000000 pkt
Error lines              =             0
------------------------------------------------------------
```

TCP Test

DITG TCP Test is more powerful, we can configure the Packet Size, Number of packets/s etc.

1. start the receiver (or server) on h2

```
xterm h2
ITGRecv -l /tmp/receiver1.log
```

2. start the sender

```
h1 ITGSend -T TCP  -a h2  -t 15000
h1 ITGSend -T TCP -a h2 -c 1000 -C 100 -t 15000
```

3. check the logs as above test.

Packet distribution (Rate & Size)

Inter-departure time options (Time factor)

Constant distribution

-C rate Constant

```
h1 ITGSend -T TCP -a h2 -c 1000 -C 100 -t 15000
```

-U min_rate max_rate Uniform distribution.

```
h1 ITGSend -T TCP -a h2 -c 1000 -U 10 100 -t 15000
```

Poisson distribution

-O mean Poisson distribution.

```
h1 ITGSend -T TCP -a h2 -c 1000 -O 50 -t 15000
```

Packet size variation

-c pkt_size Constant (default: 512 bytes).

-u min_pkt_size max_pkt_size Uniform distribution.

-o mean Poisson distribution.

```
h1 ITGSend -T TCP -a h2 -c 1000 -O 50 -u 10 1000 -t 15000
h1 ITGSend -T TCP -a h2 -c 1000 -O 50 -o 100 -t 15000
```

VOIP,Telnet,DNS

For VOIP, Telnet, DNS Applications, Receiver method will be same as above examples. only sender option will change as below

1. VOIP

```
h1 ITGSend  -t 15000 -a h2 -rp 10001 VoIP -x G.711.2 -h RTP -VAD
```

2. Telnet

```
h1 ITGSend -t 15000 -a h2 -rp 10002 Telnet
```

3. DNS

```
h1 ITGSend -t 15000 -a h2 -rp 10003 DNS
```

3. FTP

Read payload from file

udp

```
h1 ITGSend  -a h2 -Fp test.dat
```

tcp

```
h1 ITGSend  -T TCP -a h2 -Fp test.dat
```

Video streaming

1. receiver

```
xterm h2
ITGRecv -l /tmp/receiver1.log
```

2. Sender

Script file (/tmp/send.sh)

```
-C 24 -a 10.1.1.2 -n 27791 6254 -t 720000  -T UDP -m rttm -sp 10101 -rp 10001
-C 24 -a 10.1.1.2 -n 27791 6254 -t 720000  -T UDP -m rttm -sp 10102 -rp 10002
```

Option details are below -C 24 pkts/second -n 27791 6254 (normal distribution options - packet size) -t 720000 (test time 720s) -m rttm (meter round-trip time meter) -sp 10101 srcport -rp 10001 receiver port -H NAT Traversal

```
h1 ITGSend  /tmp/send.sh
```

---

19. References

1. https://www.opennetworking.org/sdn-definition/
2. https://en.wikipedia.org/wiki/Software-defined_networking
3. https://www.sdxcentral.com/sdn-nfv-use-cases/
4. https://github.com/mininet/mininet/wiki/Introduction-to-Mininet
5. https://github.com/mininet/mininet/tree/master/examples
6. http://mininet.org/api/annotated.html
7. https://kiranvemuri.info/dynamic-topology-changes-in-mininet-advanced-users-5c452f7f302a
8. https://stackoverflow.com/questions/43070043/mininet-wifi-custom-topologyhttps://stackoverflow.com/questions/43070043/mininet-wifi-custom-topology
9. http://mininet.org/walkthrough/
10. https://github.com/PrincetonUniversity/Coursera-SDN/tree/master/assignments/mininet-topology
11. https://www.pythonforbeginners.com/modules-in-python/how-to-use-simplehttpserver/
12. https://ryu.readthedocs.io/en/latest/writing_ryu_app.html

---

OLD Deleted Contents, keeping it for refernece

Install in ubuntu 18.04

Setup the SDN Test environment to practice Openflow usecases with RYU SDN Controller.

Tools to be installed:

OS	Ubuntu 18.04 Desktop
Test Bed	Mininet
Controller	RYU
Switch	Openvswitch
Packet Capture	Wireshark
Traffic Generator	IPerf

Installation

Requirements:

OS: Ubuntu 18.04.x

CPU: 2 Cores +

RAM: 4GB +

HDD: 15GB+

If you are using Windows or other OS, you can install Ubuntu 18.04 as a Virtual Machine(VM) using Virtual Box .

You can download the ISO installer from the below link,

https://www.ubuntu.com/download/desktop

As first step, please run this command.

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install git gcc python-dev libffi-dev libssl-dev libxml2-dev libxslt1-dev zlib1g-dev python-pip
```

Once we installed the above packages, python 2.7 and relavent dependents are installed properly.

To check the python version:

> python --version

Example:

```
suresh@suresh-vm:~$ python --version
Python 2.7.15rc1
suresh@suresh-vm:~$
```

At the time of writing, Ubuntu 18.04 official repository has the following tools versions

```
Tool Name       Version
*************************
Openvswitch :     2.9.2
Wireshark   :     2.6.6
IPERF       :     2.0.10
Mininet   :     2.2.2
```

RYU will be installed using PIP.

```
Tool Name      Version
*************************
RYU       :     4.31
```

a. Openvswitch Installation

> sudo apt-get install openvswitch-switch

To verify :

> ovs-vsctl --version

Example:

```
suresh@suresh-vm:~$ ovs-vsctl --version
ovs-vsctl (Open vSwitch) 2.9.2
DB Schema 7.15.1
suresh@suresh-vm:~$
```

b. Wireshark Installation

> sudo apt-get install wireshark

To verify :

> wireshark --version

```
suresh@suresh-vm:~$ wireshark --version
Wireshark 2.6.6 (Git v2.6.6 packaged as 2.6.6-1~ubuntu18.04.0)
```

c. Traffic Test Tools installation

IPERF:

> sudo apt-get install iperf

CURL:

> sudo apt-get install curl

HPING3:

> sudo apt-get install hping3

Apache Bench:

> sudo apt-get install apache2-utils

To verify :

> iperf --version

```
suresh@suresh-vm:~$ iperf -v
iperf version 2.0.10 (2 June 2018) pthreads
suresh@suresh-vm:~$
```

d. RYU installation

> sudo pip install ryu

To verify :

> ryu-manager --version

Example:

```
suresh@suresh-vm:~$ ryu-manager --version
ryu-manager 4.31
suresh@suresh-vm:~$
```

Note: Make sure you do with "sudo pip install ryu" . ryu package will install ryu-manager binary in /usr/local/bin folder. Hence installation requires sudo access.

e. Mininet Installation

> sudo apt-get install mininet

To verify :

> mn --version

Log:

```
suresh@suresh-vm:~$mn --version
2.2.2
suresh@suresh-vm:~$
```

Previous
« OpenDayLight (ODL) SDN Crash Course - LAB Guide

Next
Powered by MDX »

Community

Github

Facebook

Youtube

Contact Us

Email: knetsolutions2@gmail.com

Mobile/Whatsapp: +919445042007