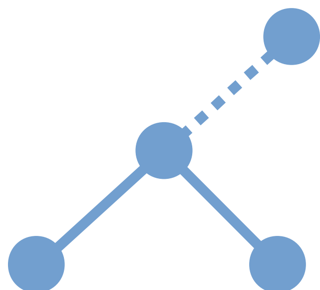


RYU SDN [竜]

Testbed Manual

Version 2.0.4

{17/05/2020}

Diarmuid Ó Briain

INSTITUTE of TECHNOLOGY
CARLOW
Institiúid Teicneolaíochta Cheatharlach

engcore
advancing technology

RYU SDN [竜] Testbed Manual by Diarmuid Ó Briain is licensed under CC BY-SA 4.0

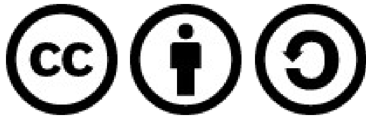


Table of Contents

1. Introduction to Software Defined Networking **9**

2. SDN Architecture **11**

2.1 SDN Controller 11
2.2 NBI Developments 12
2.3 OpenFlow Protocol 12
2.4 Open vSwitch 12
2.5 Hardware OpenFlow switches 12
2.6 Whitebox switches 13
2.7 Ryu SDN Controller 15
2.8 Mininet Emulator 15

3. Setting up the testbed **17**

3.1 Operating system 17
3.2 Install software 19
3.3 RYU testbed desktop 24

4. Build a Mininet test network **27**

4.2 Configuring hosts 33
4.3 Configuring links 33
4.4 OpenFlow traffic review 34

5. A closer view of the Open vSwitch **35**

5.1 OvS Architecture 35
5.2 Establish a simple network topology 37
5.3 OvS management tools 38

6. OpenFlow communications **41**

6.1 Initial OpenFlow handshake negotiation 41
6.2 Webserver test 48

7. RESTful API **51**

7.1 Automating the extraction from RESTful API 52
7.2 Viewing the data from the JSON output from RESTful API 52
7.3 Running the program 55

8. Building a simple test network **57**

9. Ryu Framework **59**

9.1 Testing Ryu 59
9.2 Updating the Ryu controller configuration 63
9.3 Ryu Topology viewer 68
9.4 Ryu Flowmanager 69

10. Custom Topologies **73**

10.1 Create a custom topology 75

11. Custom script to Ryu remote controller....79

11.1 Run Ryu....79
11.2 Start Mininet network....80
11.3 Mininet using Object Oriented Programming....82

12. Developing Ryu applications....85

12.1 Base classes / library....85
12.2 The application class....86
12.3 Event methods....87
12.4 Running the application....88

13. Flow parameters....91

13.1 Flow priority....91
13.2 Timeouts....93

14. OpenFlow pipeline processing....97

14.1 Example pipeline process....97
14.2 Group tables....101
14.3 Proxy Address Resolution Protocol....106

15. Splitting domains....113

15.1 Flowspace slicing....113
15.2 Virtual LANs....119

16. Building a simple L3 and L4 switches....133

16.1 The simple network layer (L3) switch....133
16.2 The simple transport layer (L4) switch....135

17. Bibliography....143

Illustration Index

Illustration 1: SDN Architecture....11
Illustration 2: OpenFlow switch internals....13
Illustration 3: OpenFlow flow fields....14
Illustration 4: Xubuntu Xfce4 desktop....17
Illustration 5: Wireshark view of traffic at the loopback interface....23
Illustration 6: Ryu - mininet testbed....24
Illustration 7: Basic test network....28
Illustration 8: xterm window....31
Illustration 9: iperf testing....32
Illustration 10: Wireshark OpenFlow traffic....34
Illustration 11: Open vSwitch architecture....35
Illustration 12: Initial handshake negotiation OpenFlow v1.3....41
Illustration 13: Switch port status....42
Illustration 14: OF Packet IN and Packet OUT....43
Illustration 15: OFPT_FLOW_MOD....45
Illustration 16: Echo Request and Reply....50
Illustration 17: RESTful API....51
Illustration 18: MAC table extracted using browser....53
Illustration 19: Ryu REST client....54
Illustration 20: Simple test network....57
Illustration 21: Manual flow creation via RESTful API....63
Illustration 22: Ryu curl POST program....66
Illustration 23: Ryu topology viewer....69
Illustration 24: Flowmanager....71
Illustration 25: Custom topology example....74
Illustration 26: Mininet custom topology example....75
Illustration 27: Custom OvS topology....76
Illustration 28: Test network with Ryu....79
Illustration 29: Custom OvS using OOP....83
Illustration 30: OpenFlow pipeline processing....97
Illustration 31: Example pipeline process....97
Illustration 32: OpenFlow Group table....101
Illustration 33: Ryu as a sniffer using the Group table....102
Illustration 34: Testing Ryu as a packet sniffer....105
Illustration 35: Proxy ARP network....107
Illustration 36: VLANs....113
Illustration 37: Mininet custom splice topology....116
Illustration 38: VLAN topology....119
Illustration 39: Mininet custom VLAN topology....130
Illustration 40: L3 switch match logic....134
Illustration 41: L4 switch match logic....139

Table of Abbreviations

API	Application Program Interfaces
CFS	Completely Fair Scheduler
CRUD	Create, Read, Update and Delete
DPID	Data Path IDentifier
HTB	Hierarchical Token Bucket
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IPC	Inter Process Communication
JSON	JavaScript Object Notation
NETCONF	Network Configuration Protocol
NTT	Nippon Telegraph and Telephone Corporation
OF-Config	OpenFlow Management and Configuration Protocol
ONF	Open Networking Foundation
ONOS	Open Network Operating System
OOP	Object Oriented Programming
OVSDB	OvS databases
OvS	Open virtual Switch (often written as Open vSwitch)
QoS	Quality of Service
REST	REpresentational State Transfer
RPC	Remote Procedure Call
SBI	South Bound Interface
SDN	Software Defined Networking
SIC	Software Innovation Centre
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
VM	Virtual Machine
vswitchd	virtual Switch daemon
WSGI	Web Server Gateway Interface
XML	eXtensible Markup Language
YANG	Yet Another Next Generation

Tesbed useful resources

Udemy:	https://github.com/knetsolutions/learn-sdn-with-ryu
Ryu website:	https://osrg.github.io/ryu/
	https://github.com/osrg/ryu/tree/master/ryu
Ryu example apps:	https://github.com/osrg/ryu/tree/master/ryu/app
Documentation:	https://ryu.readthedocs.io/en/latest/index.html
Development mail-list:	https://sourceforge.net/projects/ryu
	To post to this list, send message to: ryu-devel@lists.sourceforge.net
	General information about the mailing list is at: https://lists.sourceforge.net/
Mininet website:	http://mininet.org
Flowmanager:	https://martimy.github.io/flowmanager/

Software

- Ubuntu 20.04
- Ryu 4.32
- Mininet 2.3.0d6
- Open vSwitch 2.13.0
- iperf: 2.0.13
- mtr 0.93
- Wireshark 3.2.3
- Mozilla Firefox 76.0.1
- RESTer 4.1.1

This page is intentionally blank

1. Introduction to Software Defined Networking

Dr. Martin Casado developed an architecture to separate control and forwarding functions of networking devices, migrating control to a centralised policy server [1]. This architecture evolved to what is now known as Software Defined Networking (SDN) today. One of the first challenges was the need for a common South Bound Interface (SBI) protocol between the SDN Controller and the forwarding networking device. OpenFlow developed by the Open Networking Foundation (ONF) and is used over a secure channel (Transport Layer Security (TLS) over Transmission Control Protocol (TCP) port 6633) to modify the group and flow tables in a OpenFlow networking device. OpenFlow has evolved to version 1.5.1.

This page is intentionally blank

2. SDN Architecture

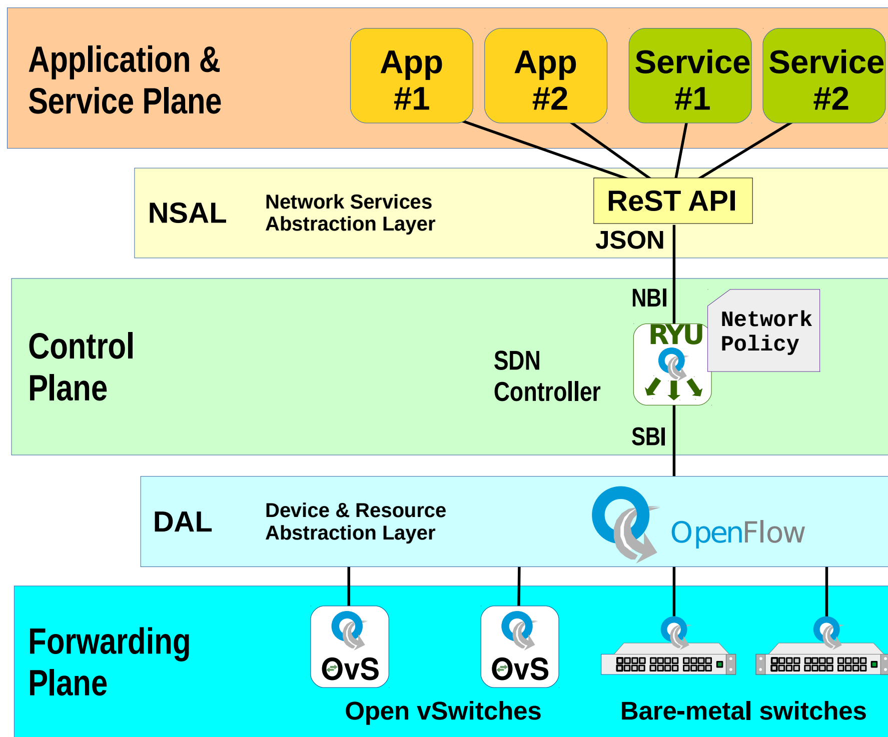


Illustration 1: SDN Architecture

This architecture is demonstrated in Illustration 1 with the data and control planes, linked by OpenFlow offering services from the Application plane via a RESTful Application Programming Interface (API), the North Bound Interface (NBI). A RESTful API is one that uses a REpresentational State Transfer (REST) style web architecture, this is a Stateless, Client-Server model allowing for the Create, Read, Update and Delete (CRUD) operations using Hypertext Transfer Protocol (HTTP) methods; GET, PUT, POST and DELETE [2]. Communication between the SDN Controller and both bare-metal and virtual switches switches are an essential component of SDN. This is achieved over the Device and Resource Abstraction Layer (DAL) on the SBI of the SDN Controller.

2.1 SDN Controller

While there are many SDN Controllers like POX, Project Floodlight, Open Network Operating System (ONOS) and OpenDaylight, this testbed document focuses on the Ryu controller.

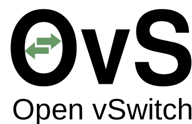
2.2 NBI Developments

As SDN evolves it has become apparent that new NBI mechanisms are required to meet the diverse Applications that will call on the SDN Controller [3]. SDN has facilitated the redesign of the data centre by offering more control of simplified switches. This has the effect of reducing CAPital EXpenditure (CAPEX) while giving more control.

2.3 OpenFlow Protocol

OpenFlow is a simple protocol that the SDN Controller uses over a secure channel (Transport Layer Security (TLS) on either TCP port 6633 or 6653 to modify the flow table in a supporting switch.

OpenFlow has also evolved, coming under the management of the ONF founded in 2011 for the promotion and adoption of SDN through open standards development. OpenFlow has evolved to version 1.5.1 [4], however, hardware typically supports up to v1.3 [5].



Open vSwitch

2.4 Open vSwitch

The OpenFlow protocol and an initial specification in 2008 for a virtual Switch daemon (vswitchd), produced for the GNU/Linux kernel led to the Open virtual Switch (OvS). OvS offers a softswitch solution that operates over OpenFlow and can be used in virtualised situations where a physical switch is unnecessary.

2.5 Hardware OpenFlow switches

Many vendors now offer OpenFlow implementations in their switches. Examples include HP and Netgear. This testbed includes the Netgear ProSAFE Managed Stackable Switch M4300-28G.

2.6 Whitebox switches

One of the attractions of SDN to the large data centre service providers is a means of reducing dependency on vendors like Cisco, Juniper and HP whose switches were very expensive and over featured. Various hardware vendors produced whitebox switches without software which were ideal for building custom OpenFlow implementations. Other vendors such as Netgear have incorporated OpenFlow into their enterprise stacked switch models.

2.6.1 OpenFlow switch internals

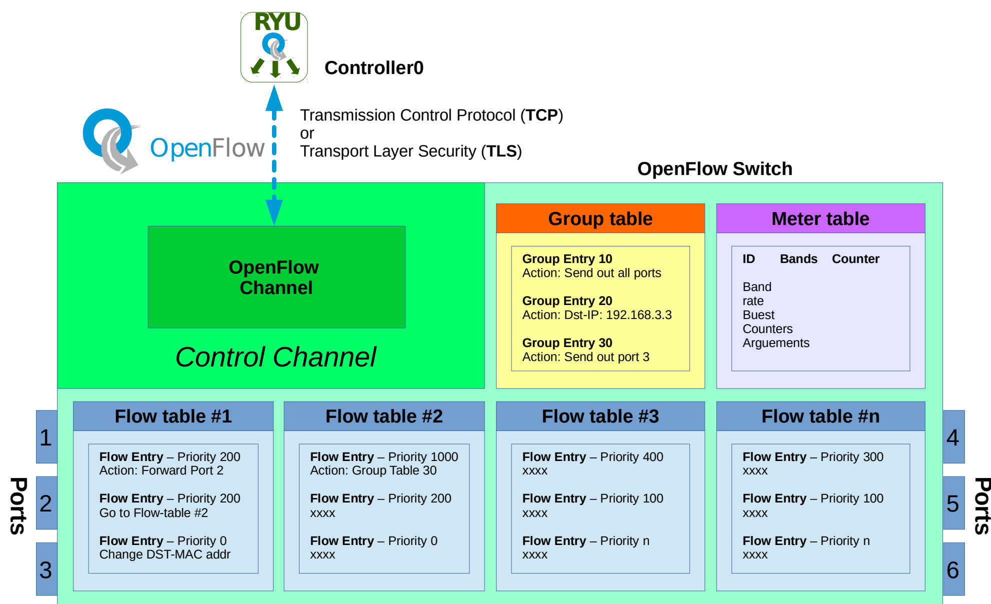


Illustration 2: OpenFlow switch internals

OpenFlow switch internals consists of one or more flow tables and a group table, which perform packet lookups and forwarding. It also includes an OpenFlow channel to an external controller as shown in Illustration 2.

The switch communicates with the controller and the controller manages the switch via the OpenFlow protocol.

The controller uses the OpenFlow protocol to add, update, and delete flow entries in flow tables. It can do this proactively or reactively in response to packets arriving at a switch.

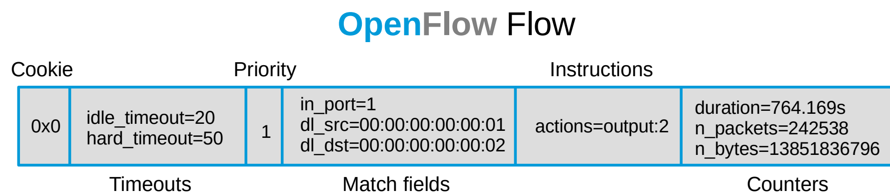


Illustration 3: OpenFlow flow fields

Each flow table in the switch contains a set of flow entries and each flow entry as demonstrated in Illustration 3 consists of:

- **Cookie**: This is an opaque data value chosen by the controller. It may be used by the controller to filter flow.
  - (0x0)
- **Timeouts**: The maximum amount of time or idle time before flow is expired by the switch.
  - **Hard timeout**: A non-zero value that causes the flow entry to be removed after the given number of seconds, regardless of how many packets have been matched.
  - **Idle timeout**: A non-zero value field that causes the flow entry to be removed when it hasn't matched any packets in the given number of seconds.
  - (idle_timeout=20, hard_timeout=50)
- **Priority**: Precedence of the flow entry.
  - (priority=1)
- **Match fields**: To match against packets. These consist of the ingress port, frame and packet header fields, and optionally metadata specified by a previous table.
  - (in_port=1,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02)
- **Instructions**: To modify the action set or pipeline processing. Typically the out port associated with the flow.
  - (actions=output:2)
- **Counters**: to update for matching packets.
  - (duration=764.169s, n_packets=242538, n_bytes=13851836796)

The action in the example specifies that matched packets should be forwarded out port 2 on the switch. Actions associated with flow entries may also direct packets to a group table. Group tables specify additional processing steps. Groups represent sets of actions for flooding, as well as more complex forwarding like multipath, fast reroute, and link aggregation. Group tables can

also be used to enable multiple flow entries to be forwarded to a single identifier, this is common for applications like IP forwarding.

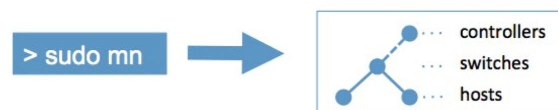
Group entries in the group table have a list of actions grouped together in buckets. The actions in one or more buckets are applied to packets sent to the group.

The meter table entries define per-flow meters. These Per-flow meters enable OpenFlow to implement Quality of Service (QoS) operations, such as rate-limiting, and can be combined with per-port queues to implement complex QoS frameworks, such as DiffServ.

2.7 Ryu SDN Controller

Ryū (竜) is the Japanese word for dragon. Ryu is an SDN Framework developed at Nippon Telegraph and Telephone Corporation (NTT) Laboratories, Software Innovation Centre (SIC) in Japan and first released in 2012. It was released under the Apache 2.0 open-source license.

Ryu provides tools and libraries for conveniently and easily assembling SDN networks and is compatible with various OpenFlow versions 1.0, 1.2, 1.3, 1.4, 1,5 and Nicira extensions. Ryu is considered a component-based, open source software defined by a networking framework and implemented entirely in Python. Another strength of Ryu is that it supports multiple southbound protocols for managing devices, such as OpenFlow, Network Configuration Protocol (NETCONF), OpenFlow Management and Configuration Protocol (OF-Config), and others. It also supports Nicira extensions.



2.8 Mininet Emulator

Mininet is a network emulator which creates a network of virtual hosts, switches, controllers, and links. Mininet hosts run standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software-Defined Networking [6].

This page is intentionally blank

3. Setting up the testbed

3.1 Operating system

Install the xubuntu-20.04-desktop-amd64.iso server version of the Ubuntu 20.04 Long Term Support (LTS) on a computer or Virtual Machine (VM). Xubuntu is Ubuntu sporting the lightweight Xfce4 desktop manager which is ideal for this testbed. If the install is on a very high performing computer and the graphics overhead doesn't matter then install the standard ubuntu desktop if that is more comfortable.

- `Language:` **`English`**
- `Install Xubuntu`
- `Layout:` **`English (UK)`**
- `Variant:` **`English (UK)`**
- `Updates and other software:` **`Download updates while installing Xubuntu`**
- `Updates and other software:` **`Install third-party software`**
- `Guided Storage Configuration:` **`Erase disk and install Xubuntu`**
- `Write changes to disks:` **`Continue`**
- `Where are you:` **`Dublin`**
- `Your name:` **`Ubuntu`**
- `Your computer's name:` **`sdn-mn`**
- `Pick a username:` **`sdn`**
- `Choose a password:` **`sdn`**
- `Confirm your password:` **`sdn`**
- `Log in automatically:` **`checked`**

After installation the desktop is as shown in Illustration 4.

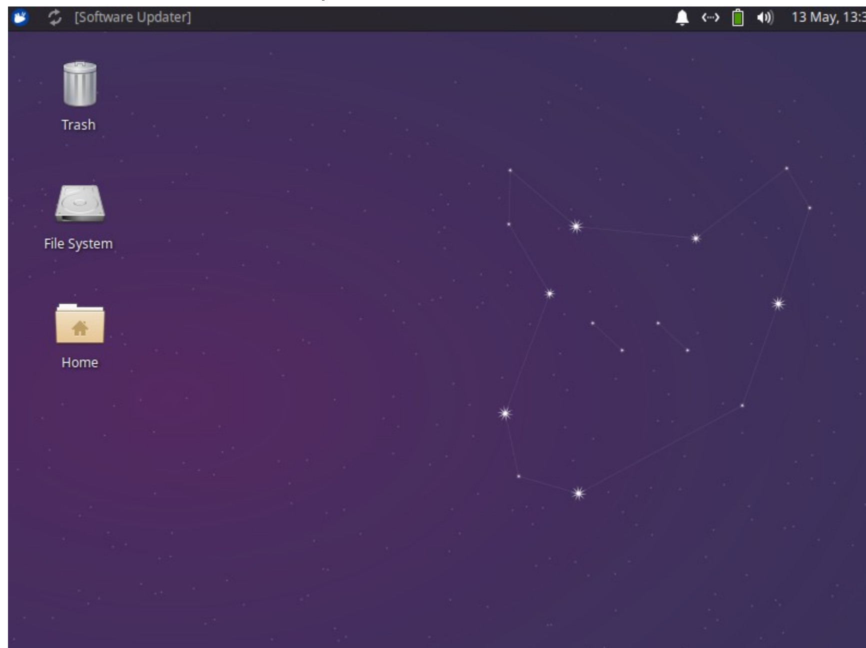


Illustration 4: Xubuntu Xfce4 desktop

3.1.1 Remove unnecessary software

Remove software that is not necessary in the testbed and tidy up.

```
sdn@sdn-mn:~$ sudo apt-get purge -y libreoffice-*
sdn@sdn-mn:~$ sudo apt-get purge -y parole xfburn pulseaudio
sdn@sdn-mn:~$ sudo apt-get purge -y gimp ristretto
sdn@sdn-mn:~$ sudo apt-get purge -y gnome-mines gnome-sudoku sgt-puzzles
sdn@sdn-mn:~$ sudo apt-get purge -y thunderbird
sdn@sdn-mn:~$ sudo apt-get purge -y transmission-* pidgin
sdn@sdn-mn:~$ sudo apt-get -y autoremove
```

3.1.2 VirtualBox Guest Additions

If the install is on VirtualBox then insert the Guest Additions CD to add the guest additions necessary for copy and paste facilities with the host operating system.

```
Devices --->
    Insert Guest Additions CD image...
```

Run the VirtualBox Guest Additions on the VM.

```
sdn@sdn-mn:~$ cd /media/ubuntu/Vbox_Gas_5.2.34
sdn@sdn-mn:/media/ubuntu/VBox_GAs_5.2.34$ sh ./autorun.sh
```

Follow instructions and reboot afterwards.

```
sdn@sdn-mn:~$ reboot
```

This is a good point to make a copy of the VM, export the appliance.

```
File ---> Export Appliance
```

Create a shared folder on the host operating system.

```
user@HOST-COMPUTER~$ mkdir ~/SHARED_DIR
```

On the VM add the user ‘sdn’ to the vboxsf group to access the shared directory.

```
sdn@sdn-mn:~$ sudo usermod -a -G vboxsf sdn
```

In the VirtualBox menubar select the following and link to the directory just created.

```
Devices ---> Shared folders ---> Shared folder settings

☐ Read-only
☑ Auto-mount
☑ Make Permanent

Click <OK> and click <OK> again.

Devices ---> Shared Clipboard ---> Bidirectional
Devices ---> Drag and Drop ---> Bidirectional
```

3.1.3 Install git and sshd

Install git and ssh server.

```
sdn@sdn-mn:~$ sudo apt-get install -y openssh-server git libncurses-dev
libssl-dev
```

3.1.4 Update and Upgrade the system and reboot

Update and upgrade the system. If the install is on a VM then it is recommended to save the image at this stage too.

```
sdn@sdn-mn:~$ sudo apt-get -y update && sudo apt-get -y upgrade
sdn@sdn-mn:~$ sudo shutdown -r now
```

3.2 Install software

3.2.1 Install packages.

```
sdn@sdn-mn:~$ sudo apt-get install -y build-essential mtr tcpdump lynx iperf tshark
fping wireshark net-tools curl openvswitch-switch git python3-pip python3-scapy

---> Configuring wireshark-common
    ---> Should non-superusers be able to capture packets? <Yes>
```

Add the ‘sdn’ user to the wireshark groups.

```
sdn@sdn-mn:~$ sudo usermod -a -G wireshark sdn
```

3.2.2 Setup the Wireshark

```
sdn@sdn-mn:~$ sudo chgrp wireshark /usr/bin/dumpcap
sdn@sdn-mn:~$ sudo chmod 4711 /usr/bin/dumpcap
sdn@sdn-mn:~$ sudo setcap cap_net_admin,cap_net_raw=eip /usr/bin/dumpcap
```

3.2.3 Install mininet

Install mininet with ‘-n’ option to install MiniNet dependencies and core files. The ‘-a’ option installs all packages, however, these have been installed on Ubuntu independently.

```
sdn@sdn-mn:~$ git -C /home/sdn/ clone git://github.com/mininet/mininet
sdn@sdn-mn:~$ sudo /home/sdn/mininet/util/install.sh -a
sdn@sdn-mn:~$ sudo mn --version
2.3.0d6
```

3.2.4 Install Ryu

This installs the latest build of Ryu.

```
sdn@sdn-mn:~$ pip3 install netaddr six pbr
sdn@sdn-mn:~$ pip3 install ryu
```

As an alternative the latest stable build from the operating system repository can be obtained simply by (Note: Do one, not both):

```
sdn@sdn-mn:~$ sudo apt-get install -y python3-ryu
```

The home directory for the ryu application is:

`~/.local/lib/python3.8/site-packages/ryu`

```
sdn@sdn-mn:~$ ls ~/.local/lib/python3.8/site-packages/ryu
app  base  cfg.py  cmd  contrib  controller  exception.py  flags.py  hooks.py
__init__.py  lib  log.py  ofproto  __pycache__  services  tests  topology
utils.py
```

The ryu application executables are located in:

`~/.local/bin`

```
sdn@sdn-mn:~$ ls ~/.local/bin | grep ryu
ryu
ryu-manager
```

3.2.5 Confirm that the testbed is operational

Test the installed software. Open three terminals. Also run Wireshark to monitor the loopback interface.

In terminal number 1 run the Ryu manager for a simple switch.

```
(Window 1) sdn@sdn-mn:~$ 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
```

In terminal number 2 run the Mininet network to interface with the Ryu controller on the loopback IP address at the default port 6653.

```
(Window 2) sdn@sdn-mn:~$ sudo mn --controller remote,ip=127.0.0.1  --switch
ovsk,protocols=OpenFlow13 --mac --ipbase=10.1.1.0/24 --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>
```

In the terminal number 3 review the OvS.

```
sdn@sdn-mn:~$ sudo ovs-vsctl show
8ba60966-6a3b-4696-884d-745a1ab733b4
    Bridge s1
        Controller "tcp:127.0.0.1:6653"
            is_connected: true
        Controller "ptcp:6654"
        fail_mode: secure
        Port s1-eth2
            Interface s1-eth2
        Port s1-eth1
            Interface s1-eth1
        Port s1-eth4
            Interface s1-eth4
        Port s1
            Interface s1
                type: internal
        Port s1-eth3
            Interface s1-eth3
    ovs_version: "2.13.0"


sdn@sdn-mn:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=153.609s, table=0, n_packets=34, n_bytes=2596, priority=0
actions=CONTROLLER:65535
```

In the terminal number 2 ‘pingall’ hosts and then in tab number 3 re-dump the flows.

```
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)
```

This confirms that the switch is operational. Now re-look at the flows and see the influence of the Ryu Controller.

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1
 cookie=0x0, duration=13.584s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01
actions=output:"s1-eth1"
 cookie=0x0, duration=13.563s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02
actions=output:"s1-eth2"
 cookie=0x0, duration=13.543s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01
actions=output:"s1-eth1"
 cookie=0x0, duration=13.528s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03
actions=output:"s1-eth3"
 cookie=0x0, duration=13.511s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01
actions=output:"s1-eth1"
 cookie=0x0, duration=13.493s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:04
actions=output:"s1-eth4"
 cookie=0x0, duration=13.471s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:02
actions=output:"s1-eth2"
 cookie=0x0, duration=13.461s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:03
actions=output:"s1-eth3"
 cookie=0x0, duration=13.440s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:02
actions=output:"s1-eth2"
 cookie=0x0, duration=13.427s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:04
actions=output:"s1-eth4"
 cookie=0x0, duration=13.386s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:03
actions=output:"s1-eth3"
 cookie=0x0, duration=13.374s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:04
actions=output:"s1-eth4"
 cookie=0x0, duration=90.011s, table=0, n_packets=52, n_bytes=3780, priority=0
actions=CONTROLLER:65535
```

Review packet flows on Wireshark as shown in Illustration 5 between the OvS and the Ryu Controller. This demonstrates that the testbed is operational.

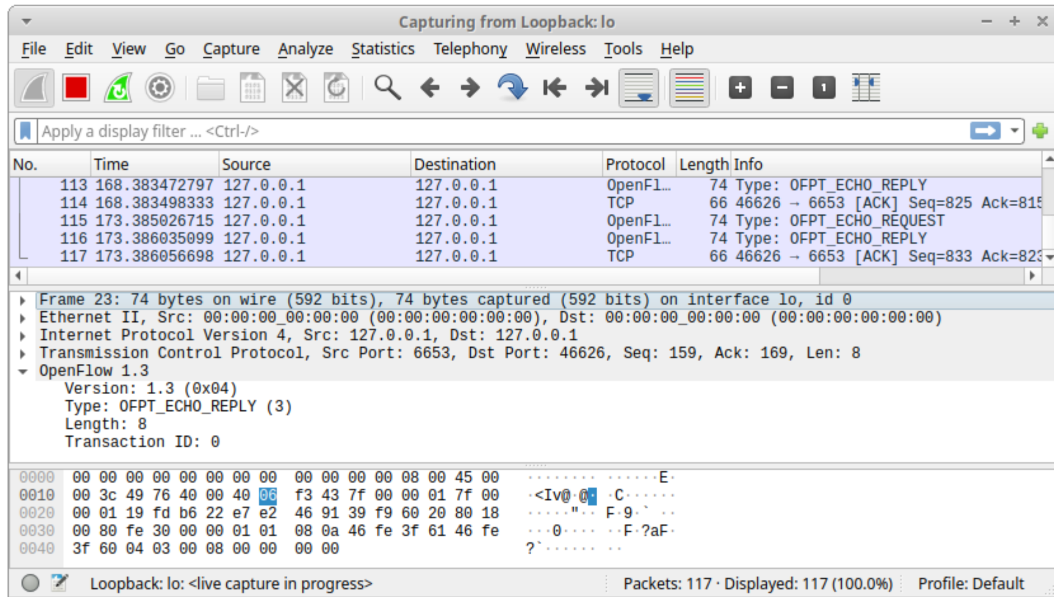


Illustration 5: Wireshark view of traffic at the loopback interface

3.2.6 Mininet exit and cleanup

When the mininet topology is completed using the 'quit' command, it is necessary to clean-up, particularly if it crashed or the terminal closed. Cleaning up removes any tunnels, kernal or OvS datapaths that remain after mininet has quit that may impact on a future running of the emulator.

```
mininet> quit
*** Stopping 1 controllers
c0
*** Stopping 6 terms
*** Stopping 4 links
....
*** Stopping 1 switches
s1
*** Stopping 4 hosts
h1 h2 h3 h4
*** Done
completed in 6.658 seconds
```

```
sdn@sdn-mn:~$ sudo mn --clean
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openflowd
ovs-controllerovs-testcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-
openflowd ovs-controllerovs-testcontroller udpbwtest mnexec ivs ryu-manager
2> /dev/null
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
***  Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
```

3.3 RYU testbed desktop

Illustration 6: Ryu - mininet testbed

Illustration 6 shows the Ryu testbed desktop created as part of this project. The Launcher at the left side gives access to the essential applications required in the testbed.

- *Mozilla Firefox* browser
- *Wireshark* Packet analyser
- *Thunar* file browser
- Terminal window
- *Xfce4-Notes* text editor

This page is intentionally blank

4. Build a Mininet test network

Consider the start-up options for Mininet.

```
sdn@sdn-mn:~$ sudo mn --help
Usage: mn [options]
(type mn -h for details)
The mn utility creates Mininet network from the command line. It can create
parametrized topologies, invoke the Mininet CLI, and run tests.

Options:
  -h, --help            show this help message and exit
  --switch=SWITCH       default|ivs|lxbr|ovs|ovsbr|ovsk|user[,param=value...]
                        ovs=OVSSwitch default=OVSSwitch ovsk=OVSSwitch
                        lxbr=LinuxBridge user=UserSwitch ivs=IVSSwitch
                        ovsbr=OVSBridge
  --host=HOST           cfs|proc|rt[,param=value...]
                        rt=CPULimitedHost{'sched': 'rt'} proc=Host
                        cfs=CPULimitedHost{'sched': 'cfs'}
  --controller=CONTROLLER
                        default|none|nox|ovsc|ref|remote|ryu[,param=value...]
                        ovsc=OVSController none=NullController
                        remote=RemoteController default=DefaultController
                        nox=NOX ryu=Ryu ref=Controller
  --link=LINK           default|ovs|tc[,param=value...] default=Link
                        ovs=OVSLink tc=TCLink
  --topo=TOPO           linear|minimal|reversed|single|torus|tree[,param=value
                        ...] linear=LinearTopo
                        reversed=SingleSwitchReversedTopo tree=TreeTopo
                        single=SingleSwitchTopo torus=TorusTopo
                        minimal=MinimalTopo
  -c, --clean           clean and exit
  --custom=CUSTOM       read custom classes or params from .py file(s)
  --test=TEST           cli|build|pingall|pingpair|iperf|all|iperfudp|none
  -x, --xterms          spawn xterms for each node
  -i IPBASE, --ipbase=IPBASE
                        base IP address for hosts
  --mac                 automatically set host MACs
  --arp                 set all-pairs ARP entries
  -v VERBOSITY, --verbosity=VERBOSITY
                        info|warning|critical|error|debug|output
  --innamespace         sw and ctrl in namespace?
  --listenport=LISTENPORT
                        base port for passive switch listening
  --nolistenport        don't use passive listening port
  --pre=PRE             CLI script to run before tests
  --post=POST           CLI script to run after tests
  --pin                 pin hosts to CPU cores (requires --host cfs or -host rt)
  --nat                 [option=val...] adds a NAT to the topology that
                        connects Mininet hosts to the physical network.
                        Warning: This may route any traffic on the machine
                        that uses Mininet's IP subnet into the Mininet
                        network. If you need to change Mininet's IP subnet,
                        see the --ipbase option.
  --version             prints the version and exits
  --cluster=server1,server2...
                        run on multiple servers (experimental!)
  --placement=block|random
                        node placement for --cluster (experimental!)
```

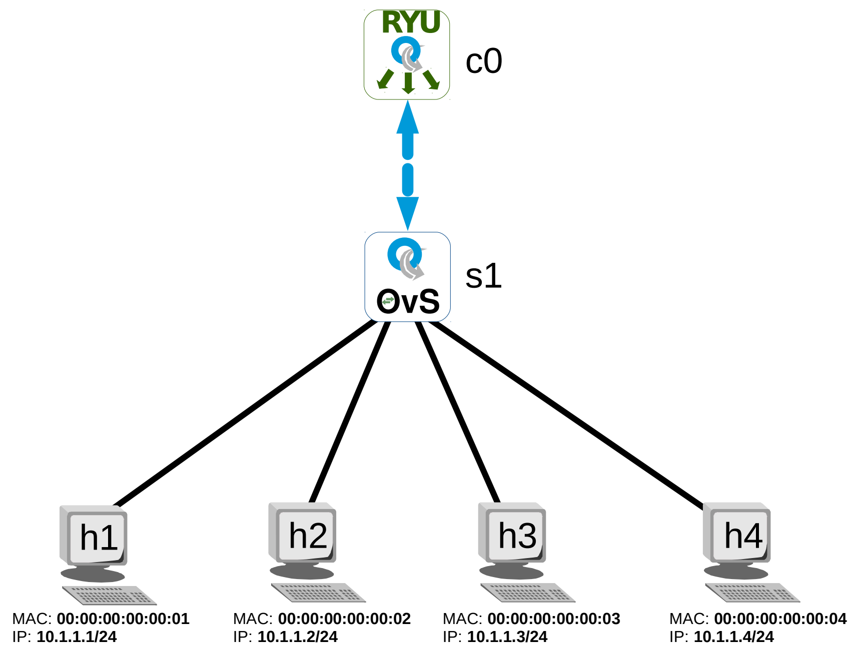


Illustration 7: Basic test network

Establish a basic network with an SDN Controller c0 an OpenFlow switch s1 and four hosts.

Options:

- **Switch**
  - **ivs** - Indigo Virtual Switch
  - **lxbr** - Linux Bridge
  - **ovs** - Open vSwitch
  - **ovsbr** - Open vSwitch in standalone/bridge mode
  - **ovsk** - OpenFlow 1.3 switch
  - **ovsl** - Open vSwitch legacy kernel-space switch using ovs-openflowd
- **Controller**
  - **nox** - Nicira Networks OpenFlow controller
  - **ovsc** - Open vSwitch controller
  - **ref** - OpenFlow reference controller
  - **remote** - Controller running outside of mininet (i.e. Ryu for example)
  - **ryu** - RYU Network Operating System

- **topo**
    - **linear** - Linear topology of k switches, with n hosts per switch
    - **minimal** - Single switch and two hosts
    - **reversed** - Single switch connected to k hosts, with reversed ports, the lowest-numbered host is connected to the highest-numbered port
    - **single** - Single switch connected to k hosts
    - **torus** - 2-D Torus mesh interconnect topology
    - **tree** - a tree network with a given depth and fanout
- **mac** - automatically set host MACs

Establish a Ryu Controller as a switch.

```
sdn@sdn-mn:~$ 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 controller is running

```
sdn@sdn-mn:~$ ss --listening | grep '66[|5]3'
tcp    LISTEN     0      50      *:6653                  *:*
```

Create a basic network to start with from the *mn* Mininet launch command.

```
sdn@sdn-mn:~$ sudo mn --controller remote,ip=127.0.0.1  --switch
ovsk,protocols=OpenFlow13 --mac --ipbase=10.1.1.0/24 --topo
tree,depth=1,fanout=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:
(s1, h1) (s1, h2) (s1, h3) (s1, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

Review the network elements and the links between them.

```
mininet> nodes
available nodes are:
c0 h1 h2 h3 h4 s1

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

mininet> dump
<Host h1: h1-eth0:10.1.1.1 pid=31002>
<Host h2: h2-eth0:10.1.1.2 pid=31004>
<Host h3: h3-eth0:10.1.1.3 pid=31006>
<Host h4: h4-eth0:10.1.1.4 pid=31008>
<OVSSwitch{'protocols': 'OpenFlow13'} s1: lo:127.0.0.1,s1-
eth1:None,s1-eth2:None,s1-eth3:None,s1-eth4:None pid=31013>
<RemoteController{'ip': '127.0.0.1'} c0: 127.0.0.1:6653 pid=11226>
```

4.1.1 Testing network

To run commands on the hosts, use the hostname followed by the command. For example look at the IP address on host h1 and route table on host h2.

```
mininet> h2 ip addr show | grep eth0
2: h2-eth0@if10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
UP group default qlen 1000
    inet 10.1.1.2/24 brd 10.1.1.255 scope global h2-eth0

mininet> h2 ip route
10.1.1.0/24 dev h2-eth0 proto kernel scope link src 10.1.1.2
```

Test connectivity from one host to another.

Test options in the cli are:

- **build**
- **pingall** - Ping between all hosts
- **pingallfull** - Ping between all hosts, including times
- **pingpair** - Ping between first two hosts
- **iperf** <host1> <host2> - Run TCP iperf between two hosts
- **iperfudp** <bw i.e. 100M> <host1> <host2> - UDP iperf

```
mininet> h1 ping -c1 h3
PING 10.1.1.3 (10.1.1.3) 56(84) bytes of data.
64 bytes from 10.1.1.3: icmp_seq=1 ttl=64 time=24.2 ms

--- 10.1.1.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 24.291/24.291/24.291/0.000 ms
```

Look at network links between elements.

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
h4 h4-eth0:s1-eth4
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0 s1-eth4:h4-eth0
c0
```

The Mininet command ‘pingall’ is useful for checking connectivity between hosts.

```
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)
```

‘iperf’ can be ran from the Mininet prompt to test bandwidth between links. Here is an example between *h1* and *h2*.

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['8.70 Gbits/sec', '8.70 Gbits/sec']
```

Run up individual terminals for hosts. Actually if the mininet command is ran with an ‘--xterms ‘ or ‘-x’ switch the terminals for each host and switch will automatically open.

```
mininet> xterm h2
```

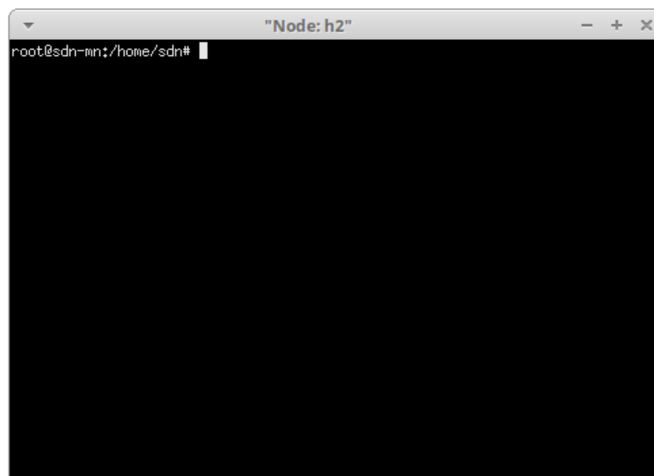


Illustration 8: xterm window

The ‘iperf’ test can be carried out in the traditional manner with two terminals as demonstrated in Illustration 9.

The ‘iperf’ test can be run in User Datagram Protocol (UDP) mode if the server has the ‘--udp’ or ‘-u’ switch added as shown.

On the server side.

```
iperf --server --udp
```

or

```
iperf -s -u
```

On the client side.

```
iperf --client 10.1.1.1 --udp --bandwidth 10m --interval 1 --time 5
```

or

```
iperf -c 10.1.1.1 -u -b 10m -i 1 -t 5
```

Illustration 9: iperf testing

4.1.2 Exiting mininet

To exit Mininet use the *exit* command and then clean-up as described earlier.

```
mininet> exit
*** Stopping 1 controllers
c0
*** Stopping 1 terms
*** Stopping 4 links
....
*** Stopping 1 switches
s1
*** Stopping 4 hosts
h1 h2 h3 h4
*** Done
completed in 349.933 seconds

sdn@sdn-mn:~$ sudo mn --clean
```

4.2 Configuring hosts

Create a network with three hosts similar to the last example; however, this time specify the bandwidth of links and adding delay. The options:

- **host**
    - **cfs** - Completely Fair Scheduler (CFS)
    - **proc**
    - **rt** - Real Time (RT)
        - cpu=<positive fraction, or -1> - CPU bandwidth limit

Note: RT_GROUP_SCHED must be enabled in the kernel to change the *rt,cpu*.

```
sdn@sdn-mn:~$ sudo mn --topo tree,depth=1,fanout=3 --switch ovsk --controller
remote,ip=127.0.0.1 --mac --ipbase=10.1.1.0/24 --host rt,cpu=0.25
```

4.3 Configuring links

Recreate the same, three host, network but this time for traffic control (tc) specify the bandwidth of links and adding delay. The options:

- **link tc**
    - **bw=<value>** - Value in Mb/s
    - **delay=<value>** - Time unit expressed as '5ms', '50us' or '1s'
    - **max_queue_size=<x>** - Queue size in packets
    - **loss=<0 - 100>** - Percentage loss
    - **use_htb=<True | False>** - Hierarchical Token Bucket (HTB) rate limiter

```
(Window 1)sdn@sdn-mn:~$ ryu-manager ryu.app.simple_switch_13

(Window 2) sdn@sdn-mn:~$ sudo mn --topo tree,depth=1,fanout=3 --switch ovsk --
controller remote,ip=127.0.0.1 --mac --ipbase=10.1.1.0/24 --link
tc,bw=100,delay=20ms
*** 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
*** Adding switches:
s1
*** Adding links:
(100.00Mbit 20ms delay) (100.00Mbit 20ms delay) (s1, h1) (100.00Mbit 20ms
delay) (100.00Mbit 20ms delay) (s1, h2) (100.00Mbit 20ms delay) (100.00Mbit
20ms delay) (s1, h3)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...(100.00Mbit 20ms delay) (100.00Mbit 20ms delay) (100.00Mbit 20ms delay)
*** Starting CLI:
```

Repeat the iperf test between hosts h1 and h2. Note the difference in results from the earlier test.

```
*** Results: ['18.0 Gbits/sec', '18.0 Gbits/sec'].

mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['86.4 Mbits/sec', '100 Mbits/sec']
```

4.4 OpenFlow traffic review

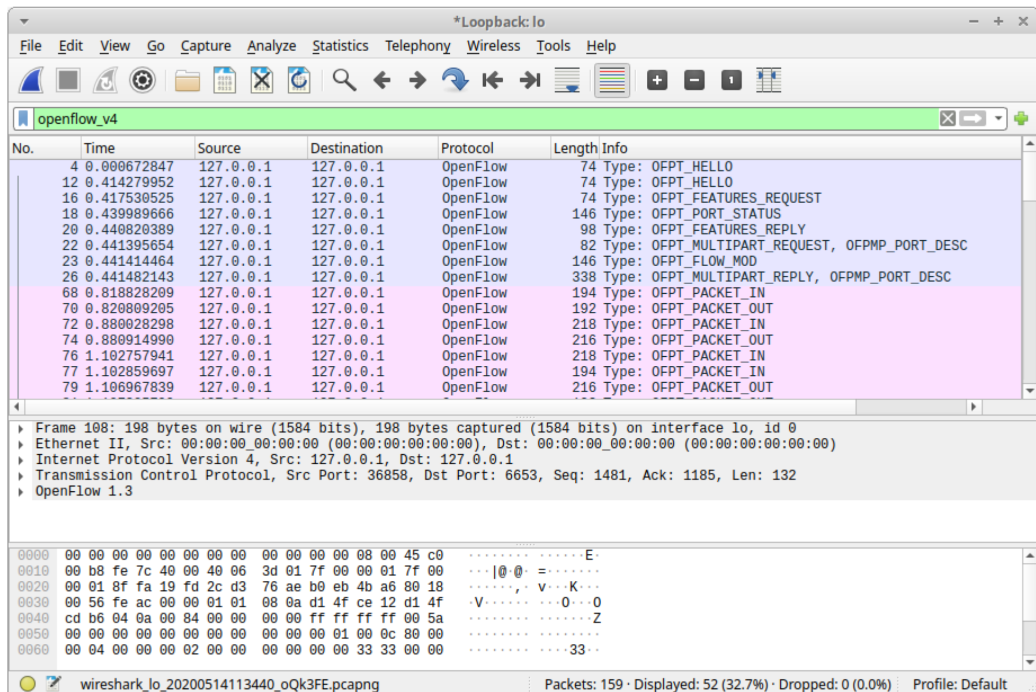


Illustration 10: Wireshark OpenFlow traffic

The OpenFlow traffic between the Controller 'c0' and the OpenFlow switch 's1'. As the controller and switch share the same VM guest the control channel is via the loopback interface, so monitoring the loopback lo0 interface will give access to this messaging. In this case the messaging is using OpenFlow v1.3 so using the filter openflow_v4 will show the communications between the devices.

5. A closer view of the Open vSwitch

5.1 OvS Architecture

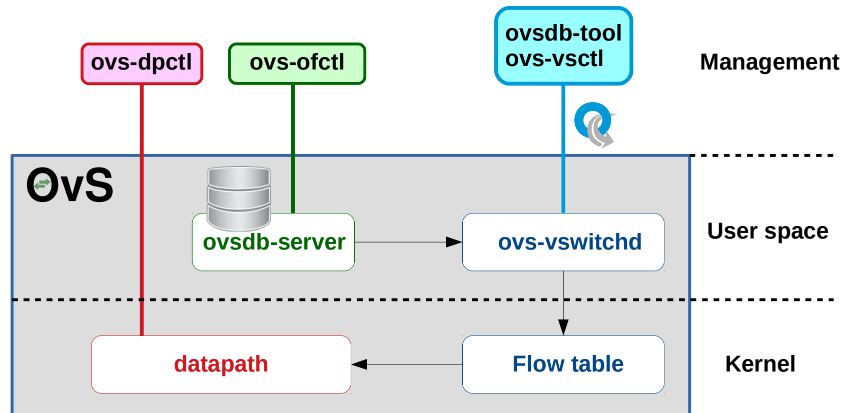


Illustration 11: Open vSwitch architecture

- **ovsdb-server**: This program provides Remote Procedure Call (RPC) interfaces to one or more OpenvSwitch databases (OVSDB).
    - It supports JavaScript Object Notation (JSON) RPC (JSON-RPC) client connections over active or passive TCP/IP or Unix domain sockets. It maintains the switch table database and list of external clients can talk to 'ovsdb-server'.
    - Each OVSDB file may be specified on the command line as database. If none is specified, the default is '/usr/local/etc/openvswitch/conf.db'. The database files must already have been created and initialised using, for example, 'ovsdb-tool create'.
    - ovsdb clients can use the ovsdb management protocol to manipulate the OVSDB tables.
- **ovs-vswitchd**: This is the main OvS userspace program. It is a daemon that manages and controls any number of OvS switches on the local machine.
    - It reads the desired OvS configuration from the 'ovsdb-server' program over an Inter Process Communication (IPC) channel.
    - It also passes certain status and statistical information back into the database.
- **ovs-vsctl**: This is a utility for querying and updating the configuration of ovs-vswitchd (with the help of ovsdb-server). Port configuration, bridge

additions/deletions, bonding, and VLAN tagging are just some of the options that are available with this command.

- **ovs-vsctl –V**: Prints the current version of openvswitch.
- **ovs-vsctl show**: Prints a brief overview of the switch database configuration.
- **ovs-vsctl list-br**: Prints a list of configured bridges
- **ovs-vsctl list-ports <bridge>**: Prints a list of ports on a specific bridge.
- **ovs-vsctl list interface**: Prints a list of interfaces.
- **ovs-vsctl add-br <bridge>**: Creates a bridge in the switch database.

Example configuration of an OvS using 'ovs-vsctl' commands.

```
sdn@sdn-mn:~$ sudo ovs-vsctl show
8ba60966-6a3b-4696-884d-745a1ab733b4
    ovs_version: "2.13.0"
```

In an instance where the Linux hardware has multiple interfaces and control for switching between them is given to an SDN Controller, in this instance running on TCP port 6633.

```
sdn@sdn-mn:~$ ls /sys/class/net

enp0s3  enp0s4  lo

sdn@sdn-mn:~$ sudo ovs-vsctl add-br s1
sdn@sdn-mn:~$ sudo ovs-vsctl add-port s1 enp0s3
sdn@sdn-mn:~$ sudo ovs-vsctl add-port s1 enp0s4
sdn@sdn-mn:~$ sudo ovs-vsctl set-controller s1 tcp:127.0.0.1:6633

sdn@sdn-mn:~$ sudo ovs-vsctl show
8ba60966-6a3b-4696-884d-745a1ab733b4
    Bridge s1
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        Port s1
            Interface s1
                type: internal
        Port enp0s3
            Interface enp0s3
        Port enp0s4
            Interface enp0s4
    ovs_version: "2.13.0"
```

- **ovs-ofctl**: This is a command line tool for monitoring and administering OpenFlow switches. It is used to list implemented flows in the OVS kernel module
    - ovs-ofctl add-flow <bridge> <flow>
    - ovs-ofctl add-flow <bridge> <match-field> actions=all
    - ovs-ofctl del-flows <bridge> <flow>

Example configuration of an OvS using 'ovs-ofctl' commands.

```
sdn@sdn-mn:~$ sudo ovs-ofctl add-flow s1 actions=NORMAL
```

```
sdn@sdn-mn:~$ sudo ovs-ofctl del-flows s1
sdn@sdn-mn:~$ sudo ovs-ofctl add-flow s1 in_port=1,actions=output:2
sdn@sdn-mn:~$ sudo ovs-ofctl add-flow s1 in_port=2,actions=output:1
```

- **ovs-dpctl**: is a command line tool to create, modify, and delete OvS datapaths. This program works only with datapaths that are implemented outside of 'ovs-vswitchd' itself, such as Operating System kernel-based datapaths.
- **ovsdb-tool**: is a command-line program for managing OVSDB files. It does not interact directly with running OvS database servers.
  - ovsdb-client list-dbs
  - ovsdb-client list-tables
  - ovsdb-client get-schema
  - ovsdb-client list-columns
  - ovsdb-client dump

Now delete this bridge and return to the testbed.

```
sdn@sdn-mn:~$ sudo ovs-vsctl del-br s1
sdn@sdn-mn:~$ sudo ovs-vsctl show
8ba60966-6a3b-4696-884d-745a1ab733b4
    ovs_version: "2.13.0"
```

5.2 Establish a simple network topology

```
(Window 1)sdn@sdn-mn:~$ ryu-manager ryu.app.simple_switch_13

(Window 2)sdn@sdn-mn:~$ sudo mn --topo tree,depth=1,fanout=3 --switch ovsk --
controller remote,ip=127.0.0.1 --mac --ipbase=10.1.1.0/24
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(s1, h1) (s1, h2) (s1, h3)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:

mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
```

5.3 OvS management tools

List the switches. The ‘ovs-vswitchd management utility’ can give an overview of database contents. There is only one switch in the topology, the OvS switch s1.

```
(Window 3)sdn@sdn-mn:~$ sudo ovs-vsctl show
8ba60966-6a3b-4696-884d-745a1ab733b4
    Bridge s1
        Controller "tcp:127.0.0.1:6653"
            is_connected: true
        Controller "ptcp:6654"
        fail_mode: secure
        Port s1-eth2
            Interface s1-eth2
        Port s1
            Interface s1
                type: internal
        Port s1-eth3
            Interface s1-eth3
        Port s1-eth1
            Interface s1-eth1
    ovs_version: "2.13.0"
```

From the list of switches get the detailed description of each using the ‘OpenFlow switch management utility’.

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-desc s1
OFPST_DESC reply (OF1.3) (xid=0x2):
Manufacturer: Nicira, Inc.
Hardware: Open vSwitch
Software: 2.13.0
Serial Num: None
DP Description: s1
```

The switch details.

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 show s1
OFPT_FEATURES_REPLY (OF1.3) (xid=0x2): dpid:0000000000000001
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS GROUP_STATS QUEUE_STATS
OFPST_PORT_DESC reply (OF1.3) (xid=0x3):
 1(s1-eth1): addr:22:c8:cd:f9:8f:6e
     config:     0
     state:      LIVE
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s1-eth2): addr:4a:d3:34:85:d8:b3
     config:     0
     state:      LIVE
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 3(s1-eth3): addr:8e:12:3f:bd:a3:a4
     config:     0
     state:      LIVE
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s1): addr:c6:b6:be:aa:e0:4f
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (OF1.3) (xid=0x9): frags=normal miss_send_len=0
```

Port statistics on the switch.

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-ports s1
OFPST_PORT reply (OF1.3) (xid=0x2): 4 ports
  port LOCAL: rx pkts=0, bytes=0, drop=30, errs=0, frame=0, over=0, crc=0
           tx pkts=0, bytes=0, drop=0, errs=0, coll=0
           duration=117.904s
  port  "s1-eth1": rx pkts=19, bytes=1486, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=53, bytes=5125, drop=0, errs=0, coll=0
           duration=117.922s
  port  "s1-eth2": rx pkts=19, bytes=1486, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=53, bytes=5125, drop=0, errs=0, coll=0
           duration=117.923s
  port  "s1-eth3": rx pkts=19, bytes=1486, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=53, bytes=5125, drop=0, errs=0, coll=0
           duration=117.922s
```

Snooping the switch activity.

```
sdn@sdn-mn:~$ sudo ovs-ofctl --verbose snoop s1
2020-05-14T10:44:57Z|00001|stream_unix|DBG|/var/run/openvswitch/s1: connection
failed (No such file or directory)
2020-05-14T10:44:57Z|00002|ofctl|DBG|connecting to
unix:/var/run/openvswitch/s1.snoop
2020-05-14T10:44:57Z|00003|hmap|DBG|../lib/ofp-msgs.c:1381: 1 bucket with 6+
nodes, including 1 bucket with 6 nodes (128 nodes total across 128 buckets)
2020-05-14T10:44:57Z|00004|hmap|DBG|../lib/ofp-msgs.c:1381: 1 bucket with 6+
nodes, including 1 bucket with 6 nodes (256 nodes total across 256 buckets)
2020-05-14T10:44:57Z|00005|hmap|DBG|../lib/ofp-msgs.c:1381: 4 buckets with 6+
nodes, including 1 bucket with 8 nodes (512 nodes total across 512 buckets)
2020-05-14T10:44:57Z|00006|hmap|DBG|../lib/ofp-msgs.c:1381: 8 buckets with 6+
nodes, including 2 buckets with 7 nodes (1024 nodes total across 1024 buckets)
2020-05-14T10:44:57Z|00007|vconn|DBG|unix:/var/run/openvswitch/s1.snoop: sent
(Success): OFPT_HELLO (xid=0x1):
 version bitmap: 0x01
2020-05-14T10:44:57Z|00008|vconn|DBG|unix:/var/run/openvswitch/s1.snoop:
received: OFPT_HELLO (OF1.5) (xid=0xf):
 version bitmap: 0x01, 0x02, 0x03, 0x04, 0x05, 0x06
2020-05-14T10:44:57Z|00009|vconn|DBG|unix:/var/run/openvswitch/s1.snoop:
negotiated OpenFlow version 0x01 (we support version 0x01, peer supports
version 0x06 and earlier)
2020-05-14T10:45:00Z|00010|poll_loop|DBG|wakeup due to [POLLIN] on fd 4
(<->/var/run/openvswitch/s1.snoop) at ../lib/stream-fd.c:157
2020-05-14T10:45:00Z|00011|vconn|DBG|unix:/var/run/openvswitch/s1.snoop:
received: OFPT_ECHO_REQUEST (OF1.3) (xid=0x0): 0 bytes of payload
OFPT_ECHO_REQUEST (OF1.3) (xid=0x0): 0 bytes of payload
2020-05-14T10:45:00Z|00012|poll_loop|DBG|wakeup due to [POLLIN] on fd 4
(<->/var/run/openvswitch/s1.snoop) at ../lib/stream-fd.c:157
2020-05-14T10:45:00Z|00013|vconn|DBG|unix:/var/run/openvswitch/s1.snoop:
received: OFPT_ECHO_REPLY (OF1.3) (xid=0x0): 0 bytes of payload
OFPT_ECHO_REPLY (OF1.3) (xid=0x0): 0 bytes of payload
```

The OvS log is at:

```
sdn@sdn-mn:~$ sudo head /var/log/openvswitch/ovs-vswitchd.log
2020-05-13T16:41:27.974Z|00001|vlog|INFO|opened log file
/var/log/openvswitch/ovs-vswitchd.log
2020-05-13T16:41:28.003Z|00002|ovs_numa|INFO|Discovered 1 CPU cores on NUMA
node 0
2020-05-13T16:41:28.003Z|00003|ovs_numa|INFO|Discovered 1 NUMA nodes and 1 CPU
cores
2020-05-13T16:41:28.003Z|00004|reconnect|INFO|unix:/var/run/openvswitch/
db.sock: connecting...
2020-05-13T16:41:28.003Z|00005|reconnect|INFO|unix:/var/run/openvswitch/
db.sock: connected
2020-05-13T16:41:28.016Z|00006|bridge|INFO|ovs-vswitchd (Open vSwitch) 2.13.0
2020-05-13T16:45:32.370Z|00007|memory|INFO|13192 kB peak resident set size
after 249.9 seconds
2020-05-13T17:39:18.952Z|00008|ofproto_dpif|INFO|system@ovs-system: Datapath
supports recirculation
2020-05-13T17:39:18.952Z|00009|ofproto_dpif|INFO|system@ovs-system: VLAN header
stack length probed as 2
2020-05-13T17:39:18.952Z|00010|ofproto_dpif|INFO|system@ovs-system: MPLS label
stack length probed as 1
```

Check the MAC Table of Switches.

```
~$ sudo ovs-appctl fdb/show s1
 port  VLAN  MAC                Age
```

Dump OpenFlow flows for a switch.

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1
 cookie=0x0, duration=194.361s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01
actions=output:"s1-eth1"
 cookie=0x0, duration=194.351s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02
actions=output:"s1-eth2"
 cookie=0x0, duration=194.337s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01
actions=output:"s1-eth1"
 cookie=0x0, duration=194.328s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03
actions=output:"s1-eth3"
 cookie=0x0, duration=194.317s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:02
actions=output:"s1-eth2"
 cookie=0x0, duration=194.315s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:03
actions=output:"s1-eth3"
 cookie=0x0, duration=217.077s, table=0, n_packets=39, n_bytes=2934, priority=0
actions=CONTROLLER:65535
```

6. OpenFlow communications

6.1 Initial OpenFlow handshake negotiation

As shown in Illustration 12 the Ryu controller and the Switch exchange OFPT_HELLO messages, each can determine the OpenFlow version of the other and work to the lowest common denominator. After this initial exchange each device can exchange further messages using OpenFlow v1.3.

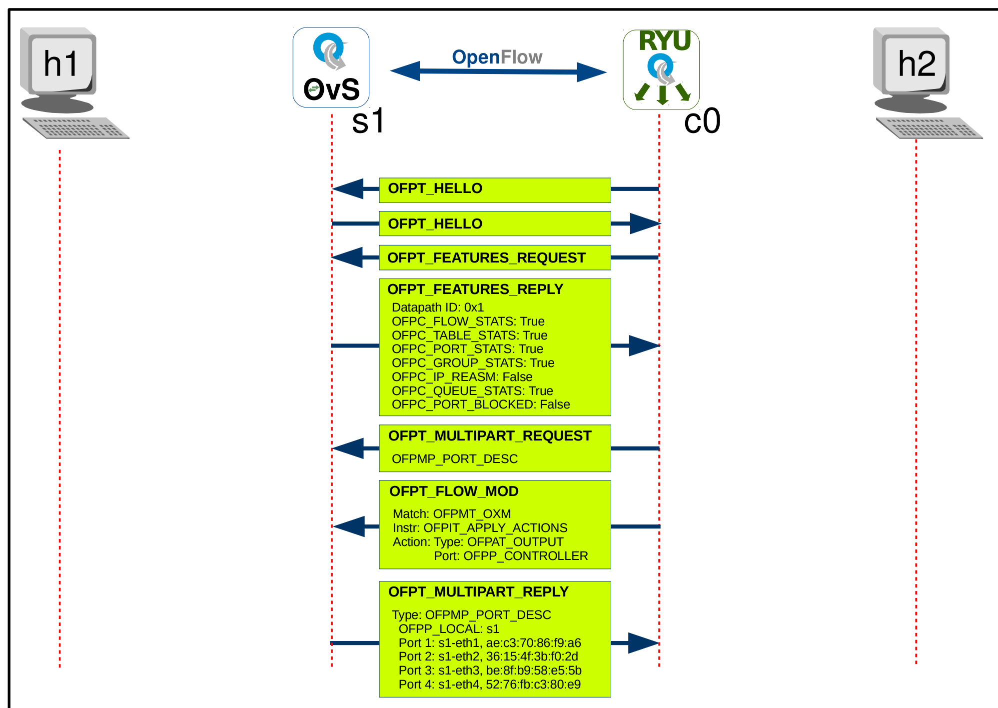


Illustration 12: Initial handshake negotiation OpenFlow v1.3

Next the Ryu controller sends an OFPT_FEATURES_REQUEST message to get the Datapath ID (DPID) and capabilities of the switch. The DPID is a 64 bit number that uniquely identifies a datapath. The lower order 48 bits are used for the switch MAC address, while the higher order 16 bits are defined by the implementer, for example for a VLAN ID.

The switch responds with a OFPT_FEATURES_REPLY which includes the DPID and the capabilities supported by the datapath.

The Ryu controller then sends a multipart request for the switch port descriptions to which it receives a reply with a description of all the ports on the switch that support OpenFlow.

The Ryu controller also sends the switch an OpenFlow Flow Modification that specifies for any Match, output to the switch port connected to the controller, in other words send to the controller.

This can be seen in the OvS too. Once the OvS receives the flow modification it adds it to its flow table.

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1

 cookie=0x0, duration=87.447s, table=0, n_packets=27, n_bytes=2138, priority=0
actions=CONTROLLER:65535
```

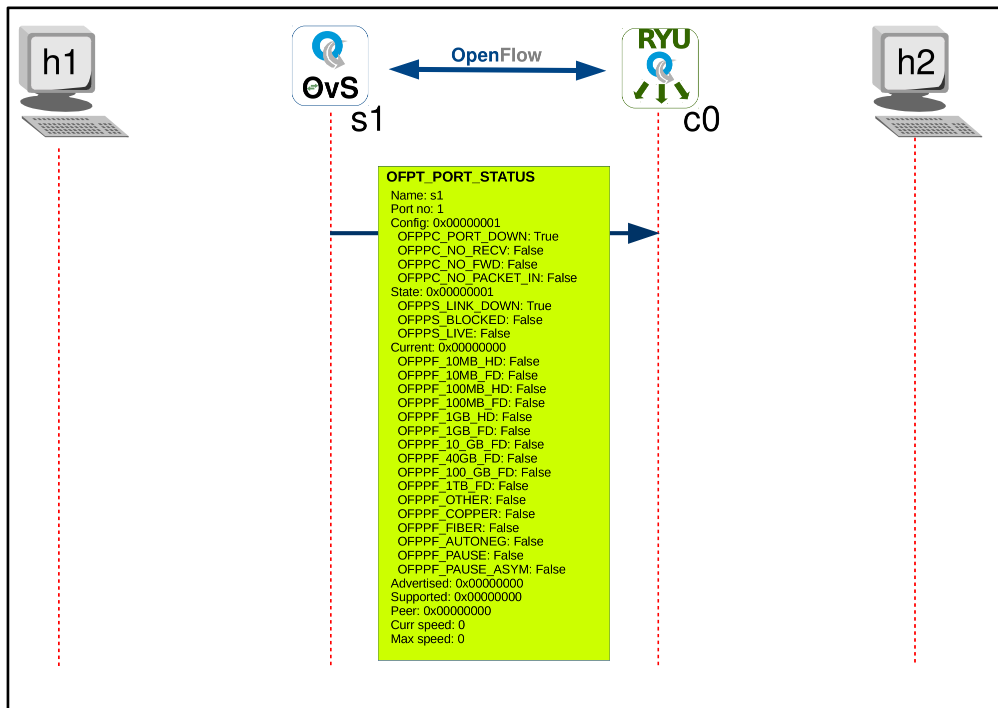


Illustration 13: Switch port status

The switch then sends PFPT_PORT_STATUS messages to the Ryu controller for each of its ports as shown in Illustration 13.

As the switch receives packets, for example in Illustration 14 and Internet Control Message Protocol (ICMP) packet is received at the switch. It carries out a table entry check and finds that it has a 'table-miss' no flow entry. To handle table miss conditions the Ryu controller sends a wildcard flow entry that responds much like a default route in IP routing. It matches all fields and has priority equal to 0. It is called the table-miss flow entry instructing the OpenFlow switch to forward the packets to the controller port (action = output:Controller port).

Here is an example table-miss flow entry in OvS s1.

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1
 cookie=0x0, duration=87.447s, table=0, n_packets=27, n_bytes=2138, priority=0
actions=CONTROLLER:65535
```

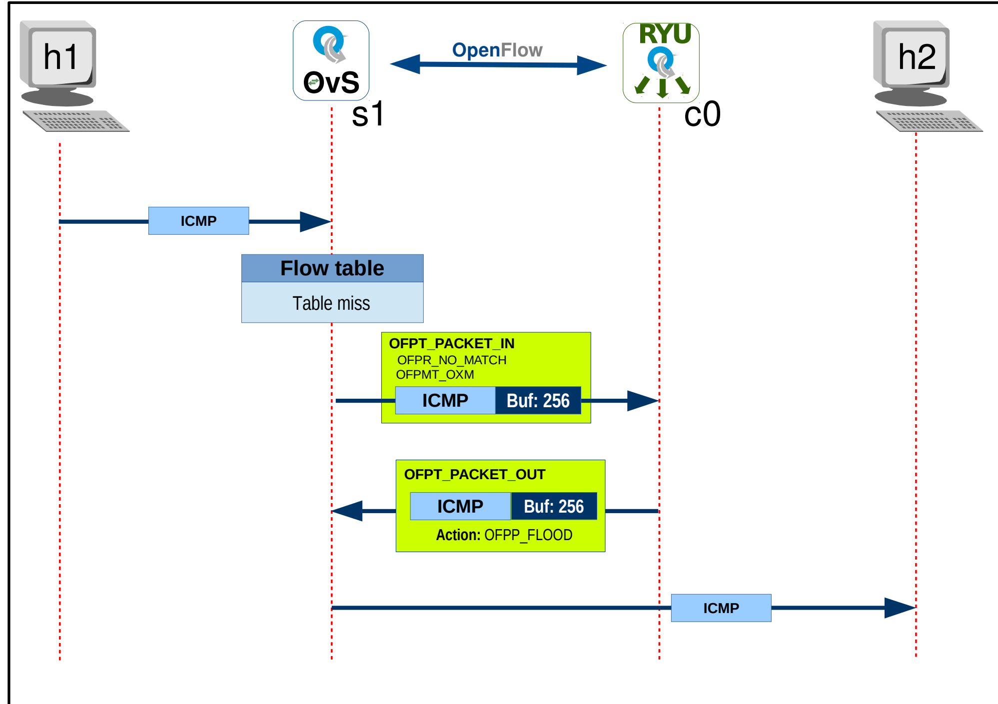


Illustration 14: OF Packet IN and Packet OUT

The OvS then sends an OFPT_PACKET_IN message to the Ryu controller c0 with a unique Buffer Identifier 256 for a decision.

```
Frame: OFPT_PACKET_IN
Ethernet II, Src: s1, Dst: c0
Internet Protocol Version 4
Transmission Control Protocol: Src Port: 40132, Dst Port: 6653
OpenFlow 1.3
    Version: 1.3 (0x04)
    Type: OFPT_PACKET_IN (10)
    Length: 132
    Transaction ID: 0
    Buffer ID: 256
    Total length: 90
    Reason: OFPR_NO_MATCH (0)
    Data
        Ethernet II Packet: ICMP
```

The Ryu controller c0 responds to the OvS s1 switch using the Buffer IDentifier 256 with the decision to flood to all switch all ports.

```
Frame: OFPT_PACKET_OUT
Ethernet II, Src: c0, Dst: s1
Internet Protocol Version 4
Transmission Control Protocol, Src Port: 6653, Dst Port: 40132
OpenFlow 1.3
    Version: 1.3 (0x04)
    Type: OFPT_PACKET_OUT (13)
    Length: 40
    Transaction ID: 3155272041
    Buffer ID: 256
    In port: 4
    Actions length: 16
    Pad: 000000000000
    Action
        Type: OFPAT_OUTPUT (0)
        Length: 16
        Port: OFPP_FLOOD (4294967291)
        Max length: 65509
        Pad: 000000000000
```

Now consider a different type of packet. This is generated by generating traffic from the host h1 to h2.

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['23.8 Gbits/sec', '23.9 Gbits/sec']
```

The packet sent to the Ryu controller via the OFPT_PACKET_IN packet is Address Resolution Protocol (ARP) and therefore the switch must learn the source and destination instead of flooding as in the example for ICMP.

```
Frame: OFPT_PACKET_IN
Ethernet II, Src: s1, Dst: c0
Internet Protocol Version 4
Transmission Control Protocol, Src Port: 40132, Dst Port: 6653
OpenFlow 1.3
    Version: 1.3 (0x04)
    Type: OFPT_PACKET_IN (10)
    Length: 84
    Transaction ID: 0
    Buffer ID: 285
    Total length: 42
    Reason: OFPR_NO_MATCH (0)
    Data
        Ethernet II Packet: ARP
```

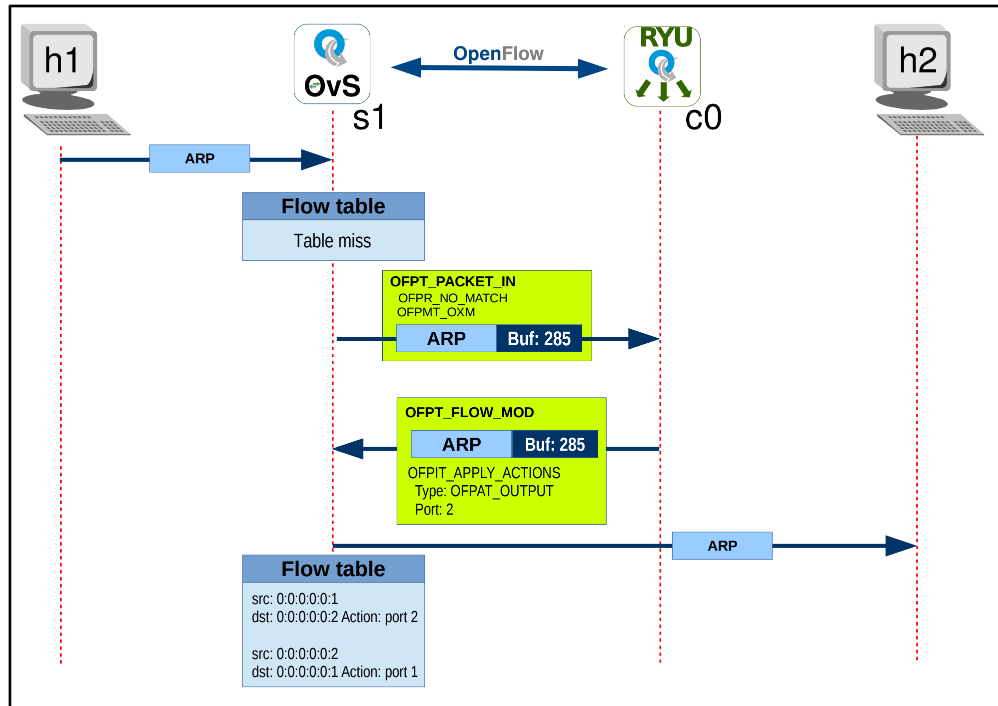


Illustration 15: OFPT_FLOW_MOD

```
Frame 13: OFPT_FLOW_MOD
Ethernet II, Src: c0, Dst: s1
Internet Protocol Version 4
Transmission Control Protocol, Src Port: 6653, Dst Port: 40132
OpenFlow 1.3
    Version: 1.3 (0x04)
    Type: OFPT_FLOW_MOD (14)
    Buffer ID: 286
    Match
        Type: OFPMT_OXM (1)
    Instruction
        Type: OFPIT_APPLY_ACTIONS (4)
        Action
            Type: OFPAT_OUTPUT (0)
            Port: 2
```

When the Ryu controller received an OFPT_PACKET_IN message for an ARP packet. The controller responded as demonstrated in Illustration 15 with a flow modification message OFPT_FLOW_MOD to the OpenFlow switch. This embedded a flow in the OvS as shown by the 'ovs-ofctl dump-flows' command. These flows have two filters. If there is a filter match of a source MAC address of 00:00:00:00:00:02 and a destination MAC address of 00:00:00:00:00:01 then the OpenFlow switch can forward such packets to port 1.

There is a corresponding flow for a source MAC address of 00:00:00:00:00:01 and a destination MAC address of 00:00:00:00:00:02 to be output on port 2.

Any other packet matches are handled by the last rule which has the OvS send an OFPT_PACKET_IN message to the Ryu controller.

This situation remains for and subsequent similar packets and they are forwarded automatically by the OvS until the idle time-out has been exceeded. In this case the flows are dropped by the OvS and the process must be repeated.

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1

 cookie=0x0, duration=34.792s, table=0, n_packets=75237, n_bytes=4965646,
priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01
actions=output:"s1-eth1"
 cookie=0x0, duration=34.783s, table=0, n_packets=242277, n_bytes=12935495194,
priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02
actions=output:"s1-eth2"
 cookie=0x0, duration=230.554s, table=0, n_packets=33, n_bytes=2506, priority=0
actions=CONTROLLER:65535
```

Taking a closer look at the OpenFlow switch. It is possible to set flows directly in the OvS. In this example the flow matches the Ethernet type '0x800' means that 'nw_dst' matches the destination IP address and 'nw_src' if it were included matches the source IP address. The actions specify the port to send the frame out on the OvS.

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 add-flow s1
dl_type=0x0800,nw_dst=10.1.1.3,actions=3

sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 add-flow s1
dl_type=0x0800,nw_dst=10.1.1.4,actions=4
```

Check the flows in the OvS.

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1

 cookie=0x0, duration=21.914s, table=0, n_packets=0, n_bytes=0,
ip,nw_dst=10.1.1.3 actions=output:"s1-eth3"
 cookie=0x0, duration=14.321s, table=0, n_packets=0, n_bytes=0,
ip,nw_dst=10.1.1.4 actions=output:4
 cookie=0x0, duration=86.348s, table=0, n_packets=75237, n_bytes=4965646,
priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01
actions=output:"s1-eth1"
 cookie=0x0, duration=86.339s, table=0, n_packets=242277, n_bytes=12935495194,
priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02
actions=output:"s1-eth2"
 cookie=0x0, duration=282.110s, table=0, n_packets=36, n_bytes=2716, priority=0
actions=CONTROLLER:65535
```

With the following flows set, arry out a ping from host h3 to host h4 and monitor wireshark for OFPT_PACKET_IN. It can also be seen by monitoring the Ryu controller. If a ping is sent from host h3 to host h1. There are no existing flows to handle this event. The ARP is sent to the Ryu controller by the OvS, this can be seen by the ff:ff:ff:ff:ff:ff destination address. Then there are flow modifications sent for mac 00:00:00:00:00:01 ⇒ 00:00:00:00:00:03 out port 3 and for 00:00:00:00:00:03 ⇒ 00:00:00:00:00:01 out port 1.

```
mininet> h3 fping h1
10.1.1.1 is alive

packet in 1 00:00:00:00:00:03 ff:ff:ff:ff:ff:ff 3
packet in 1 00:00:00:00:00:01 00:00:00:00:00:03 1
packet in 1 00:00:00:00:00:03 00:00:00:00:00:01 3
```

The same happens for mac 00:00:00:00:00:01 ⇒ 00:00:00:00:00:04 out port 4 and for 00:00:00:00:00:04 ⇒ 00:00:00:00:00:01 out port 1.

```
mininet> h4 fping h1
10.1.1.1 is alive

packet in 1 00:00:00:00:00:04 ff:ff:ff:ff:ff:ff 4
packet in 1 00:00:00:00:00:01 00:00:00:00:00:04 1
packet in 1 00:00:00:00:00:04 00:00:00:00:00:01 4
```

However, in the case of h3 to h4 or vice versa there is no need for the OpenFlow switch to consult the Ryu controller because it already has manual flows for these events. Hence there is no communication with the Ryu controller yet the ping happens.

```
mininet> h3 fping h4
10.1.1.4 is alive
```

More detail on adjusting flows in OvS can be found at:

http://www.openvswitch.org/support/dist-docs/ovs-ofctl.8.pdf

6.1.1 Mininet testing options

Ping is not the only command that can run on a host. Mininet hosts can run any command or application that is available to the underlying Linux system and its file system. It is possible to enter any bash command, including job control (&, jobs, kill, etc..)

Next, run a simple HTTP server on host h1, making a request from host h3, then shut down the web server.

6.2 Webserver test

Testing with 'lynx' text based web client.

Check the IP addresses of the h1 and h3 hosts. Confirm connectivity between them.

```
mininet> h1 ip addr | grep "inet.*eth0"
    inet 10.1.1.1/24 brd 10.1.1.255 scope global h1-eth0

mininet> h3 ip addr | grep "inet.*eth0"
    inet 10.1.1.3/24 brd 10.1.1.255 scope global h3-eth0

mininet> h1 fping 10.1.1.3
10.1.1.3 is alive

mininet> h1 ping -c1 10.1.1.3
PING 10.1.1.3 (10.1.1.3) 56(84) bytes of data.
64 bytes from 10.1.1.3: icmp_seq=1 ttl=64 time=0.089 ms

--- 10.1.1.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.089/0.089/0.089/0.000 ms
```

xterm to h1 and h3 to gain access to individual shells for each host.

```
mininet> xterm h1
mininet> xterm h3
```

Run a webserver on the host h1 xterm.

```
root@ryu-mn:~# python2 -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

Use lynx on the h3 xterm to access the webserver.

```
root@ryu-mn:~# lynx 10.1.1.1
Directory listing for /
__________________________________________________________________

     * .bash_history
     * .bash_logout
     * .bashrc
     * .cache/
     * .compiz/
     * .config/
     * .dmrc
     * .gconf/
     * .gimp-2.8/
     * .gnupg/
     * .ICEauthority
     * .local/
     * .mininet_history
     * .mozilla/
     * .pam_environment
     * .profile
     * .sudo_as_admin_successful
-- press space for next page --
  Arrow keys: Up and Down to move.  Right to follow a link; Left to go back.
 H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history list
```

On the webserver xterm, h1, the following message is displayed.

```
10.1.1.3 - - [14/May/2020 12:30:34] "GET / HTTP/1.0" 200 -
```

An alternative is to run the server in the Mininet console as a background process and the use lynx to view it.

```
mininet> h1 python2 -m SimpleHTTPServer 80 &
Serving HTTP on 0.0.0.0 port 80 ...

mininet> h3 lynx 10.1.1.1
```

Kill the webserver on host h1.

```
mininet> h1 ps -ef | grep SimpleHTTPServer
root    9769    9610  0 12:31 pts/3    00:00:00 python2 -m SimpleHTTPServer 80
root    9773    9610  0 12:33 pts/3    00:00:00 grep SimpleHTTPServer

mininet> h1 kill 3628
```

6.2.1 Echo Request/Reply Message

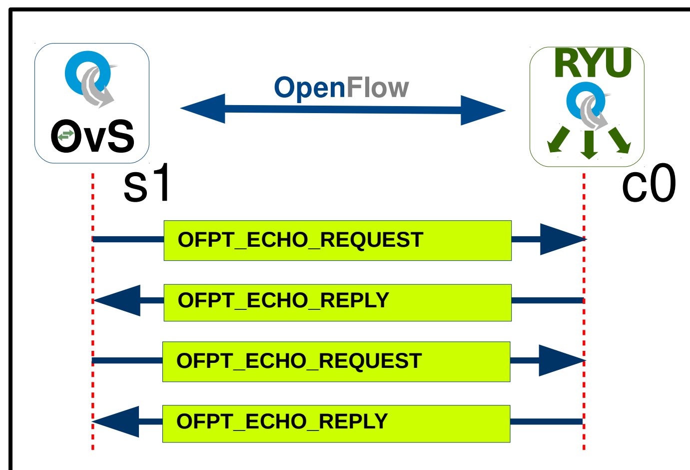


Illustration 16: Echo Request and Reply

As can be seen from Illustration 16, a form of heartbeat the OpenFlow switch sends periodic OFPT_ECHO_REQUEST message to the Ryu controller and it expects a OFPT_ECHO_REPLY response in reply.

7. RESTful API

REST or sometimes known as a RESTful API is an architectural style that defines a set of constraints to be used for creating web services. A Web Server Gateway Interface (WSGI) that conforms to the REST architectural style providing RESTful web services are very common in SDN and are often used to provide a data interface to SDN Controllers. Data is often returned in JSON which is an easy format for computer programs to interact with.

All communication between the client carried out via REST API uses only HTTP request.

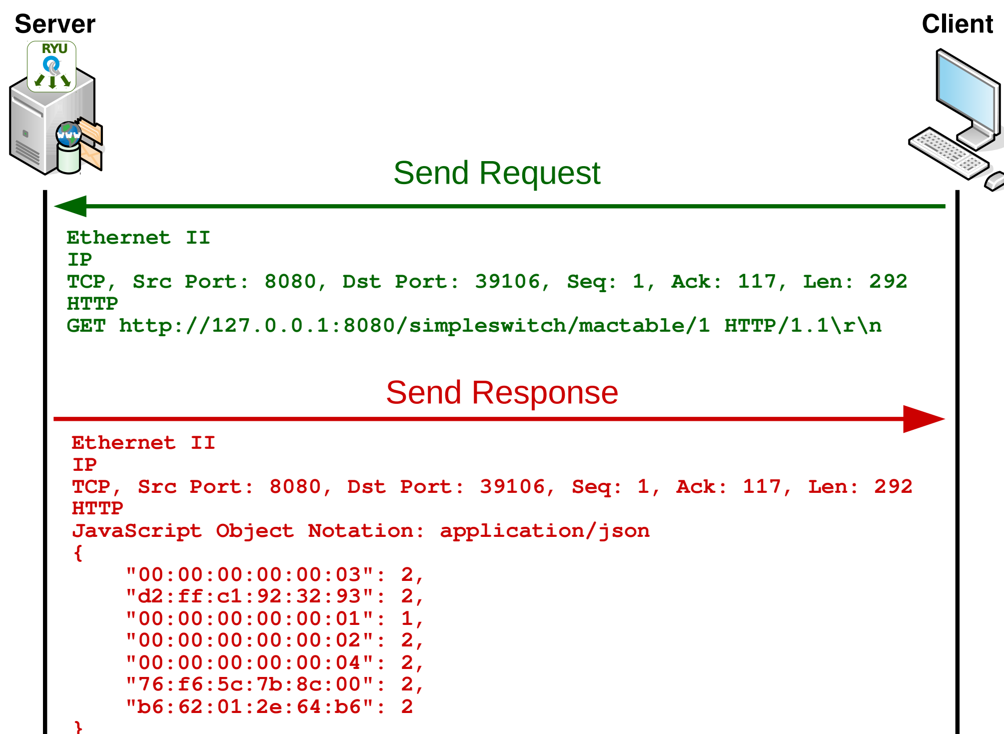


Illustration 17: RESTful API

A request is send from client to server, as demonstrated in Illustration 17, as a HTTP GET. The response from the server is in the form of JSON formatted data.

The RESTful API can respond in many formats like Hypertext Markup Language (HTML), JSON, eXtensible Markup Language (XML) and Yet Another Next Generation (YANG) or YANG. JSON is currently the format being used by Ryu web service API.

HTTP offers five methods which are commonly used in a REST based Architecture

- **POST**: used to create new resources.
- **GET**: used to read from a resource.
- **PUT**: used to update capabilities.
- **DELETE**: used to delete resources.

These correspond to the CRUD operations respectively and allow the server device to be managed remotely.

7.1 Automating the extraction from RESTful API

Demonstration program to extract information from JSON output of RESTful API.

```
(Window 1) sdn@sdn-mn:~$ ryu-manager ryu.app.simple_switch_rest_13
loading app ryu.app.simple_switch_rest_13
loading app ryu.controller.ofp_handler
creating context wsgi
instantiating app ryu.app.simple_switch_rest_13 of SimpleSwitchRest13
instantiating app ryu.controller.ofp_handler of OFPHandler
(3728) wsgi starting up on http://0.0.0.0:8080


(Window 2) sdn@sdn-mn:~$ sudo mn --topo tree,depth=1,fanout=3 --switch ovsk --
controller remote,ip=127.0.0.1 --mac --ipbase=10.1.1.0/24
```

7.2 Viewing the data from the JSON output from RESTful API

Generate some traffic between ports to place MAC addresses in the MAC table.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
```

Additional messages on the ‘ryu-manager’ terminal.

```
packet in 1 00:00:00:00:00:03 33:33:ff:00:00:03 3
packet in 1 00:00:00:00:00:02 33:33:00:00:00:16 2
packet in 1 00:00:00:00:00:01 33:33:00:00:00:16 1
packet in 1 00:00:00:00:00:01 33:33:ff:00:00:01 1
packet in 1 00:00:00:00:00:03 33:33:00:00:00:16 3
packet in 1 00:00:00:00:00:02 33:33:00:00:00:16 2
packet in 1 00:00:00:00:00:02 33:33:00:00:00:02 2
packet in 1 00:00:00:00:00:03 33:33:00:00:00:16 3
packet in 1 00:00:00:00:00:03 33:33:00:00:00:02 3
packet in 1 00:00:00:00:00:03 33:33:00:00:00:16 3
```

7.2.1 cURL command

Attempt to get information from the RESTful API using a 'curl' GET command. 'curl' is a tool used for the transfer of data to or from a server. In this case 'curl' is using HTTP, one of the program's supported protocols.

```
sdn@sdn-mn:~$ curl -X GET
http://127.0.0.1:8080/simpleswitch/mactable/0000000000000001;echo
{"00:00:00:00:00:03": 3, "00:00:00:00:00:02": 2, "00:00:00:00:00:01": 1}
```

While 'curl' retrieves the information, additional messages appear on the 'ryu-manager' terminal.

```
(9873) accepted ('127.0.0.1', 49302)
127.0.0.1 - - [14/May/2020 12:41:10] "GET
/simpleswitch/mactable/0000000000000001 HTTP/1.1" 200 180 0.001291
```

7.2.2 Browsers

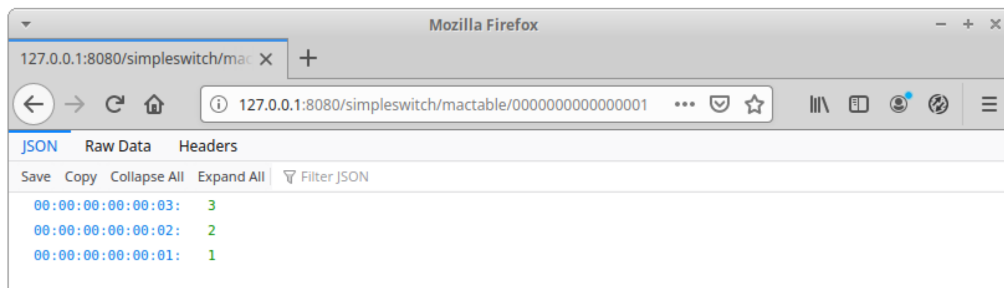


Illustration 18: MAC table extracted using browser

JSON output can be displayed in some Internet browsers. Illustration 18 demonstrates a Mozilla Firefox browser listing a series of three switches under the control of the Ryu Controller.

7.2.3 Python program

Python has a simple HTTP library called 'requests'.

In line 16 the requests get() method is supplied with a URL and the result is extracted as a Response object called 'response'. This object has all the information needed. The 'response.content', is a bytes formatted data and it is necessary to decode this into a string. If the response was positive (i.e. 200 OK) the 'ok()' method applied to the object as 'response.ok' will return 'True'.

The 'json.loads()' method converts the string into a python dictionary in line 21.

The remainder of the program outputs in the form of a dictionary in a number of ways for demonstration.

```
sdn@sdn-mn:~$ cat ryu_rest_client.py

 1 #! /usr/bin/env python3
 2 # -*- coding: utf-8 -*-
 3
 4 import requests
 5 import json
 6
 7 ip_address = '127.0.0.1'
 8 port = '8080'
 9 rest_path = '/simpleswitch/mactable'
10 switch = '0000000000000001'
11 pad = 20
12
13 # Get the JSON output from the RESTful API
14 url = 'http://{}:{}{}/{}'.format(ip_address, port, rest_path, switch)
15
16 response = requests.get(url)
17 response_bytes = response.content
18 response_str = response_bytes.decode('utf-8')
19
20 if (response.ok):
21     json_dict = json.loads(response_str)
22 else:
23     print('There has been a problem connecting to REST API')
24     exit(1)
25
26 # Print equivalent 'curl' command
27 heading = 'Equivalent cURL command to interact with server from server'
28 print('\n{}'.format(heading))
29 print('-' * len(heading),'\n')
30
31 print('  $ curl {}'.format(url))
32
33 # Print out the naked list received
34 heading = 'Naked JSON list received from server'
35 print('\n{}'.format(heading))
36 print('-' * len(heading),'\n')
37
38 print(json_dict)
39
40 # Breakdown list into individual element pairs
41 heading = 'Breakdown list into individual element pairs'
42 print('\n{}'.format(heading))
43 print('-' * len(heading),'\n')
44
45 print ('MAC address {} Port'.format(' ' * 7))
46 print ('{} {} {}'.format('-' * 11, ' ' * 7, '-' * 4))
47 print()
48
49 for mac in sorted(json_dict.keys()):
50     pad_digits = pad - len(mac)
51     print ('{} {} {}'.format(mac, ' ' * pad_digits, json_dict[mac]))
52
53 # End program
54
55 exit(0)
```

Illustration 19: Ryu REST client

7.3 Running the program

```
sdn@sdn-mn:~$ cd example_scripts/
sdn@sdn-mn:~/example_scripts$ ./ryu_rest_client.py

Equivalent cURL command to interact with server from server
-----------------------------------------------------------

  $ curl http://127.0.0.1:8080/simpleswitch/mactable/0000000000000001

Naked JSON list received from server
------------------------------------

{'00:00:00:00:00:03': 3, '00:00:00:00:00:02': 2, '00:00:00:00:00:01': 1}

Breakdown list into individual element pairs
--------------------------------------------

MAC address         Port
-----------         ----

00:00:00:00:00:01     1
00:00:00:00:00:02     2
00:00:00:00:00:03     3
```

This page is intentionally blank

8. Building a simple test network

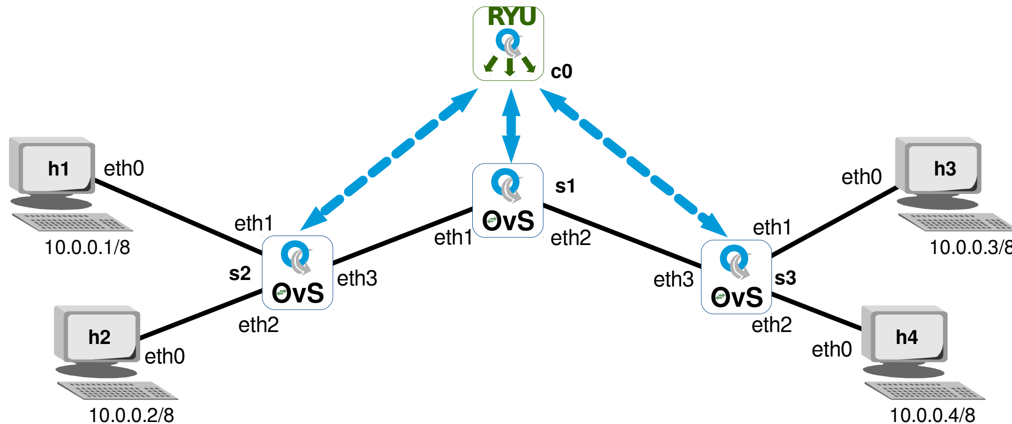


Illustration 20: Simple test network

Illustration 20 demonstrates a simple mininet test network that is built with the following mininet command. There are three switches and the SDN Controller is external to mininet, running on TCP port 6633. This SDN Controller is ran in a separate window.

```
sudo mn --topo tree,depth=2,fanout=2   # Simple 2x2 tree topology
        --switch ovsk                  # Switch is OvS
        --mac                          # Automatically set host MACs
        --controller=remote            # Controller is remote (not mininet)
          ,ip=127.0.0.1                # Controller is on same hardware so loop
          ,port=6633                   # TCP Port to access SDN Controller
```

Derault network is 10.0.0.0/8.

```
(window 1) sdn@sdn-mn:~$ sudo mn --topo tree,depth=2,fanout=2 --switch ovsk
--mac --controller=remote,ip=127.0.0.1,port=6633
```

In this case the controller is a Ryu controller which is located on the same server so is accessed via the loopback address 127.0.0.1 and on port 6633.

```
(window 2) sdn@sdn-mn:~$ 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
```

This page is intentionally blank

9. Ryu Framework

Like the other SDN Controllers in the testbed the Ryu framework [7] provides the SDN components to connect southbound with switches like OvS as well as whitebox OpenFlow switches and well-defined Application Program Interfaces (API). The framework of components approach makes Ryu particularly suited to custom deployments.

9.1 Testing Ryu

Run a Mininet topology where Ryu is the SDN controller running in a separate terminal on the same server.

9.1.1 Running mininet network

Note that starting with the mininet work before running the Ryu controller is commented during the mininet network start as it is unable to contact the controller.

```
(Window 1) sdn@sdn-mn:~$ sudo mn --topo tree,depth=2,fanout=2 --switch ovsk --
controller=remote,ip=127.0.0.1,port=6653 --mac
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3
*** Adding links:
(s1, s2) (s1, s3) (s2, h1) (s2, h2) (s3, h3) (s3, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet>
```

9.1.2 Running the Ryu controller

Ryu operates over on the standard TCP port 6633 by default. To change the port simply add the '--ofp-tcp-listen-port' switch. Additionally enable both the simple switch and the ofctl RESTful API, this is achieved using the '--app-lists' switch.

The 'ryu.app.ofctl_rest' provides REST APIs for retrieving the switch statistics and a means for updating the switch statistics. This application helps debugging Ryu applications and extract various statistics. It supports OpenFlow versions 1.0, 1.2, 1.3, 1.4, 1.5 as well as supporting Nicira extensions.

This example demonstrates running OpenFlow v1.3 which is currently the most popular version across implementation in various manufacturer products.

```
sdn@sdn-mn:~$ ryu-manager --ofp-tcp-listen-port 6653 --wsapi-port 8081
--verbose --app-lists ryu.app.simple_switch_13 ryu.app.ofctl_rest

loading app ryu.app.simple_switch_13
loading app ryu.app.ofctl_rest
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app ryu.app.simple_switch_13 of SimpleSwitch13
instantiating app ryu.app.ofctl_rest of RestStatsApi
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK dpset
  CONSUMES EventOFPStateChange
  CONSUMES EventOFPPortStatus
  CONSUMES EventOFPSwitchFeatures
BRICK SimpleSwitch13
  CONSUMES EventOFPPacketIn
  CONSUMES EventOFPSwitchFeatures
BRICK RestStatsApi
  CONSUMES EventOFPSwitchFeatures
  CONSUMES EventOFPQueueGetConfigReply
  CONSUMES EventOFPRoleReply
  CONSUMES EventOFPStatsReply
  CONSUMES EventOFPDescStatsReply
  CONSUMES EventOFPFlowStatsReply
  CONSUMES EventOFPAggregateStatsReply
  CONSUMES EventOFPTableStatsReply
  CONSUMES EventOFPTableFeaturesStatsReply
  CONSUMES EventOFPPortStatsReply
  CONSUMES EventOFPQueueStatsReply
  CONSUMES EventOFPQueueDescStatsReply
  CONSUMES EventOFPMeterStatsReply
  CONSUMES EventOFPMeterFeaturesStatsReply
  CONSUMES EventOFPMeterConfigStatsReply
  CONSUMES EventOFPGroupStatsReply
  CONSUMES EventOFPGroupFeaturesStatsReply
  CONSUMES EventOFPGroupDescStatsReply
  CONSUMES EventOFPPortDescStatsReply
BRICK ofp_event
  PROVIDES EventOFPStateChange TO {'dpset': {'dead', 'main'}}
  PROVIDES EventOFPPortStatus TO {'dpset': {'main'}}
  PROVIDES EventOFPSwitchFeatures TO {'dpset': {'config'}, 'SimpleSwitch13':
{'config'}, 'RestStatsApi': {'main'}}
  PROVIDES EventOFPPacketIn TO {'SimpleSwitch13': {'main'}}
  PROVIDES EventOFPQueueGetConfigReply TO {'RestStatsApi': {'main'}}
  PROVIDES EventOFPRoleReply TO {'RestStatsApi': {'main'}}
  PROVIDES EventOFPStatsReply TO {'RestStatsApi': {'main'}}
```

```
  PROVIDES EventOFPDescStatsReply TO {'RestStatsApi': {'main'}}
  PROVIDES EventOFPFlowStatsReply TO {'RestStatsApi': {'main'}}
  PROVIDES EventOFPAggregateStatsReply TO {'RestStatsApi': {'main'}}
  PROVIDES EventOFPTableStatsReply TO {'RestStatsApi': {'main'}}
  PROVIDES EventOFPTableFeaturesStatsReply TO {'RestStatsApi': {'main'}}
  PROVIDES EventOFPPortStatsReply TO {'RestStatsApi': {'main'}}
  PROVIDES EventOFPQueueStatsReply TO {'RestStatsApi': {'main'}}
  PROVIDES EventOFPQueueDescStatsReply TO {'RestStatsApi': {'main'}}
  PROVIDES EventOFPMeterStatsReply TO {'RestStatsApi': {'main'}}
  PROVIDES EventOFPMeterFeaturesStatsReply TO {'RestStatsApi': {'main'}}
  PROVIDES EventOFPMeterConfigStatsReply TO {'RestStatsApi': {'main'}}
  PROVIDES EventOFPGroupStatsReply TO {'RestStatsApi': {'main'}}
  PROVIDES EventOFPGroupFeaturesStatsReply TO {'RestStatsApi': {'main'}}
  PROVIDES EventOFPGroupDescStatsReply TO {'RestStatsApi': {'main'}}
  PROVIDES EventOFPPortDescStatsReply TO {'RestStatsApi': {'main'}}
  CONSUMES EventOFPEchoReply
  CONSUMES EventOFPEchoRequest
  CONSUMES EventOFPErrorMsg
  CONSUMES EventOFPHello
  CONSUMES EventOFPPortDescStatsReply
  CONSUMES EventOFPPortStatus
  CONSUMES EventOFPSwitchFeatures
(14126) wsgi starting up on http://0.0.0.0:8081
```

9.1.3 Review the RESTful API

```
sdn@sdn-mn:~$ curl -X GET http://127.0.0.1:8081/stats/switches; echo
[1, 2, 3]

sdn@sdn-mn:~$ curl -X GET http://127.0.0.1:8081/stats/desc/1; echo
{"1": {"mfr_desc": "Nicira, Inc.", "hw_desc": "Open vSwitch", "sw_desc":
"2.13.0", "serial_num": "None", "dp_desc": "s1"}}

sdn@sdn-mn:~$ curl -X GET http://127.0.0.1:8081/stats/flow/1;echo
{"1": [{"priority": 0, "cookie": 0, "idle_timeout": 0, "hard_timeout": 0,
"byte_count": 214, "duration_sec": 1, "duration_nsec": 173000000,
"packet_count": 2, "length": 80, "flags": 0, "actions": ["OUTPUT:CONTROLLER"],
"match": {}, "table_id": 0}]}
```

9.1.4 Test the configuration

Test between the hosts.

```
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)
```

```
sdn@sdn-mn:~$ curl -X GET http://127.0.0.1:8081/stats/flow/1;echo
{"1": [{"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout": 0,
"byte_count": 196, "duration_sec": 3, "duration_nsec": 403000000,
"packet_count": 2, "length": 104, "flags": 0, "actions": ["OUTPUT:1"], "match":
{"in_port": 2, "dl_src": "00:00:00:00:00:03", "dl_dst": "00:00:00:00:00:01"},
"table_id": 0}, {"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout":
0, "byte_count": 98, "duration_sec": 3, "duration_nsec": 389000000,
"packet_count": 1, "length": 104, "flags": 0, "actions": ["OUTPUT:2"], "match":
{"in_port": 1, "dl_src": "00:00:00:00:00:01", "dl_dst": "00:00:00:00:00:03"},
"table_id": 0}, {"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout":
0, "byte_count": 196, "duration_sec": 3, "duration_nsec": 378000000,
"packet_count": 2, "length": 104, "flags": 0, "actions": ["OUTPUT:1"], "match":
{"in_port": 2, "dl_src": "00:00:00:00:00:04", "dl_dst": "00:00:00:00:00:01"},
"table_id": 0}, {"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout":
0, "byte_count": 98, "duration_sec": 3, "duration_nsec": 373000000,
"packet_count": 1, "length": 104, "flags": 0, "actions": ["OUTPUT:2"], "match":
{"in_port": 1, "dl_src": "00:00:00:00:00:01", "dl_dst": "00:00:00:00:00:04"},
"table_id": 0}, {"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout":
0, "byte_count": 196, "duration_sec": 3, "duration_nsec": 346000000,
"packet_count": 2, "length": 104, "flags": 0, "actions": ["OUTPUT:1"], "match":
{"in_port": 2, "dl_src": "00:00:00:00:00:03", "dl_dst": "00:00:00:00:00:02"},
"table_id": 0}, {"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout":
0, "byte_count": 98, "duration_sec": 3, "duration_nsec": 333000000,
"packet_count": 1, "length": 104, "flags": 0, "actions": ["OUTPUT:2"], "match":
{"in_port": 1, "dl_src": "00:00:00:00:00:02", "dl_dst": "00:00:00:00:00:03"},
"table_id": 0}, {"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout":
0, "byte_count": 196, "duration_sec": 3, "duration_nsec": 307000000,
"packet_count": 2, "length": 104, "flags": 0, "actions": ["OUTPUT:1"], "match":
{"in_port": 2, "dl_src": "00:00:00:00:00:04", "dl_dst": "00:00:00:00:00:02"},
"table_id": 0}, {"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout":
0, "byte_count": 98, "duration_sec": 3, "duration_nsec": 302000000,
"packet_count": 1, "length": 104, "flags": 0, "actions": ["OUTPUT:2"], "match":
{"in_port": 1, "dl_src": "00:00:00:00:00:02", "dl_dst": "00:00:00:00:00:04"},
"table_id": 0}, {"priority": 0, "cookie": 0, "idle_timeout": 0, "hard_timeout":
0, "byte_count": 2504, "duration_sec": 56, "duration_nsec": 261000000,
"packet_count": 35, "length": 80, "flags": 0, "actions": ["OUTPUT:CONTROLLER"],
"match": {}, "table_id": 0}]}
```

9.2 Updating the Ryu controller configuration

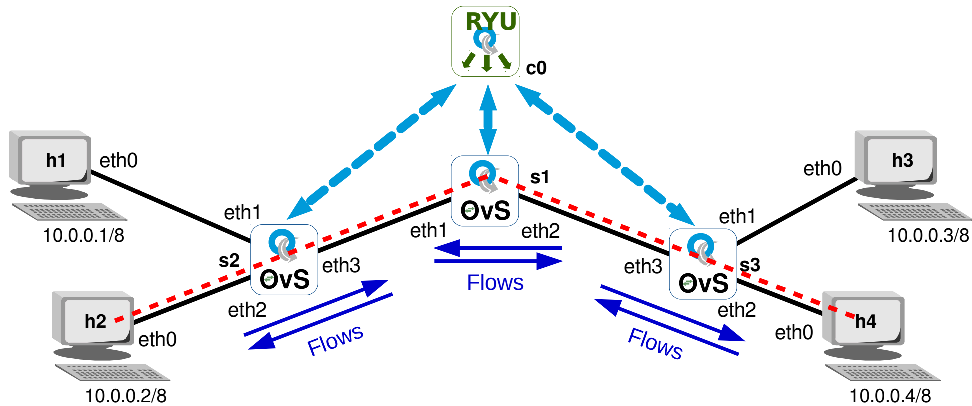


Illustration 21: Manual flow creation via RESTful API

Consider the dashed red line in Illustration 21, to create a flow from host h2 to host h4 the following flows marked with blue arrows are necessary. These are:

Switch	In port	Source MAC	Destination MAC	Out port
s1	1	00:00:00:00:00:02	00:00:00:00:00:04	2
s1	2	00:00:00:00:00:04	00:00:00:00:00:02	1
s2	2	00:00:00:00:00:02	00:00:00:00:00:04	3
s2	3	00:00:00:00:00:04	00:00:00:00:00:02	2
s3	3	00:00:00:00:00:02	00:00:00:00:00:04	2
s3	2	00:00:00:00:00:04	00:00:00:00:00:02	3

It is possible to add these flows to OvS via the RESTful API on the Ryu controller using the HTTP POST method. Here is a demonstration using 'curl' for one flow entry on OvS s1.

```
sdn@sdn-mn:~$ curl -X POST -d '{
    "dpid": 1,
    "cookie": 1,
    "cookie_mask": 1,
    "table_id": 0,
    "idle_timeout": 3000,
    "hard_timeout": 3000,
    "priority": 1,
    "flags": 1,
    "match":{
        "in_port": 1,
        "dl_dst": "00:00:00:00:00:04",
        "dl_src": "00:00:00:00:00:02"
    },
    "actions":[
        {
            "type":"OUTPUT",
            "port": 2
        }
    ]
 }' http://localhost:8081/stats/flowentry/add
```

This can be verified by using a GET method request from the Ryu controller RESTful API or directly from the OvS itself.

```
sdn@sdn-mn:~$ curl -X GET http://127.0.0.1:8081/stats/flow/1; echo
{"1": [{"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout": 0,
"byte_count": 238, "duration_sec": 56, "duration_nsec": 509000000,
"packet_count": 3, "length": 104, "flags": 0, "actions": ["OUTPUT:1"], "match":
{"in_port": 2, "dl_src": "00:00:00:00:00:03", "dl_dst": "00:00:00:00:00:01"},
"table_id": 0}, {"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout":
0, "byte_count": 140, "duration_sec": 56, "duration_nsec": 495000000,
"packet_count": 2, "length": 104, "flags": 0, "actions": ["OUTPUT:2"], "match":
{"in_port": 1, "dl_src": "00:00:00:00:00:01", "dl_dst": "00:00:00:00:00:03"},
"table_id": 0}, {"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout":
0, "byte_count": 238, "duration_sec": 56, "duration_nsec": 484000000,
"packet_count": 3, "length": 104, "flags": 0, "actions": ["OUTPUT:1"], "match":
{"in_port": 2, "dl_src": "00:00:00:00:00:04", "dl_dst": "00:00:00:00:00:01"},
"table_id": 0}, {"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout":
0, "byte_count": 140, "duration_sec": 56, "duration_nsec": 479000000,
"packet_count": 2, "length": 104, "flags": 0, "actions": ["OUTPUT:2"], "match":
{"in_port": 1, "dl_src": "00:00:00:00:00:01", "dl_dst": "00:00:00:00:00:04"},
"table_id": 0}, {"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout":
0, "byte_count": 238, "duration_sec": 56, "duration_nsec": 452000000,
"packet_count": 3, "length": 104, "flags": 0, "actions": ["OUTPUT:1"], "match":
{"in_port": 2, "dl_src": "00:00:00:00:00:03", "dl_dst": "00:00:00:00:00:02"},
"table_id": 0}, {"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout":
0, "byte_count": 140, "duration_sec": 56, "duration_nsec": 439000000,
"packet_count": 2, "length": 104, "flags": 0, "actions": ["OUTPUT:2"], "match":
{"in_port": 1, "dl_src": "00:00:00:00:00:02", "dl_dst": "00:00:00:00:00:03"},
"table_id": 0}, {"priority": 1, "cookie": 0, "idle_timeout": 0, "hard_timeout":
0, "byte_count": 238, "duration_sec": 56, "duration_nsec": 413000000,
"packet_count": 3, "length": 104, "flags": 0, "actions": ["OUTPUT:1"], "match":
{"in_port": 2, "dl_src": "00:00:00:00:00:04", "dl_dst": "00:00:00:00:00:02"},
"table_id": 0}, {"priority": 1, "cookie": 1, "idle_timeout": 3000,
"hard_timeout": 3000, "byte_count": 140, "duration_sec": 10, "duration_nsec":
396000000, "packet_count": 2, "length": 104, "flags": 1, "actions":
["OUTPUT:2"], "match": {"in_port": 1, "dl_src": "00:00:00:00:00:02", "dl_dst":
"00:00:00:00:00:04"}, "table_id": 0}, {"priority": 0, "cookie": 0,
"idle_timeout": 0, "hard_timeout": 0, "byte_count": 2998, "duration_sec": 109,
"duration_nsec": 367000000, "packet_count": 41, "length": 80, "flags": 0,
"actions": ["OUTPUT:CONTROLLER"], "match": {}, "table_id": 0}]}

sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):

cookie=0x1, duration=2295.871s, table=0, n_packets=2, n_bytes=140,
idle_timeout=3000, hard_timeout=3000, send_flow_rem priority=1,in_port="s1-
eth1",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:04 actions=output:"s1-
eth2"
 cookie=0x0, duration=2394.842s, table=0, n_packets=80, n_bytes=6098,
priority=0 actions=CONTROLLER:65535
```

Obviously this process could be repeated for each flow. Illustration 22 shows a short python program that will automate this. It would also be simple to solve this problem using a bash shell script and one is included with this document for reference.

```
sdn@sdn-mn:~$ cat ryu_curl_post.py
```

```
 1 #! /usr/bin/env python3
 2 # -*- coding: utf-8 -*-
 3
 4 import requests
 5 import json
 6
 7 # Declare variables
 8 ryu_ip = '127.0.0.1'
 9 ryu_port = '8081'
10 timer = '3000'
11 priority = '1'
12 switches = list()
13
14 # Dataset to be uploaded
15 data = [[ '1', {'IN_PORT':'1','SRC':'2','DST':'4','OUT_PORT':'2'},
16                {'IN_PORT':'2','SRC':'4','DST':'2','OUT_PORT':'1'}],
17         [ '2', {'IN_PORT':'2','SRC':'2','DST':'4','OUT_PORT':'3'},
18                {'IN_PORT':'3','SRC':'4','DST':'2','OUT_PORT':'2'}],
19         [ '3', {'IN_PORT':'3','SRC':'2','DST':'4','OUT_PORT':'2'},
20                {'IN_PORT':'2','SRC':'4','DST':'2','OUT_PORT':'3'}]]
21
22 # Template flow entry
23 flow_entry = """{
24     "dpid": %s,
25     "table_id": 0,
26     "idle_timeout": %s,
27     "hard_timeout": %s,
28     "priority": %s,
29     "match":{
30         "in_port": %s,
31         "dl_src": "00:00:00:00:00:0%s",
32         "dl_dst": "00:00:00:00:00:0%s"
33     },
34     "actions":[
35         {
36             "type":"OUTPUT",
37             "port": %s
38         }
39     ]
40 }"""
41
42 # Process dataset and upload to Ryu controller
43 url_post = 'http://{}:{}/stats/flowentry/add'.format(ryu_ip,ryu_port)
44
```

```
44
45 for switch in data:
46     switches.append(switch[0])
47     dpid = switch[0]
48     print('\nUploading flows for OvS s{}'.format(dpid))
49     iter_flows = iter(switch)
50     next(iter_flows)
51     for flow in iter_flows:
52         flow_items = (flow['IN_PORT'],flow['SRC'],flow['DST'],flow['OUT_PORT'])
53         all_items = (dpid,timer,timer,priority) + flow_items
54         this_flow = flow_entry % all_items  # Replace all items in tuple for &s
55         this_flow_dict = json.loads(this_flow)  # Convert str to dict
56
57         # Send URL to Ryu controller
58         response = requests.post(url_post, data=this_flow)
59         if (response.ok):
60             print('In port: %s, SRC: %s DST: %s Out port: %s' % flow_items)
61         else:
62             print('There is a problem connecting to REST API')
63             exit(1)
64
65 print('\nAll flows added')
66
67 # Get the JSON output from the RESTful API confirming flows
68 for switch in switches:
69     url_get = 'http://{}:{}/stats/flow/{}'.format(ryu_ip, ryu_port, switch)
70     response = requests.get(url_get)
71     response_bytes = response.content
72     response_str = response_bytes.decode('utf-8')
73     print()
74     if (response.ok):
75         json_dict = json.loads(response_str)
76         print(json_dict)
77         print()
78     else:
79         print('There has been a problem connecting to REST API')
80         exit(1)
81
82 print('''
83 You can also check the flows with the following commands:
84
85   sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1
86   sudo ovs-ofctl --protocols OpenFlow13 dump-flows s2
87   sudo ovs-ofctl --protocols OpenFlow13 dump-flows s3
88 ''')
89
90 exit(0)
```

Illustration 22: Ryu curl POST program

Run the python program to upload the flows in the dataset. The program will get the flows afterwards for confirmation. A ping in ‘mininet’ between the hosts h2 and h4 without the Ryu controller having to make flow modifications. It can be easily confirmed afterwards that now new flows have been added by the controller.

```
mininet> h2 fping h4
10.0.0.4 is alive
```

```
sdn@sdn-mn:~$ cd example_scripts/
sdn@sdn-mn:~/example_scripts$ ./ryu_curl_post.py

Uploading flows for OvS s1
In port: 1, SRC: 2 DST: 4 Out port: 2
In port: 2, SRC: 4 DST: 2 Out port: 1

Uploading flows for OvS s2
In port: 2, SRC: 2 DST: 4 Out port: 3
In port: 3, SRC: 4 DST: 2 Out port: 2

Uploading flows for OvS s3
In port: 3, SRC: 2 DST: 4 Out port: 2
In port: 2, SRC: 4 DST: 2 Out port: 3

All flows added

{'1': [{'actions': ['OUTPUT:2'], 'idle_timeout': 3000, 'table_id': 0, 'cookie':
0, 'length': 104, 'packet_count': 0, 'duration_sec': 0, 'match': {'dl_src':
'00:00:00:00:00:02', 'in_port': 1, 'dl_dst': '00:00:00:00:00:04'},
'hard_timeout': 3000, 'byte_count': 0, 'flags': 0, 'duration_nsec': 45000000,
'priority': 1}, {'actions': ['OUTPUT:1'], 'idle_timeout': 3000, 'table_id': 0,
'cookie': 0, 'length': 104, 'packet_count': 0, 'duration_sec': 0, 'match':
{'dl_src': '00:00:00:00:00:04', 'in_port': 2, 'dl_dst': '00:00:00:00:00:02'},
'hard_timeout': 3000, 'byte_count': 0, 'flags': 0, 'duration_nsec': 40000000,
'priority': 1}, {'actions': ['OUTPUT:CONTROLLER'], 'idle_timeout': 0,
'table_id': 0, 'cookie': 0, 'length': 80, 'packet_count': 47, 'duration_sec':
4, 'match': {}, 'hard_timeout': 0, 'byte_count': 5964, 'flags': 0,
'duration_nsec': 712000000, 'priority': 0}]}


{'2': [{'actions': ['OUTPUT:3'], 'idle_timeout': 3000, 'table_id': 0, 'cookie':
0, 'length': 104, 'packet_count': 0, 'duration_sec': 0, 'match': {'dl_src':
'00:00:00:00:00:02', 'in_port': 2, 'dl_dst': '00:00:00:00:00:04'},
'hard_timeout': 3000, 'byte_count': 0, 'flags': 0, 'duration_nsec': 64000000,
'priority': 1}, {'actions': ['OUTPUT:2'], 'idle_timeout': 3000, 'table_id': 0,
'cookie': 0, 'length': 104, 'packet_count': 0, 'duration_sec': 0, 'match':
{'dl_src': '00:00:00:00:00:04', 'in_port': 3, 'dl_dst': '00:00:00:00:00:02'},
'hard_timeout': 3000, 'byte_count': 0, 'flags': 0, 'duration_nsec': 57000000,
'priority': 1}, {'actions': ['OUTPUT:CONTROLLER'], 'idle_timeout': 0,
'table_id': 0, 'cookie': 0, 'length': 80, 'packet_count': 47, 'duration_sec':
4, 'match': {}, 'hard_timeout': 0, 'byte_count': 5952, 'flags': 0,
'duration_nsec': 728000000, 'priority': 0}]}


{'3': [{'actions': ['OUTPUT:2'], 'idle_timeout': 3000, 'table_id': 0, 'cookie':
0, 'length': 104, 'packet_count': 0, 'duration_sec': 0, 'match': {'dl_src':
'00:00:00:00:00:02', 'in_port': 3, 'dl_dst': '00:00:00:00:00:04'},
'hard_timeout': 3000, 'byte_count': 0, 'flags': 0, 'duration_nsec': 52000000,
'priority': 1}, {'actions': ['OUTPUT:3'], 'idle_timeout': 3000, 'table_id': 0,
'cookie': 0, 'length': 104, 'packet_count': 0, 'duration_sec': 0, 'match':
{'dl_src': '00:00:00:00:00:04', 'in_port': 2, 'dl_dst': '00:00:00:00:00:02'},
'hard_timeout': 3000, 'byte_count': 0, 'flags': 0, 'duration_nsec': 43000000,
'priority': 1}, {'actions': ['OUTPUT:CONTROLLER'], 'idle_timeout': 0,
'table_id': 0, 'cookie': 0, 'length': 80, 'packet_count': 46, 'duration_sec':
4, 'match': {}, 'hard_timeout': 0, 'byte_count': 5874, 'flags': 0,
'duration_nsec': 749000000, 'priority': 0}]}


You can also check the flows with the following commands:

  sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1
  sudo ovs-ofctl --protocols OpenFlow13 dump-flows s2
  sudo ovs-ofctl --protocols OpenFlow13 dump-flows s3
```

9.3 Ryu Topology viewer

Ryu has a rather simple topology viewer. Add it to the apps list to execute it. The RESTful API is a dependency of the viewer so if the topology viewer is running, then the GUI can be accessed the following URL in a browser:

```
sdn@sdn-mn:~$ ryu-manager --ofp-tcp-listen-port 6653 --wsapi-port 8081
--observe-links --app-lists ryu.app.simple_switch_13 ryu.app.ofctl_rest
ryu.app.gui_topology.gui_topology

loading app ryu.app.simple_switch_13
loading app ryu.app.ofctl_rest
loading app ryu.app.gui_topology.gui_topology
loading app ryu.controller.ofp_handler
loading app ryu.app.ws_topology
loading app ryu.app.rest_topology
loading app ryu.app.ofctl_rest
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app None of Switches
creating context switches
instantiating app ryu.app.simple_switch_13 of SimpleSwitch13
instantiating app ryu.app.ofctl_rest of RestStatsApi
instantiating app ryu.app.gui_topology.gui_topology of GUIServerApp
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu.app.ws_topology of WebSocketTopology
instantiating app ryu.app.rest_topology of TopologyAPI
(14947) wsgi starting up on http://0.0.0.0:8081
Packet in 1 d2:7a:44:95:4d:62 33:33:00:00:00:fb 2
packet in 2 d2:7a:44:95:4d:62 33:33:00:00:00:fb 3
packet in 3 4a:bf:61:3b:75:e4 33:33:00:00:00:fb 3
packet in 2 b6:1e:b0:18:4c:f6 33:33:00:00:00:fb 3
packet in 1 0e:12:4e:1a:6a:e4 33:33:00:00:00:fb 1
packet in 3 0e:12:4e:1a:6a:e4 33:33:00:00:00:fb 3
packet in 2 00:00:00:00:00:01 33:33:00:00:00:02 1
packet in 1 00:00:00:00:00:01 33:33:00:00:00:02 1
packet in 3 00:00:00:00:00:01 33:33:00:00:00:02 3
(14947) accepted ('127.0.0.1', 48818)
127.0.0.1 - - [14/May/2020 14:57:08] "GET / HTTP/1.1" 200 515 0.035428
127.0.0.1 - - [14/May/2020 14:57:08] "GET /ryu.topology.css HTTP/1.1" 200 514
0.151302
(14947) accepted ('127.0.0.1', 48820)
127.0.0.1 - - [14/May/2020 14:57:08] "GET /ryu.topology.js HTTP/1.1" 200 8652
0.140234
(14947) accepted ('127.0.0.1', 48830)
127.0.0.1 - - [14/May/2020 14:57:08] "GET /v1.0/topology/switches HTTP/1.1" 200
1088 0.000505
127.0.0.1 - - [14/May/2020 14:57:08] "GET /favicon.ico HTTP/1.1" 404 393
0.000566
127.0.0.1 - - [14/May/2020 14:57:08] "GET /v1.0/topology/links HTTP/1.1" 200
1029 0.000508
127.0.0.1 - - [14/May/2020 14:57:09] "GET /router.svg HTTP/1.1" 200 3647
0.001084
```

http://127.0.0.1:8081

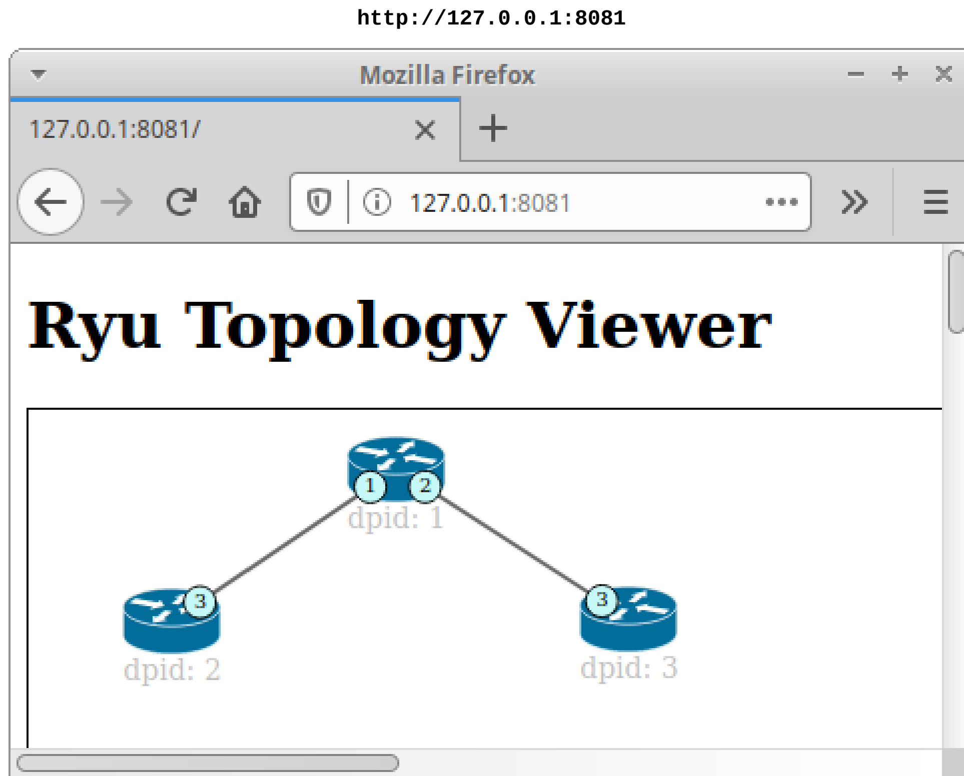


Illustration 23: Ryu topology viewer

9.4 Ryu Flowmanager

The FlowManager is an Apache 2 Licensed Ryu controller application written by Maen Artimy of Dalhousie University in Canada that gives the browser interface manual control over the flow tables in an OpenFlow network. Through the Flowmanager it is possible to create, modify, or delete flows directly from the application. Flowmanager allows for the monitoring of the OpenFlow switches as well as view statistics.

9.4.1 Install Flowmanager

```
sdn@sdn-mn:~$ git clone https://github.com/martimy/flowmanager
Cloning into 'flowmanager'...
remote: Enumerating objects: 48, done.
remote: Counting objects: 100% (48/48), done.
remote: Compressing objects: 100% (35/35), done.
remote: Total 769 (delta 23), reused 28 (delta 13), pack-reused 721
Receiving objects: 100% (769/769), 2.09 MiB | 135.00 KiB/s, done.
Resolving deltas: 100% (498/498), done.
Checking connectivity... done.
sdn@sdn-mn:~$
```

9.4.2 Running Ryu controller with Flowmanager

```
(Window 1) sdn@sdn-mn:~$ ryu-manager --observe-links --app-lists
~/flowmanager/flowmanager.py ryu.app.simple_switch_13
loading app /home/sdn/flowmanager/flowmanager.py
You are using Python v3.8.2.final.0
loading app ryu.app.simple_switch_13
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
creating context wsgi
instantiating app None of DPSet
creating context dpset
instantiating app /home/sdn/flowmanager/flowmanager.py of FlowManager
instantiating app ryu.app.simple_switch_13 of SimpleSwitch13
instantiating app ryu.topology.switches of Switches
instantiating app ryu.controller.ofp_handler of OFPHandler
(15142) wsgi starting up on http://0.0.0.0:8080
```

9.4.3 Run the mininet topology

```
(Window 2) sdn@sdn-mn:~$ sudo mn --topo tree,depth=2,fanout=2 --switch ovsk --
controller=remote,ip=127.0.0.1,port=6653 --mac
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3
*** Adding links:
(s1, s2) (s1, s3) (s2, h1) (s2, h2) (s3, h3) (s3, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet>
```

`http://127.0.0.1:8080/home/`

Flow Manager - Mozilla Firefox

Flow Manager

127.0.0.1:8080/home/

Flow Manager

Home
Flows
Groups
Meters
Flow Control
Group Control
Meter Control
Topology
Messages
Configuration
About

Switch ID(s)

#>1
2
3

Switch Desc

Mfr Desc : Nicira, Inc.
Hw Desc : Open vSwitch
Sw Desc : 2.13.0
Serial Num : None
Dp Desc : s1

Port Desc

SUPPORTED	STATE	PORT NO	PEER	NAME	MAX SPEED	HW ADDR
0	1	LOCAL	0	s1	0	b6:2d:d5:27:06:4c
0	4	1	0	s1-eth1	0	fa:43:e5:3a:37:fc
0	4	2	0	s1-eth2	0	7e:e5:5e:f2:e3:93

Ports stats

TX PACKETS	TX ERRORS	TX DROPPED	TX BYTES	RX PACKETS	RX OVER_ERR	RX FRA
0	0	0	0	0	0	0
260	0	0	18998	231	0	0
257	0	0	18728	234	0	0

Flow Summary

PACKET COUNT	FLOW COUNT	BYTE COUNT
461	2	31290

Table stats

TABLE ID	MATCHED COUNT	LOOKUP COUNT	ACTIVE COUNT
0	461	461	2
1	0	0	0
2	0	0	0
3	0	0	0

Illustration 24: Flowmanager

This page is intentionally blank

10. Custom Topologies

The topologies created thus far have been defined by the *mn* command options and these are limited. It will become necessary to create more customised topologies and this can be achieved using Python scripting. Mininet has example scripts in:

```
sdn@sdn-mn:~$ ls ~/mininet/examples
baresshd.py       controlnet.py       mobility.py       README.md
bind.py           cpu.py              multilink.py      scratchnet.py
clustercli.py     emptynet.py         multiping.py      scratchnetuser.py
clusterdemo.py    hwintf.py           multipoll.py      simpleperf.py
clusterperf.py    __init__.py         multitest.py      sshd.py
cluster.py        intfoptions.py      natnet.py         test
clusterSanity.py  limit.py            nat.py            tree1024.py
consoles.py       linearbandwidth.py  numberedports.py  treeping64.py
controllers2.py   linuxrouter.py      popenpoll.py      vlanhost.py
controllers.py    miniedit.py         popen.py
```

and custom scripts can be created in:

```
sdn@sdn-mn:~$ ls ~/mininet/custom
README  topo-2sw-2host.py

sdn@sdn-mn:~$ cat ~/mininet/custom/README
This directory should hold configuration files for custom mininets.

See custom_example.py, which loads the default minimal topology.  The advantage
of defining a mininet in a separate file is that you then use the --custom
option in mn to run the CLI or specific tests with it.

To start up a mininet with the provided custom topology, do:
  sudo mn --custom custom_example.py --topo mytopo
```

An example is given for a two switch solution with a host attached to each.

```
sdn@sdn-mn:~$ cp ~/example_scripts/topo-2sw-2host.py ~/mininet/custom/
sdn@sdn-mn:~$ cat ~/mininet/custom/topo-2sw-2host.py
```

```
 1 """Custom topology example
 2 
 3 Two directly connected switches plus a host for each switch:
 4 
 5    host --- switch --- switch --- host
 6 
 7 Adding the 'topos' dict with a key/value pair to generate our newly defined
 8 topology enables one to pass in '--topo=mytopo' from the command line.
 9 """
10 
11 from mininet.topo import Topo
12 
13 class MyTopo( Topo ):
14     "Simple topology example."
15 
16     def __init__( self ):
17         "Create custom topo."
18 
19         # Initialize topology
20         Topo.__init__( self )
21 
22         # Add hosts and switches
23         leftHost = self.addHost( 'h1' )
24         rightHost = self.addHost( 'h2' )
25         leftSwitch = self.addSwitch( 's3' )
26         rightSwitch = self.addSwitch( 's4' )
27 
28         # Add links
29         self.addLink( leftHost, leftSwitch )
30         self.addLink( leftSwitch, rightSwitch )
31         self.addLink( rightSwitch, rightHost )
32 
33 
34 topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Illustration 25: Custom topology example

```
sdn@sdn-mn:~$ sudo mn --custom ~/mininet/custom/topo-2sw-2host.py --topo mytopo
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s3 s4
*** Adding links:
(h1, s3) (s3, s4) (s4, h2)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 2 switches
s3 s4 ...
*** Starting CLI:

mininet> net
h1 h1-eth0:s3-eth1
h2 h2-eth0:s4-eth2
s3 lo:  s3-eth1:h1-eth0 s3-eth2:s4-eth1
s4 lo:  s4-eth1:s3-eth2 s4-eth2:h2-eth0
c0
```

10.1 Create a custom topology

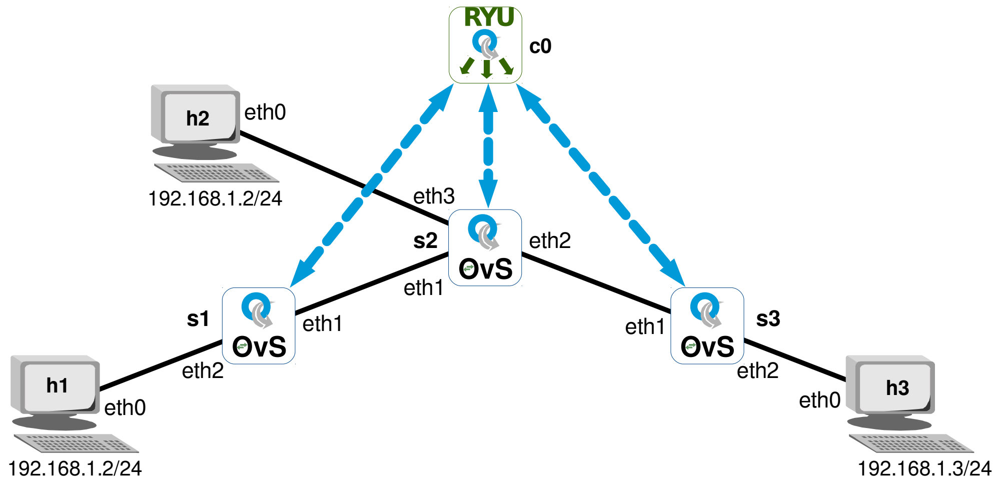


Illustration 26: Mininet custom topology example

Taking the diagram in Illustration 26 as an example to build. Use the existing file as a template to build the required custom topology.

Run up a simple Ryu controller on port 6653.

```
sdn@sdn-mn:~$ ryu-manager --ofp-tcp-listen-port 6653 --app-lists
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
```

Copy the custom-OvS.py program to the mininet custom directory. Make the program executable and review.

```
sdn@sdn-mn:~$ chmod +x ~/mininet/custom/custom-OvS.py
sdn@sdn-mn:~$ cat ~/mininet/custom/custom-OvS.py
```

```
 1 #! /usr/bin/env python2
 2 # -*- coding: utf-8 -*-
 3
 4 from mininet.net import Mininet
 5 from mininet.node import Controller, RemoteController
 6 from mininet.cli import CLI
 7 from mininet.log import setLogLevel, info
 8
 9 """ Custom topology example """
10
11 # Declare variables
12 ryu_ip = '127.0.0.1'
13 ryu_port = 6653
14
15 def customNet():
16
17     "Create a customNet and add devices to it."
18
19     net = Mininet( topo=None, build=False )
20
21     # Add controller
22     info( 'Adding Ryu controller\n' )
23
24     net.addController( 'c0',
25                        controller=RemoteController,
26                        ip=ryu_ip,
27                        port=ryu_port
28                      )
29
30     # Add hosts
31     info( 'Adding hosts\n' )
32     h1, h2, h3 = [ net.addHost(h) for h in ('h1', 'h2', 'h3') ]
33
34     # Add switches
35     info( 'Adding switches\n' )
36     s1, s2, s3 = [ net.addSwitch(s) for s in ('s1', 's2', 's3') ]
37
38     # Add links
39     info( 'Adding switch links\n' )
40     for sa, sb in [ (s1, s2), (s2, s3) ]:
41         net.addLink( sa, sb )
42
43     for h, s in [ (h1, s1), (h2, s2), (h3, s3) ]:
44         net.addLink( h, s )
45
46     info( '*** Starting network ***\n')
47     net.start()
48
49     info( '*** Running CLI ***\n' )
50     CLI( net )
51
52     info( '*** Stopping network ***' )
53     net.stop()
54
55 if __name__ == '__main__':
56     setLogLevel( 'info' )
57     customNet()
58
59 exit(0)
```

Illustration 27: Custom OvS topology

Run the custom-OvS.py program to generate the custom topology.

```
sdn@sdn-mn:~$ sudo ~/mininet/custom/custom-OvS.py
Adding Ryu controller
Adding hosts
Adding switches
Adding switch links
Adding host links
*** Starting network ***
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Running CLI ***
*** Starting CLI:
mininet>
```

Reviewing the new network with the ‘dump’, ‘net’, ‘pingall’, ’iperf’ and ‘dpctl dump-flows’ commands.

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=7078>
<Host h2: h2-eth0:10.0.0.2 pid=7081>
<Host h3: h3-eth0:10.0.0.3 pid=7084>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=7090>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=7093>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None pid=7096>
<RemoteController c0: 127.0.0.1:6653 pid=7072>

mininet> net
h1 h1-eth0:s1-eth2
h2 h2-eth0:s2-eth3
h3 h3-eth0:s3-eth2
s1 lo:  s1-eth1:s2-eth1 s1-eth2:h1-eth0
s2 lo:  s2-eth1:s1-eth1 s2-eth2:s3-eth1 s2-eth3:h2-eth0
s3 lo:  s3-eth1:s2-eth2 s3-eth2:h3-eth0
c0

mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)

mininet> h1 fping h2
10.0.0.2 is alive

mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
.*** Results: ['21.6 Gbits/sec', '21.6 Gbits/sec']

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.595/2.078/2.826/1.048 ms
```

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=25.996s, table=0, n_packets=4, n_bytes=280, idle_age=20,
priority=1,in_port=1,dl_src=8a:52:90:73:a2:bf,dl_dst=ea:07:23:16:c3:48
actions=output:2
 cookie=0x0, duration=25.991s, table=0, n_packets=3, n_bytes=238, idle_age=20,
priority=1,in_port=2,dl_src=ea:07:23:16:c3:48,dl_dst=8a:52:90:73:a2:bf
actions=output:1
 cookie=0x0, duration=25.954s, table=0, n_packets=4, n_bytes=280, idle_age=20,
priority=1,in_port=1,dl_src=f2:53:ac:73:42:d2,dl_dst=ea:07:23:16:c3:48
actions=output:2
 cookie=0x0, duration=25.952s, table=0, n_packets=3, n_bytes=238, idle_age=20,
priority=1,in_port=2,dl_src=ea:07:23:16:c3:48,dl_dst=f2:53:ac:73:42:d2
actions=output:1
 cookie=0x0, duration=476.008s, table=0, n_packets=74, n_bytes=7814,
idle_age=25, priority=0 actions=CONTROLLER:65535
*** s2 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=25.999s, table=0, n_packets=4, n_bytes=280, idle_age=20,
priority=1,in_port=3,dl_src=8a:52:90:73:a2:bf,dl_dst=ea:07:23:16:c3:48
actions=output:1
 cookie=0x0, duration=25.985s, table=0, n_packets=3, n_bytes=238, idle_age=20,
priority=1,in_port=1,dl_src=ea:07:23:16:c3:48,dl_dst=8a:52:90:73:a2:bf
actions=output:3
 cookie=0x0, duration=25.958s, table=0, n_packets=4, n_bytes=280, idle_age=20,
priority=1,in_port=2,dl_src=f2:53:ac:73:42:d2,dl_dst=ea:07:23:16:c3:48
actions=output:1
 cookie=0x0, duration=25.946s, table=0, n_packets=3, n_bytes=238, idle_age=20,
priority=1,in_port=1,dl_src=ea:07:23:16:c3:48,dl_dst=f2:53:ac:73:42:d2
actions=output:2
 cookie=0x0, duration=25.927s, table=0, n_packets=4, n_bytes=280, idle_age=20,
priority=1,in_port=2,dl_src=f2:53:ac:73:42:d2,dl_dst=8a:52:90:73:a2:bf
actions=output:3
 cookie=0x0, duration=25.925s, table=0, n_packets=3, n_bytes=238, idle_age=20,
priority=1,in_port=3,dl_src=8a:52:90:73:a2:bf,dl_dst=f2:53:ac:73:42:d2
actions=output:2
 cookie=0x0, duration=475.993s, table=0, n_packets=75, n_bytes=7876,
idle_age=25, priority=0 actions=CONTROLLER:65535
*** s3 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=25.967s, table=0, n_packets=4, n_bytes=280, idle_age=20,
priority=1,in_port=2,dl_src=f2:53:ac:73:42:d2,dl_dst=ea:07:23:16:c3:48
actions=output:1
 cookie=0x0, duration=25.948s, table=0, n_packets=3, n_bytes=238, idle_age=20,
priority=1,in_port=1,dl_src=ea:07:23:16:c3:48,dl_dst=f2:53:ac:73:42:d2
actions=output:2
 cookie=0x0, duration=25.931s, table=0, n_packets=4, n_bytes=280, idle_age=20,
priority=1,in_port=2,dl_src=f2:53:ac:73:42:d2,dl_dst=8a:52:90:73:a2:bf
actions=output:1
 cookie=0x0, duration=25.919s, table=0, n_packets=3, n_bytes=238, idle_age=20,
priority=1,in_port=1,dl_src=8a:52:90:73:a2:bf,dl_dst=f2:53:ac:73:42:d2
actions=output:2
 cookie=0x0, duration=475.933s, table=0, n_packets=74, n_bytes=7826,
idle_age=25, priority=0 actions=CONTROLLER:65535
```

11. Custom script to Ryu remote controller

Create a clone of the Ryu-Mininet-Testbed VM. In this second VM run the Ryu controller. Make sure that VirtualBox picks a new MAC address for this VM.

```
Settings --->
     Network --->
        Advanced --->
            Click the MAC address refresh icon
```

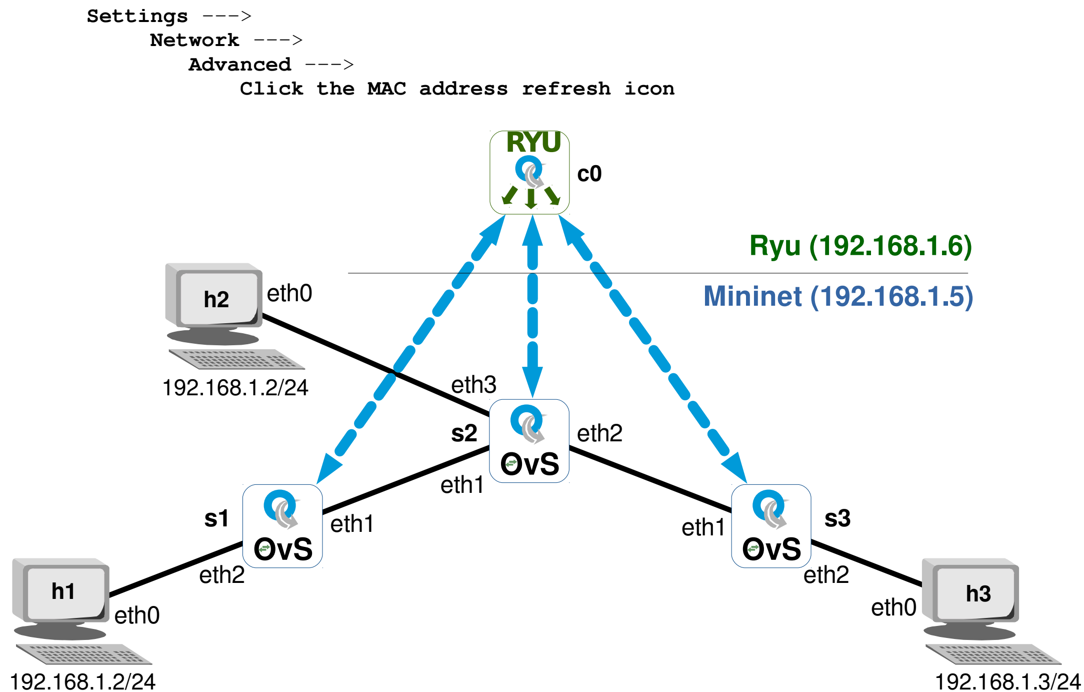


Illustration 28: Test network with Ryu

11.1 Run Ryu

Run the Ryu controller in the second Ryu-Mininet-Testbed VM. Clone the VM and run both at the same time, in onr run ryu and the second run the custom mininet network..

```
sdn@sdn-mn:~$ ip addr show enp0s3 | grep 'inet ' | awk '{print $2}'
192.168.1.6/24

sdn@sdn-mn:~$ ryu-manager --ofp-tcp-listen-port 6653 --app-lists
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
```

11.2 Start Mininet network

Start a Mininet network on the original Ryu-Mininet-Testbed VM. In this case set the 'ryu_ip' variable to the IP address of the second Ryu-Mininet-Testbed VM. The following python script is a variant of the previous script except the controller now points to the remote Ryu SDN controller.

```
sdn@sdn-mn:~$ cp ~/example_scripts/custom-remote-RYU.py ~/mininet/custom/
sdn@sdn-mn:~$ cat <<EOM > ~/mininet/custom/custom-remote-RYU.py
#! /usr/bin/env python
# -*- coding: utf-8 -*-

"""
Custom topology example

Three directly connected switches plus a host attached to each switch
with a remote RYU SDN Controller (c0):

               c0
   RYU        /|\       192.168.1.6
   .........../.|.\.................
   Mininet   /  |  \     192.168.1.5
            /   |   \
    h1 --- s1   |   s3 --- h3
            \   |   /
             \  |  /
              \ | /
               \|/
               s2 --- h2
"""
from mininet.net import Mininet
from mininet.node import Controller, RemoteController
from mininet.cli import CLI
from mininet.log import setLogLevel, info

# Ryu controller
ryu_ip = '192.168.1.6'
ryu_port = 6653

# Define remote RYU Controller

print 'Ryu IP Addr:', ryu_ip
print 'Ryu Port:', ryu_port

def customNet():

    "Create a customNet and add devices to it."

    net = Mininet( topo=None, build=False )


    # Add controller
    info( 'Adding controller\n' )
    net.addController( 'c0',
                       controller=RemoteController,
                       ip = ryu_ip,
                       port = ryu_port
                     )

    # Add hosts
    info( 'Adding hosts\n' )
    h1, h2, h3 = [ net.addHost(h) for h in ('h1', 'h2', 'h3') ]
```

```
    # Add switches
    info( 'Adding switches\n' )
    s1, s2, s3 = [ net.addSwitch(s) for s in ('s1', 's2', 's3') ]

    # Add links
    info( 'Adding switch links\n' )
    for sa, sb in [ (s1, s2), (s2, s3) ]:
        net.addLink( sa, sb )

    for h, s in [ (h1, s1), (h2, s2), (h3, s3) ]:
        net.addLink( h, s )

    info( '*** Starting network ***\n')
    net.start()

    info( '*** Running CLI ***\n' )
    CLI( net )

    info( '*** Stopping network ***' )
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    customNet()

exit(0)

EOM
```

```
sdn@sdn-mn:~$ chmod +x ~/mininet/custom/custom-remote-RYU.py
```

Execute the program.

```
sdn@sdn-mn:~$ sudo ~/mininet/custom/custom-remote-RYU.py
Ryu IP Addr: 192.168.1.6
Ryu Port: 6653
Adding controller
Adding hosts
Adding switches
Adding switch links
Adding host links
*** Starting network ***
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Running CLI ***
*** Starting CLI:
mininet>
```

Review the topology.

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=3504>
<Host h2: h2-eth0:10.0.0.2 pid=3507>
<Host h3: h3-eth0:10.0.0.3 pid=3510>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=3516>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=3519>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None pid=3522>
<RemoteController c0: 192.168.1.6:6653 pid=3498>

mininet> net
h1 h1-eth0:s1-eth2
h2 h2-eth0:s2-eth3
h3 h3-eth0:s3-eth2
s1 lo:  s1-eth1:s2-eth1 s1-eth2:h1-eth0
s2 lo:  s2-eth1:s1-eth1 s2-eth2:s3-eth1 s2-eth3:h2-eth0
s3 lo:  s3-eth1:s2-eth2 s3-eth2:h3-eth0
c0
```

Test the topology.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)


mininet> iperf h1 h3
*** Iperf: testing TCP bandwidth between h1 and h3
.*** Results: ['22.4 Gbits/sec', '22.5 Gbits/sec']
```

11.3 Mininet using Object Oriented Programming

It is also possible to create a topology using Object Oriented Programming (OOP) programming paradigm. This is demonstrated in Illustration 29. The ‘class MyTopo (Topo)’ defines the mininet topology.

In the main() function the class MyTopo is instantiated as ‘topo’ and the mininet network is instanted as ‘net’ without a controller.

The next line adds the controller to the ‘net’ instance. The rest of the main() function mirrors the ‘custom-OvS.py’ program.

```
sdn@sdn-mn:~$ cp ~/example_scripts/custom-OvS-OOP.py ~/mininet/custom/
sdn@sdn-mn:~$ sudo ~/mininet/custom/custom-OvS-OOP.py
```

```
1  #! /usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  from mininet.topo import Topo
5  from mininet.net import Mininet
6  from mininet.node import RemoteController
7  from mininet.cli import CLI
8  from mininet.log import setLogLevel, info
9
10 """ Custom topology example to demonstrate VLANs """
11
12 # Declare variables
13 ryu_ip = '127.0.0.1'
14 ryu_port = 6653
15
16 class MyTopo( Topo ):
17     """Create a custom network class"""
18
19     def __init__(self):
20         """ Constructor method """
21
22         # Initialise topology
23         Topo.__init__(self)
24
25         # Add switches
26         s1,s2,s3 = [ self.addSwitch(s) for s in ('s1', 's2', 's3') ]
27
28         # Add hosts
29         h1, h2, h3 = [ self.addHost(h) for h in ('h1', 'h2', 'h3') ]
30
31         # Add links
32         for sa, sb in [ (s1, s2), (s2, s3) ]:
33             self.addLink( sa, sb )
34
35         for h, s in [ (h1, s1), (h2, s2), (h3, s3) ]:
36             self.addLink( h, s )
37
38 # main() function
39 def main():
40     """Test custom network"""
41
42     topo = MyTopo()
43     net = Mininet(topo=topo, controller=None)
44     net.addController('c0', controller=RemoteController,ip=ryu_ip, port=ryu_port)
45
46     info('*** Starting network ***\n')
47     net.start()
48
49     info('*** Running CLI ***\n' )
50     CLI(net)
51
52     info('*** Stopping network ***' )
53     net.stop()
54
55 # Call run_net() function
56 if __name__ == '__main__':
57     setLogLevel('info')
58     main()
59
60 exit(0)
```

Illustration 29: Custom OvS using OOP

This page is intentionally blank

12. Developing Ryu applications

The process of developing applications in Ryu can be lead by reviewing the example applications given with the application. The simple switch for Openflow v1.3.0 will be the initial guide for the purpose of this testbed document.

Access the code as follows:

```
sdn@sdn-mn:~$ cp ~/.local/lib/python3.8/site-packages/ryu
~/L2_simple_switch_13.py
```

12.1 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
```

The first section includes all of the imports of base classes and libraries required by the application.

12.1.1 app_manager

This is the main entry point for the application. It is the central management of Ryu applications and includes the following classes:

- **RyuApp**: the base class for Ryu applications.
- **AppManager**: the management class that includes methods, such as, load_app, create_contexts and brick management methods for the instantiation, reporting and tearing down of bricks. Bricks are individual elements handled by the app_manager.

12.1.2 OpenFlow event definition

OpenFlow event definitions are handled by the ‘ofp_event’ dispatcher. This captures events when OpenFlow packets are received. It includes the base class of OpenFlow event class to handle OpenFlow events having at least the following attributes.

Attribute	Description
msg	An object which describes the corresponding OpenFlow message.
msg.datapath	A datapath instance description of the OpenFlow switch the packet came from.

timestamp	Timestamp of when this event was generated.

It also handles phase change notification as well as port state change events associated with a datapath instance.

12.1.3 Handler

This incudes a '**set_ev_cls**' decorator method that describes how to handle an event class. Dispatchers argument specifies one of the following negotiation phases for which events should be generated for this handler

Negotiation phase (ryu.controller.handler.)	**Description**
HANDSHAKE_DISPATCHER	Sending and waiting for hello message
CONFIG_DISPATCHER	Version negotiated and sent features-request message
MAIN_DISPATCHER	Switch-features message received and message
DEAD_DISPATCHER	Disconnect from the peer. Or disconnecting due to some unrecoverable errors.

12.1.4 OpenFlow v1.3 definitions

The application imports the OpenFlow v1.3 definitions '**ofproto_v1_3**' which specify which OpenFlow version to be used.

12.1.5 Frame and Packet processing

packet, ethernet, ether_types: packet processing library that includes: packet decoder/encoder class, Ethernet header encoder/decoder class and a lookup list of Ethernet type values like ETH_TYPE_IP = 0x0800, ETH_TYPE_ARP = 0x0806, etc..

12.2 The application class

```
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 = {}
```

The application class is derived from the app_manager RyuApp class. The OpenFlow version is defined and a constructor method that is called when an object is instantiated using the definitions found within the class.

12.3 Event methods

This application has the following event handler methods.

- **switch_features_handler**
- **add_flow**
- **packet_in_handler**

12.3.1 Switch features handler

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
```

The decorator 'set_ev_cls' is used to link the 'switch_features_handler' method with the 'EventOFPSwitchFeatures' event. When such an event is detected then the 'switch_features_handler' method is accessed. This method essentially installs table-miss flow entries in the switches.

For example:

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1

 cookie=0x0, duration=62.808s, table=0, n_packets=66, n_bytes=8320, priority=0
actions=CONTROLLER:65535
```

12.3.2 The add flow method

The role of the 'add_flow' method is to accept requests from other methods to add flows in switches. To do that it expects to be supplied with the:

- Datapath ID
- Priority
- Match details
- Actions
- Buffer_id

For example:

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1

 cookie=0x0, duration=23.971s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01
actions=output:"s1-eth1"
```

12.3.3 The packet in handler

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
```

The packet in handler method '_packet_in_handler' is associated with a decorator 'set_ev_cls' which calls the method for packet in 'EventOFPPacketIn' events.

This method inspects incoming packets from switches, it updates the MAC table, this is a MAC to port dictionary that pairs the source MAC address with the incoming port number. It can check if the destination MAC address is present in the dictionary and if so it can install a flow to avoid handling the next occurrence of this packet otherwise it instructs the switch to flood to all its remaining ports.

12.4 Running the application

```
sdn@sdn-mn:~$ ryu-manager ./example_scripts/L2_simple_switch_13.py
loading app ./example_scripts/L2_simple_switch_13.py
loading app ryu.controller.ofp_handler
instantiating app ./example_scripts/L2_simple_switch_13.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
```

When the 'ryu-manager' calls the 'ryu.app.simple_switch_13', it instantiates both the controller 'ofp_handler' which comes from the imports as well as the 'SimpleSwitch13' class from the 'l2_simple_switch_13.py' application.

Running a very simple topology of a single switch with two hosts connected for demonstration.

```
sdn@sdn-mn:~$ sudo mn --controller remote,ip=127.0.0.1  --switch
ovsk,protocols=OpenFlow13 --mac --ipbase=10.1.1.0/24 --topo single,2

*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
```

```
mininet> pingall

*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
```

Check the flows added to the OpenFlow switch.

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):

 cookie=0x0, duration=73.590s, table=0, n_packets=4, n_bytes=280,
priority=1,in_port=2,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01
actions=output:1

 cookie=0x0, duration=73.584s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port=1,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02
actions=output:2

 cookie=0x0, duration=82.628s, table=0, n_packets=17, n_bytes=1298, priority=0
actions=CONTROLLER:65535
```

Note the matches for the two flow entries.

- **Flow 1**
  - in_port=2
  - dl_src=00:00:00:00:00:02
  - dl_dst=00:00:00:00:00:01
- **Flow 2**
  - in_port=1
  - dl_src=00:00:00:00:00:01
  - dl_dst=00:00:00:00:00:02

This page is intentionally blank

13. Flow parameters

13.1 Flow priority

The OvS checks for a match of each incoming packet against its tables and only the highest priority flow entry that matches the packet is selected. The counters associated with the selected flow entry are updated and the instruction set included in the selected flow entry is applied to the packet. If there are multiple matching flow entries with the same highest priority, the selected flow entry is explicitly undefined. This case can only arise when the controller doesn't set the OFPFF_CHECK_OVERLAP bit on flow mod messages and adds overlapping entries.

To demonstrate this, make some minor changes to the L2 simple switch. These changes will drop all ICMP packets. First include the IPv4 library and the 'in_proto' protocol list in the base classes and libraries.

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

Next add the drop flow for ICMP at the end of the switch features handler method. There is no explicit action to represent DROP, however packets whose action does not have output are dropped. As no 'instructions/action' is specified the default action, DROP is applied.

```
# Drop ICMP flow, priority higher than others making it top flow
match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP,
                        ip_proto=in_proto.IPPROTO_ICMP
                        )
mod = parser.OFPFlowMod(datapath=datapath,
                        table_id=0,
                        priority=10,
                        match=match
                       )
datapath.send_msg(mod)
```

13.1.1 Testing priority

Run the test as per that for the L2 simple switch and review the flow table on the OpenFlow switch.

```
sdn@sdn-mn:~$ ryu-manager ./example_scripts/priority_simple_switch_13.py
loading app ./example_scripts/priority_simple_switch_13.py
loading app ryu.controller.ofp_handler
instantiating app ./example_scripts/priority_simple_switch_13.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
```

Pingall to generate ICMP traffic.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X
h2 -> X
*** Results: 100% dropped (0/2 received)
```

Note the top entry in the OpenFlow switch flow table. It has an action to drop ICMP traffic with a priority of 10.

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1

OFPST_FLOW reply (OF1.3) (xid=0x2):

 cookie=0x0, duration=69.342s, table=0, n_packets=2, n_bytes=196,
priority=10,icmp actions=drop

 cookie=0x0, duration=66.221s, table=0, n_packets=2, n_bytes=84,
priority=1,in_port=2,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01
actions=output:1

 cookie=0x0, duration=51.209s, table=0, n_packets=1, n_bytes=42,
priority=1,in_port=1,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02
actions=output:2

 cookie=0x0, duration=69.342s, table=0, n_packets=17, n_bytes=1242, priority=0
actions=CONTROLLER:65535
```

Carry out the test again but this time use TCP and UDP traffic. This time the traffic is allowed pass.

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
.*** Results: ['25.2 Gbits/sec', '25.2 Gbits/sec']

mininet> iperfudp 10m h1 h2
*** Iperf: testing UDP bandwidth between h1 and h2
*** Results: ['10m', '10.0 Mbits/sec', '10.0 Mbits/sec']
```

Relook at the OpenFlow switch flow table. Compare the priorities. The TCP and UDP traffic can pass because the highest priority flow (priority=10), only matches ICMP traffic, so the next priority (priority=1) rules apply.

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):

 cookie=0x0, duration=261.244s, table=0, n_packets=2, n_bytes=196,
priority=10,icmp actions=drop

 cookie=0x0, duration=258.123s, table=0, n_packets=63517, n_bytes=4192078,
priority=1,in_port=2,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01
actions=output:1

 cookie=0x0, duration=243.111s, table=0, n_packets=276697, n_bytes=15785699338,
priority=1,in_port=1,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02
actions=output:2
 cookie=0x0, duration=261.244s, table=0, n_packets=17, n_bytes=1242, priority=0
actions=CONTROLLER:65535
```

13.2 Timeouts

There are two timeout values associated with an OpenFlow flow.

- **Hard timeout**: A non-zero value causes the flow entry to be removed after the given number of seconds, regardless of how many packets it has matched.
- **Idle timeout**: A non-zero value field causes the flow entry to be removed when it has matched no packets in the given number of seconds.

Consider the flows inserted in the OvS after the flows were added in the l2 simple switch.

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1

 cookie=0x0, duration=23.971s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01
actions=output:"s1-eth1"
```

There are no timeout values specified. When this is the case the timeout values are consider to be the default, ‘0’, which means no timeout and the flows remain in the OpenFlow switch infinitum, or at least until the switch is reset.

13.2.1 The add flow method

Consider the 'add_flow' method in the L2 simple switch below. There is no mention of timeout values so the flow gets the default value of 0.

```
def add_flow(self, datapath, priority, match, actions, buffer_id=None):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                         actions)]
    if buffer_id:
        mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                priority=priority, match=match,
                                instructions=inst)
    else:
        mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                match=match, instructions=inst)
    datapath.send_msg(mod)
```

With a minor adjustment to the mod (modification) variable to it is possible to set timeouts on each flow modification.

```
def add_flow(self, datapath, priority, match, actions, buffer_id=None):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                          actions)]
    if buffer_id:
        mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                idle_timeout=20, hard_timeout=50,
                                priority=priority, match=match,
                                instructions=inst)
    else:
        mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                idle_timeout=20, hard_timeout=50,
                                match=match, instructions=inst)
    datapath.send_msg(mod)
```

13.2.2 Testing the timeouts

Run the test as per that for the L2 simple switch and review the flow table on the OpenFlow switch.

```
sdn@sdn-mn:~$ ryu-manager ./example_scripts/timeout_simple_switch_13.py
loading app ./example_scripts/timeout_simple_switch_13.py
loading app ryu.controller.ofp_handler
instantiating app ./example_scripts/timeout_simple_switch_13.py of
SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
```

Note the inclusion of the timeouts.

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1

OFPST_FLOW reply (OF1.3) (xid=0x2):

 cookie=0x0, duration=5.792s, table=0, n_packets=8, n_bytes=672,
idle_timeout=10, hard_timeout=30, priority=1, in_port=2,
dl_src=00:00:00:00:00:02, dl_dst=00:00:00:00:00:01 actions=output:1

 cookie=0x0, duration=5.782s, table=0, n_packets=7, n_bytes=630,
idle_timeout=10, hard_timeout=30, priority=1, in_port=1,
dl_src=00:00:00:00:00:01, dl_dst=00:00:00:00:00:02 actions=output:2

 cookie=0x0, duration=9.812s, table=0, n_packets=16, n_bytes=1228,
idle_timeout=10, hard_timeout=30, priority=0 actions=CONTROLLER:65535
```

Carry out the test again after a minute and notice that the flows have been removed.

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
```

This page is intentionally blank

14. OpenFlow pipeline processing

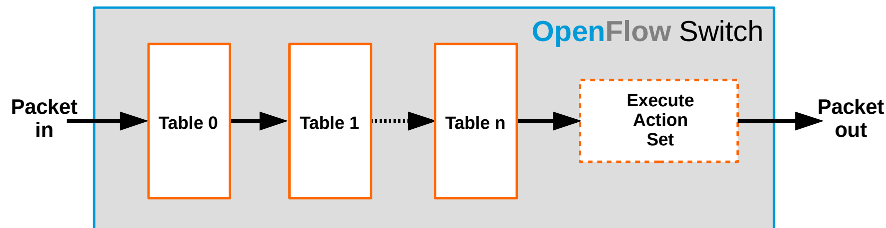


Illustration 30: OpenFlow pipeline processing

To this stage only one flow table, table: 0 has been considered, however it is possible to use multiple tables in a flow pipeline. The OpenFlow pipeline in a switch contains multiple flow tables, each flow table containing multiple flow entries. The OpenFlow pipeline processing defines how packets interact with those flow tables as demonstrated in Illustration 30. An OpenFlow switch is required to have at least one flow table, and can optionally have more flow tables. An OpenFlow switch with only a single flow table is valid and is a case of a simplified pipeline process.

14.1 Example pipeline process

The diagram in Illustration 31 demonstrates that ingress packets arriving at the switch are always checked against table=0 so for the pipeline description this will be called the DEFAULT table. In this example all packets are then forwarded to be matched by the flow rules in table=1, the FILTER table in this case. Table 1, the FILTER table has a rule to drop ICMP packets and forward all other packets to table=2, the FORWARD table where they are processed with a L2 forwarding rule.

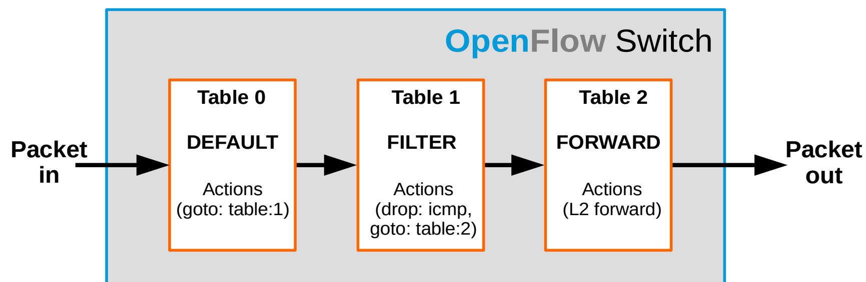


Illustration 31: Example pipeline process

Starting with a basic switch file.

```
sdn@sdn-mn:~$ cp ~/.local/lib/python3.8/site-packages/ryu
~/pp_simple_switch_13.py
```

First include the IPv4 library and the ‘in_proto’ protocol list in the base classes and libraries and add variables to specify a filter table ID and a forwarding table ID.

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

# default table is = 0
filter_table_id = 1
forward_table_id = 2
```

In the switch features method handler add default tables/rules applied during startup.

```
# adding default tables/rules in the startup
self.add_default_table(datapath)
self.add_filter_table_id(datapath)
self.apply_filter_table_id_rules(datapath)
```

In the ‘add_flow’ method add ‘table_id=forward_table_id’ into the flow modification lists. This creates the FORWARD table=2 and adds rules to it dynamically.

```
def add_flow(self, datapath, priority, match, actions, buffer_id=None):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                         actions)]
    if buffer_id:
        mod = parser.OFPFlowMod(datapath=datapath,
                                buffer_id=buffer_id,
                                priority=priority,
                                table_id=forward_table_id,
                                match=match,
                                instructions=inst
                               )
    else:
        mod = parser.OFPFlowMod(datapath=datapath,
                                priority=priority,
                                match=match,
                                table_id=forward_table_id,
                                instructions=inst
                               )
    datapath.send_msg(mod)
```

Add the following three methods after the 'add_flow' method.

The first table, table=0 remains the default table which matches all. The action from that table is to pass on flows to the filter table table=1. This is achieved by the line:

```
  inst = [parser.OFPInstructionGotoTable(filter_table_id)]
```

As no match is specified the flow rule applied to all traffic.

```
def add_default_table(self, datapath):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    inst = [parser.OFPInstructionGotoTable(filter_table_id)]
    mod = parser.OFPFlowMod(datapath=datapath,
                            table_id=0,
                            instructions=inst
                           )
    datapath.send_msg(mod)
```

The next two methods can be considered together, the first creates table=1, the FILTER table and applies the first flow modification rule which again has no match so applies to all traffic to forward to the FORWARD table=2. This rule is given a low priority=1.

The second method adds an additional flow modification to the FILTER table=1. This flow modification matches all ICMP traffic and as no action is specified it is a drop rule. As the priority of this rule is higher than the general forward rule it applies to ICMP traffic first and as such, ICMP traffic is dropped.

```
def add_filter_table_id(self, datapath):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    inst = [parser.OFPInstructionGotoTable(forward_table_id)]
    mod = parser.OFPFlowMod(datapath=datapath,
                            table_id=filter_table_id,
                            priority=1,
                            instructions=inst
                           )
    datapath.send_msg(mod)

def apply_filter_table_id_rules(self, datapath):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP,
                            ip_proto=in_proto.IPPROTO_ICMP
                           )
    mod = parser.OFPFlowMod(datapath=datapath,
                            table_id=filter_table_id,
                            priority=10,
                            match=match
                           )
        datapath.send_msg(mod)
```

14.1.1 Testing the multi-table pipeline process

Testing the multi-table pipeline process is similar to the priority testing earlier as both drop ICMP traffic.

```
sdn@sdn-mn:~$ ryu-manager ./example_scripts/pp_simple_switch_13.py
loading app ./example_scripts/pp_simple_switch_13.py
loading app ryu.controller.ofp_handler
instantiating app ./example_scripts/pp_simple_switch_13.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
```

Run the test as per that for the L2 simple switch and review the flow table on the OpenFlow switch. Pingall to generate ICMP traffic.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X
h2 -> X
*** Results: 100% dropped (0/2 received)
```

Note the top entry in the OpenFlow switch flow table. It has an action to drop ICMP traffic with a priority of 10.

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1

OFPST_FLOW reply (OF1.3) (xid=0x2):

 cookie=0x0, duration=29.859s, table=0, n_packets=20, n_bytes=1480,
actions=goto_table:1

 cookie=0x0, duration=29.859s, table=1, n_packets=2, n_bytes=196,
priority=10,icmp actions=drop

 cookie=0x0, duration=29.859s, table=1, n_packets=18, n_bytes=1284, priority=1
actions=goto_table:2

 cookie=0x0, duration=26.266s, table=2, n_packets=2, n_bytes=84,
priority=1,in_port=2,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01
actions=output:1

 cookie=0x0, duration=11.252s, table=2, n_packets=1, n_bytes=42,
priority=1,in_port=1,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02
actions=output:2

 cookie=0x0, duration=29.859s, table=2, n_packets=17, n_bytes=1242, priority=0
actions=CONTROLLER:65535
```

Carry out the test again but this time use TCP and UDP traffic. This traffic is now allowed pass.

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['27.9 Gbits/sec', '28.0 Gbits/sec']

mininet> iperfudp 10m h1 h2
*** Iperf: testing UDP bandwidth between h1 and h2
*** Results: ['10m', '10.0 Mbits/sec', '10.0 Mbits/sec']
```

In this case re-looking at the table flows will show no change as there have been no new conditions that the switch has had to handle.

14.2 Group tables

Illustration 32: OpenFlow Group table

Group tables consist of group entries. A flow entry can point to a group as well as tables and this enables OpenFlow to represent additional methods of forwarding like ‘select’ and ‘all’.

Each group entry as shown in Illustration 32 contains:

- **Group identifier**: a 32 bit integer to uniquely identify the group.
- **Group type**: determines the group semantics.
- **Counters**: updated when packets are processed by a group.
- **Action buckets**: an ordered list of action buckets that contain a set of actions to be executed and their associated parameters.

A group can include many buckets, and in turn, a bucket can have a set of actions (set, pop, or output).

Sample use cases include the ability to copy a packet to ALL buckets and process it. This has obvious application for traffic mirroring for monitoring or a load balancer function can be achieved by forwarding packets to 1 bucket from many buckets and processing it.

14.2.1 Ryu as a packet sniffer

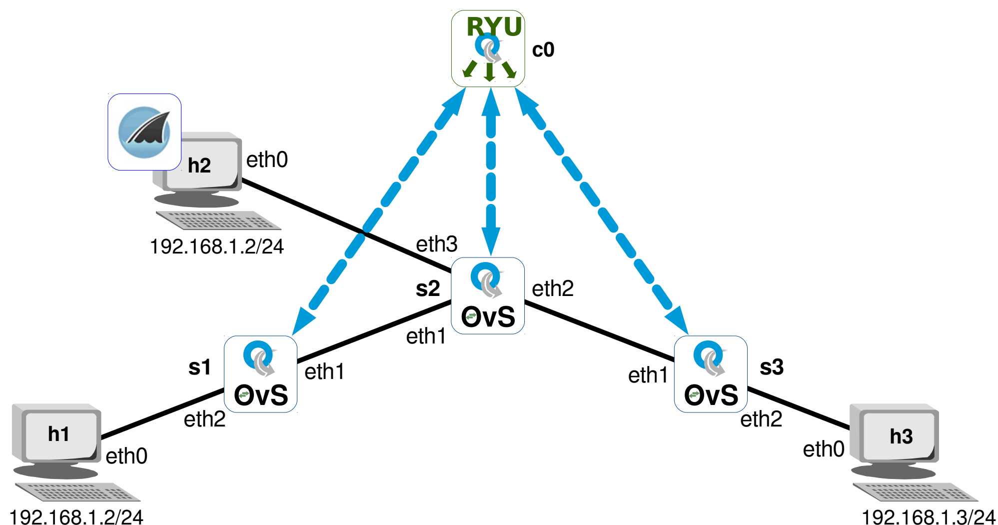


Illustration 33: Ryu as a sniffer using the Group table

Consider the example in Illustration 33. From a mininet perspective it is the same network from earlier created by the custom topology ‘custom-OvS.py’. What is different is the fact that Ryu will create group tables and flows such that traffic arriving at switch port s2:eth1 will be directed to both switch port s2:eth2 and switch port s2:eth3. Traffic arriving at switch port s2:eth2 will be directed to switch port s2:eth1 and switch port s2:eth3. In this way host h2 can operated as a sniffer as it receives a copy of all packets.

Start with a basic switch file.

```
sdn@sdn-mn:~$ cp ~/.local/lib/python3.8/site-packages/ryu
~/gpt_simple_switch_13.py
```

14.2.2 Creating the Ryu sniffer

In the ‘switch_features_handler’ method add a loop that creates flows for switch s2, in particular for the non-sniffer ports s2:eth1 and s2:eth2. All traffic arriving at switch port s2:eth1 is directed to group_id=10 while all traffic arriving at switch port s2:eth2 is directed to group_id=20.

```
# Switch s2
if datapath.id == 2:
    # add group tables

    # Entry for port 1
    self.send_group_mod(datapath)
    actions = [parser.OFPActionGroup(group_id=10)]
    match = parser.OFPMatch(in_port=1)
    self.add_flow(datapath, 10, match, actions)

    # Entry for port 2
    actions = [parser.OFPActionGroup(group_id=20)]
    match = parser.OFPMatch(in_port=2)
    self.add_flow(datapath, 10, match, actions)
```

Add a new method to send group modifications to the switch s2 ‘send_group_mod’. For group table 10 there are two buckets generated, one to direct traffic to s2:eth2 and the other to direct traffic to s2:eth3. The second group table 20 also has two buckets generated, one to direct traffic to s2:eth1 and the other to direct traffic to s2:eth3.

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

        # Matching the custom-OvS topology diagram.

        # Group table 10
        # Receiver port1, forward to port2 and Port3

        actions1 = [parser.OFPActionOutput(2)]
        actions2 = [parser.OFPActionOutput(3)]
        buckets = [parser.OFPBucket(actions=actions1),
                   parser.OFPBucket(actions=actions2)]
        req = parser.OFPGroupMod(datapath,
                                 ofproto.OFPGC_ADD,
                                 ofproto.OFPGT_ALL,
                                 10,
                                 buckets
                                )
        datapath.send_msg(req)
```

```
# Group table 20
# Receive Port2, forward to port2 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,
                         20,
                         buckets
                         )
datapath.send_msg(req)
```

14.2.3 Testing the Ryu sniffer

To see it operate run the adjusted Ryu controller.

```
sdn@sdn-mn:~$ ryu-manager ./example_scripts/gpt_simple_switch_13.py
loading app ./example_scripts/gpt_simple_switch_13.py
loading app ryu.controller.ofp_handler
instantiating app ./example_scripts/gpt_simple_switch_13.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
```

Run the custom topology.

```
sdn@sdn-mn:~$ cp ~/example_scripts/custom-OvS.py  ~/mininet/custom/
sdn@sdn-mn:~$ sudo ~/mininet/custom/custom-OvS.py
Adding Ryu controller
Adding hosts
Adding switches
Adding switch links
Adding host links
*** Starting network ***
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Running CLI ***
*** Starting CLI:
mininet>
```

Run up three xterms

```
mininet> xterm h1

mininet> xterm h2

mininet> xterm h3
```

```
"Node: h1"
root@sdn-mn:~# ping -c1 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.532 ms

--- 10.0.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.532/0.532/0.532/0.000 ms
root@sdn-mn:~#
```

```
"Node: h3"
root@sdn-mn:~# ping -c1 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.113 ms

--- 10.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.113/0.113/0.113/0.000 ms
root@sdn-mn:~#
```

```
"Node: h2"
root@sdn-mn:~# tcpdump -a
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
17:00:18.218977 IP 10.0.0.1 > 10.0.0.3: ICMP echo request, id 17700, seq 1, length 64
17:00:18.219167 IP 10.0.0.3 > 10.0.0.1: ICMP echo reply, id 17700, seq 1, length 64
17:00:19.156297 IP 10.0.0.3 > 10.0.0.1: ICMP echo request, id 17701, seq 1, length 64
17:00:19.156327 IP 10.0.0.1 > 10.0.0.3: ICMP echo reply, id 17701, seq 1, length 64
^C
4 packets captured
4 packets received by filter
0 packets dropped by kernel
root@sdn-mn:~#
```

Illustration 34: Testing Ryu as a packet sniffer

Run the ‘tcpdump’ application on node h2. A ping from host h1 to host h3 and vice versa demonstrates this as both sets of traffic can be seen arrive at host h2 via the trace.

So what happened in the three switches? Looking at the flow tables in switches s1 and s3 they are standard enough with matches on in ports plus L2 MAC addresses and output via ports as expected. There are no group tables created in either of these switches.

```
sdn@sdn-mn:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1

OFPST_FLOW reply (OF1.3) (xid=0x2):

 cookie=0x0, duration=14.068s, table=0, n_packets=6, n_bytes=476, priority=1,
in_port=1, dl_src=22:28:37:d7:54:15, dl_dst=ea:d7:3c:12:e4:63 actions=output:2

 cookie=0x0, duration=14.066s, table=0, n_packets=5, n_bytes=434, priority=1,
in_port=2, dl_src=ea:d7:3c:12:e4:63, dl_dst=22:28:37:d7:54:15 actions=output:1

 cookie=0x0, duration=21.830s, table=0, n_packets=61, n_bytes=7616, priority=0
actions=CONTROLLER:65535

sdn@sdn-mn:~$ sudo ovs-ofctl -O OpenFlow13 dump-groups s1
OFPST_GROUP_DESC reply (OF1.3) (xid=0x2):
```

```
sdn@sdn-mn:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s3

OFPST_FLOW reply (OF1.3) (xid=0x2):

 cookie=0x0, duration=18.358s, table=0, n_packets=6, n_bytes=476,
priority=1,in_port=2, dl_src=22:28:37:d7:54:15, dl_dst=ea:d7:3c:12:e4:63
actions=output:1

 cookie=0x0, duration=18.344s, table=0, n_packets=5, n_bytes=434,
priority=1,in_port=1, dl_src=ea:d7:3c:12:e4:63, dl_dst=22:28:37:d7:54:15
actions=output:2

 cookie=0x0, duration=26.013s, table=0, n_packets=62, n_bytes=7706, priority=0
actions=CONTROLLER:65535

sdn@sdn-mn:~$ sudo ovs-ofctl -O OpenFlow13 dump-groups s3
OFPST_GROUP_DESC reply (OF1.3) (xid=0x2):
```

Considering switch s2, traffic arriving on switch port s2:eth1 is passed to group table 10 and traffic arriving on switch port s2:eth2 is passed to group table 20.

```
sdn@sdn-mn:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s2
OFPST_FLOW reply (OF1.3) (xid=0x2):

 cookie=0x0, duration=24.410s, table=0, n_packets=33, n_bytes=4043,
priority=10,in_port=1 actions=group:10

 cookie=0x0, duration=24.410s, table=0, n_packets=31, n_bytes=3875,
priority=10,in_port=2 actions=group:20

 cookie=0x0, duration=24.410s, table=0, n_packets=7, n_bytes=558, priority=0
actions=CONTROLLER:65535
```

Group table 20 directs the traffic it receives to two buckets corresponding to switch ports s2:eth1 and s2:eth3 while group table 10 directs the traffic it receives to switch ports s2:eth2 and s2:eth3. In this way all traffic is passed via switch ports s2:eth3 to host h2 for sniffing.

```
sdn@sdn-mn:~$ sudo ovs-ofctl -O OpenFlow13 dump-groups s2
OFPST_GROUP_DESC reply (OF1.3) (xid=0x2):
 group_id=20,type=all, bucket=actions=output:1, bucket=actions=output:3
 group_id=10,type=all, bucket=actions=output:2, bucket=actions=output:3
```

14.3 Proxy Address Resolution Protocol

Proxy ARP is a technique by which a device on a given network answers the ARP queries for an IP address that is not on that network. The ARP proxy is aware of the location of the traffic's destination, and offers its own MAC address as the destination.

Consider the network in Illustration 35, now run the l2 simple switch, monitor traffic on all interfaces and specifically filter for ARP messages. Ping from host h1 to host h4 across the network.

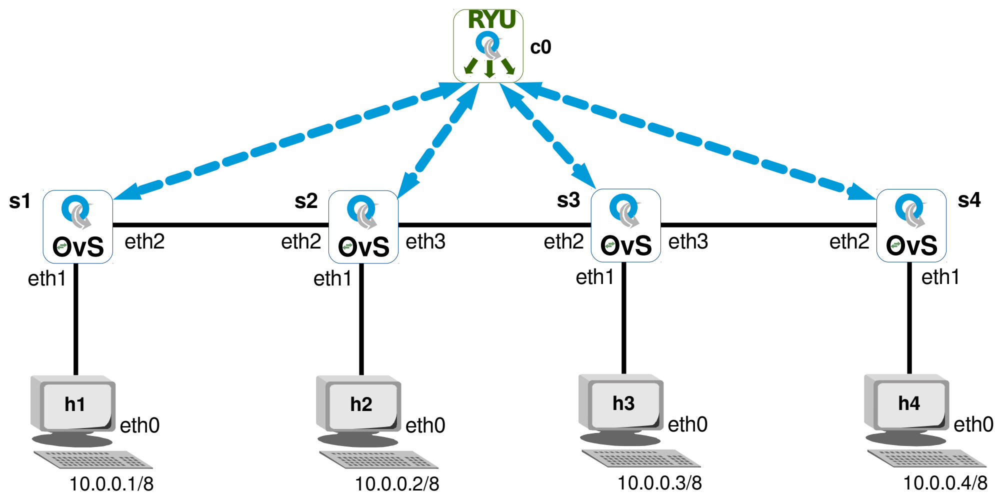


Illustration 35: Proxy ARP network

```
sdn@sdn-mn:~$ ryu-manager ./example_scripts/pa_simple_switch_13.py
loading app ./example_scripts/pa_simple_switch_13.py
loading app ryu.controller.ofp_handler
instantiating app ./example_scripts/pa_simple_switch_13.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
```

```
sdn@sdn-mn:~$ sudo mn --controller=remote,ip=127.0.0.1 --mac
--switch=ovsk,protocols=OpenFlow13 --topo=linear,4
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (s2, s1) (s3, s2) (s4, s3)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
mininet> h1 ping h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=34.9 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.555 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.045 ms
```

Now consider the output of the monitor.

No.	Source	Destination	Protocol	Length	Info
1	00:00:00:00:00:01	00:00:00:00:00:00	ARP	44	Who has 10.0.0.4? Tell 10.0.0.1
2	127.0.0.1	127.0.0.1	OpenFlow	152	Type: OFPT_PACKET_IN
3	00:00:00:00:00:01	00:00:00:00:00:00	ARP	44	Who has 10.0.0.4? Tell 10.0.0.1
4	00:00:00:00:00:01	00:00:00:00:00:00	ARP	44	Who has 10.0.0.4? Tell 10.0.0.1
5	127.0.0.1	127.0.0.1	OpenFlow	152	Type: OFPT_PACKET_IN
6	00:00:00:00:00:01	00:00:00:00:00:00	ARP	44	Who has 10.0.0.4? Tell 10.0.0.1
7	00:00:00:00:00:01	00:00:00:00:00:00	ARP	44	Who has 10.0.0.4? Tell 10.0.0.1
8	00:00:00:00:00:01	00:00:00:00:00:00	ARP	44	Who has 10.0.0.4? Tell 10.0.0.1
9	127.0.0.1	127.0.0.1	OpenFlow	152	Type: OFPT_PACKET_IN

```
10  00:00:00:00:00:01  00:00:00:00:00:00  ARP       44    Who has 10.0.0.4? Tell 10.0.0.1
11  00:00:00:00:00:01  00:00:00:00:00:00  ARP       44    Who has 10.0.0.4? Tell 10.0.0.1
12  00:00:00:00:00:01  00:00:00:00:00:00  ARP       44    Who has 10.0.0.4? Tell 10.0.0.1
13  127.0.0.1          127.0.0.1          OpenFlow  152   Type: OFPT_PACKET_IN
14  00:00:00:00:00:01  00:00:00:00:00:00  ARP       44    Who has 10.0.0.4? Tell 10.0.0.1
15  00:00:00:00:00:04  00:00:00:00:00:00  ARP       44    10.0.0.4 is at 00:00:00:00:00:04
16  127.0.0.1          127.0.0.1          OpenFlow  152   Type: OFPT_PACKET_IN
17  00:00:00:00:00:04  00:00:00:00:00:00  ARP       44    10.0.0.4 is at 00:00:00:00:00:04
18  00:00:00:00:00:04  00:00:00:00:00:00  ARP       44    10.0.0.4 is at 00:00:00:00:00:04
19  127.0.0.1          127.0.0.1          OpenFlow  152   Type: OFPT_PACKET_IN
20  00:00:00:00:00:04  00:00:00:00:00:00  ARP       44    10.0.0.4 is at 00:00:00:00:00:04
21  00:00:00:00:00:04  00:00:00:00:00:00  ARP       44    10.0.0.4 is at 00:00:00:00:00:04
22  127.0.0.1          127.0.0.1          OpenFlow  152   Type: OFPT_PACKET_IN
23  00:00:00:00:00:04  00:00:00:00:00:00  ARP       44    10.0.0.4 is at 00:00:00:00:00:04
24  00:00:00:00:00:04  00:00:00:00:00:00  ARP       44    10.0.0.4 is at 00:00:00:00:00:04
25  127.0.0.1          127.0.0.1          OpenFlow  152   Type: OFPT_PACKET_IN
26  00:00:00:00:00:04  00:00:00:00:00:00  ARP       44    10.0.0.4 is at 00:00:00:00:00:04
27  00:00:00:00:00:04  00:00:00:00:00:00  ARP       44    Who has 10.0.0.1? Tell 10.0.0.4
28  00:00:00:00:00:04  00:00:00:00:00:00  ARP       44    Who has 10.0.0.1? Tell 10.0.0.4
29  00:00:00:00:00:04  00:00:00:00:00:00  ARP       44    Who has 10.0.0.1? Tell 10.0.0.4
30  00:00:00:00:00:04  00:00:00:00:00:00  ARP       44    Who has 10.0.0.1? Tell 10.0.0.4
31  00:00:00:00:00:04  00:00:00:00:00:00  ARP       44    Who has 10.0.0.1? Tell 10.0.0.4
32  00:00:00:00:00:04  00:00:00:00:00:00  ARP       44    Who has 10.0.0.1? Tell 10.0.0.4
33  00:00:00:00:00:04  00:00:00:00:00:00  ARP       44    Who has 10.0.0.1? Tell 10.0.0.4
34  00:00:00:00:00:04  00:00:00:00:00:00  ARP       44    Who has 10.0.0.1? Tell 10.0.0.4
35  00:00:00:00:00:01  00:00:00:00:00:00  ARP       44    10.0.0.1 is at 00:00:00:00:00:01
36  00:00:00:00:00:01  00:00:00:00:00:00  ARP       44    10.0.0.1 is at 00:00:00:00:00:01
37  00:00:00:00:00:01  00:00:00:00:00:00  ARP       44    10.0.0.1 is at 00:00:00:00:00:01
38  00:00:00:00:00:01  00:00:00:00:00:00  ARP       44    10.0.0.1 is at 00:00:00:00:00:01
39  00:00:00:00:00:01  00:00:00:00:00:00  ARP       44    10.0.0.1 is at 00:00:00:00:00:01
40  00:00:00:00:00:01  00:00:00:00:00:00  ARP       44    10.0.0.1 is at 00:00:00:00:00:01
41  00:00:00:00:00:01  00:00:00:00:00:00  ARP       44    10.0.0.1 is at 00:00:00:00:00:01
42  00:00:00:00:00:01  00:00:00:00:00:00  ARP       44    10.0.0.1 is at 00:00:00:00:00:01
```

This is significant traffic as ARP packets are forwarded across the network due to the broadcast nature of the traffic. The Ryu controller has visibility of all switches on the network and could be used to reduce this traffic by generating packets directly in response to ARP requests, in other words acting as a proxy ARP application by answering the ARP requests directly and not forward them.

So the Ryu controller on receipt of a packet in for ARP will generate a response directly to the source and the packet is not forwarded.

14.3.1 Creating the Ryu proxy ARP

Start with a basic switch file.

```
sdn@sdn-mn:~$ cp ~/.local/lib/python3.8/site-packages/ryu
~/pa_simple_switch_13.py
```

First it is necessary to import the ARP module.

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

Create a dictionary of IP addresses as keys and MAC addresses as corresponding values.

```
arp_table = {"10.0.0.1": "00:00:00:00:00:01",
             "10.0.0.2": "00:00:00:00:00:02",
             "10.0.0.3": "00:00:00:00:00:03",
             "10.0.0.4": "00:00:00:00:00:04"
            }
```

After the ‘add_flow’ method add an ARP processing method, ‘arp_process’. This is called if ARP traffic is matched and checks the ‘arp_table’ dictionary for a match. If there is a match it is reported to the console and a new packet is generated, first the Ethernet frame, then the ARP layer and the new packet is serialised.
Serialisation means that the packet is encoded and stored as a bytearray. A bytearray is a set of integers between 0 and 256. This is a mutable sequence of integers that can be transmitted to the wire.

A list of actions are created and the newly created ARP response packet is returned to the switch via a packet out response to the original packet in OpenFlow message.

```
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")
```

In the packet handler method, ‘_packet_in_handler’ and after the line ‘self.mac_to_port[dpid][src] = in_port’ add an if loop to detect ARP packets, log such an event to the console and send the packet data to the ‘arp_process’ method for response packet creation and sending. Despite the destination MAC address being a broadcast the Ryu controller responds to the ARP and does not forward the packet.

```
# check for ARP packets
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
```

14.3.2 Testing the Ryu proxy ARP

Run the Ryu controller.

```
sdn@sdn-mn:~$ ryu-manager ./example_scripts/pa_simple_switch_13.py
loading app ./example_scripts/pa_simple_switch_13.py
loading app ryu.controller.ofp_handler
instantiating app ./example_scripts/pa_simple_switch_13.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
```

Run the mininet topology.

```
sdn@sdn-mn:~$ sudo mn --controller=remote,ip=127.0.0.1 --mac
--switch=ovsk,protocols=OpenFlow13 --topo=linear,4
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (s2, s1) (s3, s2) (s4, s3)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
```

Run a test ping from hist h1 to host h4.

```
mininet> h1 ping h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=34.9 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.555 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.045 ms
```

Monitor on the wire, significantly less ARP traffic.

No.	Source	Destination	Protocol	Length	Info
1	00:00:00_00:00:01	00:00:00:00:00:00	ARP	44	Who has 10.0.0.4? Tell 10.0.0.1
2	127.0.0.1	127.0.0.1	OpenFlow	152	Type: OFPT_PACKET_IN
3	127.0.0.1	127.0.0.1	OpenFlow	168	Type: OFPT_PACKET_OUT
4	00:00:00_00:00:04	00:00:00:00:00:00	ARP	62	10.0.0.4 is at 00:00:00:00:00:04
5	00:00:00_00:00:04	00:00:00:00:00:00	ARP	44	Who has 10.0.0.1? Tell 10.0.0.4
6	127.0.0.1	127.0.0.1	OpenFlow	152	Type: OFPT_PACKET_IN
7	127.0.0.1	127.0.0.1	OpenFlow	168	Type: OFPT_PACKET_OUT
8	00:00:00_00:00:01	00:00:00:00:00:00	ARP	62	10.0.0.1 is at 00:00:00:00:00:01

In the Ryu controller terminal there is output like this.

```
packet in 4 00:00:00:00:00:04 00:00:00:00:00:02 1
packet in 3 00:00:00:00:00:04 00:00:00:00:00:02 3
packet in 2 00:00:00:00:00:04 00:00:00:00:00:02 3
packet in 3 82:6b:5a:82:06:49 33:33:00:00:00:fb 3
packet in 3 00:00:00:00:00:03 ff:ff:ff:ff:ff:ff 1
Received ARP Packet 3 00:00:00:00:00:03 ff:ff:ff:ff:ff:ff
Matched MAC 00:00:00:00:00:04
Proxied ARP Response packet
packet in 3 00:00:00:00:00:03 00:00:00:00:00:04 1
```

Review the switch flows. As the response from the Ryu controller is via a Packet Out message there is no flow added to any of the flow tables.

```
sdn@sdn-mn:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):

 cookie=0x0, duration=4.110s, table=0, n_packets=5, n_bytes=490,
priority=1,in_port=1,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:04
actions=output:2

 cookie=0x0, duration=4.076s, table=0, n_packets=5, n_bytes=490,
priority=1,in_port=2,dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01
actions=output:1

 cookie=0x0, duration=9.661s, table=0, n_packets=86, n_bytes=10740, priority=0
actions=CONTROLLER:65535


sdn@sdn-mn:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s2
OFPST_FLOW reply (OF1.3) (xid=0x2):

 cookie=0x0, duration=18.654s, table=0, n_packets=19, n_bytes=1862,
priority=1,in_port=2,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:04
actions=output:3

 cookie=0x0, duration=18.631s, table=0, n_packets=19, n_bytes=1862,
priority=1,in_port=3,dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01
actions=output:2

 cookie=0x0, duration=24.212s, table=0, n_packets=89, n_bytes=11377, priority=0
actions=CONTROLLER:65535


sdn@sdn-mn:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s3
OFPST_FLOW reply (OF1.3) (xid=0x2):

 cookie=0x0, duration=34.067s, table=0, n_packets=34, n_bytes=3332,
priority=1,in_port=2,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:04
actions=output:3

 cookie=0x0, duration=34.048s, table=0, n_packets=34, n_bytes=3332,
priority=1,in_port=3,dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01
actions=output:2

 cookie=0x0, duration=39.629s, table=0, n_packets=92, n_bytes=11986, priority=0
actions=CONTROLLER:65535

sdn@sdn-mn:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s4
OFPST_FLOW reply (OF1.3) (xid=0x2):

 cookie=0x0, duration=47.059s, table=0, n_packets=48, n_bytes=4648,
priority=1,in_port=2,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:04
actions=output:1
```

```
 cookie=0x0, duration=47.047s, table=0, n_packets=48, n_bytes=4648,
priority=1,in_port=1,dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01
actions=output:2

 cookie=0x0, duration=52.627s, table=0, n_packets=93, n_bytes=12028, priority=0
actions=CONTROLLER:65535
```

15. Splitting domains

15.1 Flowspace slicing

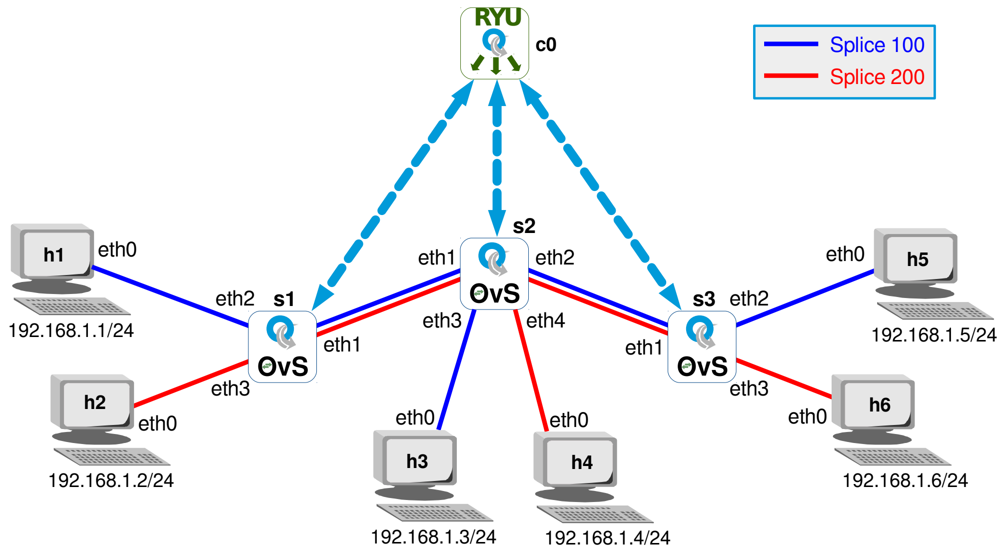


Illustration 36: VLANs

A network slice controls a subset of traffic, in this simple example there are two flow spaces, one for odd MAC addresses and the other for even MAC addresses.

Looking at the mininet custom network for the flowspace slicing code in Illustration 37, it creates three OvS switches with two hosts on each switch s1, s2 and s3. Switches s1 and s3 are connected via switch s2. All hosts are in the same IP subnet 192.168.1.0/24.

To achieve this start with the L2 simple switch as a base.

```
sdn@sdn-mn:~$ cp ~/.local/lib/python3.8/site-packages/ryu
~/splice_simple_switch_13.py
```

Under the import of the base classes and libraries add a list of slice tuples. Each tuple consists of a slice ID followed by the MAC addresses associated with the slice.

```
# Slices 100 and 200 with their respective MAC addresses
slices_data = [(100,"00:00:00:00:00:01","00:00:00:00:00:03","00:00:00:00:00:05"),
               (200,"00:00:00:00:00:02","00:00:00:00:00:04","00:00:00:00:00:06")]
```

Hash out the L2 packet logger as it is two noisy.

```
#self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)
```

Remove the lines in this section that are struck through and replace.

```
if out_port != ofproto.OFPP_FLOOD:

    match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
    # verify if we have a valid buffer_id, if yes avoid to send both
    # flow_mod & packet_out
    if msg.buffer_id != ofproto.OFP_NO_BUFFER:
        self.add_flow(datapath, 1, match, actions, msg.buffer_id)
        return
    else:
        self.add_flow(datapath, 1, match, actions)

        data = None
        if msg.buffer_id == ofproto.OFP_NO_BUFFER:
            data = msg.data
```

```
if out_port != ofproto.OFPP_FLOOD:

    # loop through the slices
    for net_slice in slices_data:
        slice_id = net_slice[0]      # extract the slice ID
        net_slice = net_slice[1:]    # remove the slice ID from the list

        # check if this packets src & dst are in the same slice
        if src in net_slice and dst in net_slice:
            self.logger.info("dpid %s in eth%s out eth%s", dpid, in_port, out_port)
            self.logger.info("Slice pair [%s, %s] in slice %i", src, dst, slice_id)

            # create a match and generate a flow
            match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
            if msg.buffer_id != ofproto.OFP_NO_BUFFER:
                self.add_flow(datapath, 1, match, actions, msg.buffer_id)
                return
            else:
                self.add_flow(datapath, 1, match, actions)

        else: # pair of MAC addresses are not in a slice so skip
            return

    data = None
    if msg.buffer_id == ofproto.OFP_NO_BUFFER:
        data = msg.data
```

15.1.1 Testing the slicing

Start the Ryu controller.

```
sdn@sdn-mn:~$ ryu-manager ./example_scripts/slice_simple_switch_13.py
loading app ./example_scripts/slice_simple_switch_13.py
loading app ryu.controller.ofp_handler
instantiating app ./example_scripts/slice_simple_switch_13.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
```

Start the mininet topology.

```
sdn@sdn-mn:~$ cp ~/example_scripts/custom-slice.py  ~/mininet/custom/
sdn@sdn-mn:~$ sudo ~/mininet/custom/custom-slice.py
*** Creating network
*** Adding hosts:
h1 h2 h3 h4 h5 h6
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (h5, s3) (h6, s3) (s1, s2) (s2, s3)
*** Configuring hosts
h1 h2 h3 h4 h5 h6
*** Starting network ***
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Running CLI ***
*** Starting CLI:
mininet>
```

Carry out a ‘pingall’ on the mininet topology and monitor the Ryu controller terminal.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X h3 X h5 X
h2 -> X X h4 X h6
h3 -> h1 X X h5 X
h4 -> X h2 X X h6
h5 -> h1 X h3 X X
h6 -> X h2 X h4 X
*** Results: 60% dropped (12/30 received)


{{ Ryu terminal output }}

dpid 2 in eth3 out eth1
Slice pair [00:00:00:00:00:03, 00:00:00:00:00:01] in slice 100
dpid 1 in eth1 out eth2
Slice pair [00:00:00:00:00:03, 00:00:00:00:00:01] in slice 100
dpid 1 in eth2 out eth1
Slice pair [00:00:00:00:00:01, 00:00:00:00:00:03] in slice 100

~~~ ~~~ ~~~ ~~~ ~~~

dpid 2 in eth2 out eth4
Slice pair [00:00:00:00:00:06, 00:00:00:00:00:04] in slice 200
dpid 2 in eth4 out eth2
Slice pair [00:00:00:00:00:04, 00:00:00:00:00:06] in slice 200
dpid 3 in eth1 out eth3
Slice pair [00:00:00:00:00:04, 00:00:00:00:00:06] in slice 200
```

```
sdn@sdn-mn:~$ cat ~/mininet/custom/custom-splice.py
```

```
 1 #! /usr/bin/env python2
 2 # -*- coding: utf-8 -*-
 3
 4 from mininet.topo import Topo
 5 from mininet.net import Mininet
 6 from mininet.node import RemoteController
 7 from mininet.cli import CLI
 8 from mininet.log import setLogLevel, info
 9
10 """ Custom topology example to demonstrate slicing """
11
12 # Declare variables
13 ryu_ip = '127.0.0.1'
14 ryu_port = 6653
15
16 class MyTopo( Topo ):
17     """Create a custom network class"""
18
19     def __init__(self):
20         """ Constructor method """
21
22         # Initialise topology
23         Topo.__init__(self)
24
25         # Add switches
26         s1,s2,s3 = [ self.addSwitch(s) for s in ('s1', 's2', 's3') ]
27
28         # Add hosts
29         h1 = self.addHost('h1', ip='192.168.1.1/24', mac='00:00:00:00:00:01')
30         h2 = self.addHost('h2', ip='192.168.1.2/24', mac='00:00:00:00:00:02')
31         h3 = self.addHost('h3', ip='192.168.1.3/24', mac='00:00:00:00:00:03')
32         h4 = self.addHost('h4', ip='192.168.1.4/24', mac='00:00:00:00:00:04')
33         h5 = self.addHost('h5', ip='192.168.1.5/24', mac='00:00:00:00:00:05')
34         h6 = self.addHost('h6', ip='192.168.1.6/24', mac='00:00:00:00:00:06')
35
36         # Add links
37         for sa, sb in [(s1, s2), (s2, s3)]:
38             self.addLink( sa, sb )
39
40         for h, s in [(h1, s1), (h2, s1), (h3, s2), (h4, s2), (h5, s3), (h6, s3)]:
41             self.addLink( h, s )
42
43 # main() function
44 def main():
45     """Test custom network"""
46
47     topo = MyTopo()
48     net = Mininet(topo=topo, controller=None)
49     net.addController('c0', controller=RemoteController,ip=ryu_ip, port=ryu_port)
50
51     info('*** Starting network ***\n')
52     net.start()
53
54     info('*** Running CLI ***\n' )
55     CLI(net)
56
57     info('*** Stopping network ***' )
58     net.stop()
59
60 # Call run_net() function
61 if __name__ == '__main__':
62     setLogLevel('info')
63     main()
64
65 exit(0)
```

Illustration 37: Mininet custom splice topology

Review the OvS switch flow tables.

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1

OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=218.400s, table=0, n_packets=4, n_bytes=280,
priority=1,in_port=1,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01
actions=output:2
 cookie=0x0, duration=218.385s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port=2,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03
actions=output:1
 cookie=0x0, duration=215.368s, table=0, n_packets=5, n_bytes=322,
priority=1,in_port=1,dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:01
actions=output:2
 cookie=0x0, duration=215.363s, table=0, n_packets=4, n_bytes=280,
priority=1,in_port=2,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:05
actions=output:1
 cookie=0x0, duration=199.336s, table=0, n_packets=5, n_bytes=322,
priority=1,in_port=1,dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:02
actions=output:3
 cookie=0x0, duration=199.333s, table=0, n_packets=4, n_bytes=280,
priority=1,in_port=3,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:04
actions=output:1
 cookie=0x0, duration=196.315s, table=0, n_packets=6, n_bytes=364,
priority=1,in_port=1,dl_src=00:00:00:00:00:06,dl_dst=00:00:00:00:00:02
actions=output:3
 cookie=0x0, duration=196.314s, table=0, n_packets=5, n_bytes=322,
priority=1,in_port=3,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:06
actions=output:1
 cookie=0x0, duration=316.976s, table=0, n_packets=137, n_bytes=13208, priority=0
actions=CONTROLLER:65535

sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s2

OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=228.482s, table=0, n_packets=4, n_bytes=280,
priority=1,in_port=3,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01
actions=output:1
 cookie=0x0, duration=228.457s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port=1,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03
actions=output:3
 cookie=0x0, duration=225.450s, table=0, n_packets=5, n_bytes=322,
priority=1,in_port=2,dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:01
actions=output:1
 cookie=0x0, duration=225.440s, table=0, n_packets=4, n_bytes=280,
priority=1,in_port=1,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:05
actions=output:2
 cookie=0x0, duration=209.419s, table=0, n_packets=5, n_bytes=322,
priority=1,in_port=4,dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:02
actions=output:1
 cookie=0x0, duration=209.408s, table=0, n_packets=4, n_bytes=280,
priority=1,in_port=1,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:04
actions=output:4
 cookie=0x0, duration=206.398s, table=0, n_packets=6, n_bytes=364,
priority=1,in_port=2,dl_src=00:00:00:00:00:06,dl_dst=00:00:00:00:00:02
actions=output:1
 cookie=0x0, duration=206.392s, table=0, n_packets=5, n_bytes=322,
priority=1,in_port=1,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:06
actions=output:2
 cookie=0x0, duration=193.377s, table=0, n_packets=4, n_bytes=280,
priority=1,in_port=2,dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:03
actions=output:3
 cookie=0x0, duration=193.373s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port=3,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:05
actions=output:2
 cookie=0x0, duration=167.333s, table=0, n_packets=5, n_bytes=322,
priority=1,in_port=2,dl_src=00:00:00:00:00:06,dl_dst=00:00:00:00:00:04
actions=output:4
```

```
 cookie=0x0, duration=167.332s, table=0, n_packets=4, n_bytes=280,
priority=1,in_port=4,dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:06
actions=output:2
 cookie=0x0, duration=327.054s, table=0, n_packets=154, n_bytes=14110,
priority=0 actions=CONTROLLER:65535

sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s3

OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=228.030s, table=0, n_packets=5, n_bytes=322,
priority=1,in_port=2,dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:01
actions=output:1
 cookie=0x0, duration=228.005s, table=0, n_packets=4, n_bytes=280,
priority=1,in_port=1,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:05
actions=output:2
 cookie=0x0, duration=208.978s, table=0, n_packets=6, n_bytes=364,
priority=1,in_port=3,dl_src=00:00:00:00:00:06,dl_dst=00:00:00:00:00:02
actions=output:1
 cookie=0x0, duration=208.969s, table=0, n_packets=5, n_bytes=322,
priority=1,in_port=1,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:06
actions=output:3
 cookie=0x0, duration=195.958s, table=0, n_packets=4, n_bytes=280,
priority=1,in_port=2,dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:03
actions=output:1
 cookie=0x0, duration=195.939s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port=1,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:05
actions=output:2
 cookie=0x0, duration=169.913s, table=0, n_packets=5, n_bytes=322,
priority=1,in_port=3,dl_src=00:00:00:00:00:06,dl_dst=00:00:00:00:00:04
actions=output:1
 cookie=0x0, duration=169.908s, table=0, n_packets=4, n_bytes=280,
priority=1,in_port=1,dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:06
actions=output:3
 cookie=0x0, duration=329.632s, table=0, n_packets=162, n_bytes=14362,
priority=0 actions=CONTROLLER:65535
```

15.2 Virtual LANs

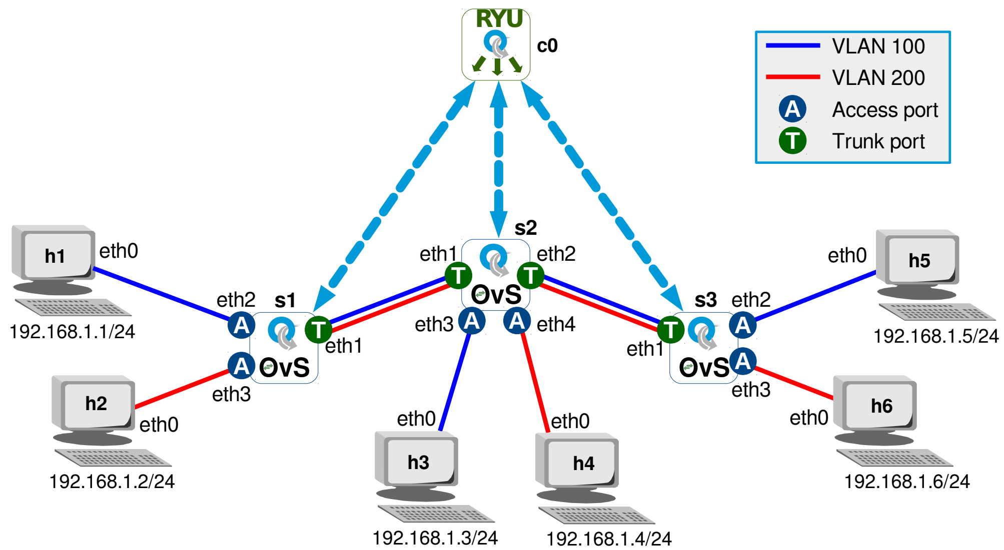


Illustration 38: VLAN topology

The Ryu simple switch is used as a base application. A set of dictionary mapping the ports to the VLANs for each switch and the VLANs to the trunk ports is required. Here the access ports for each switch is defined with their associated VLAN.

Starting with a basic switch file.

```
sdn@sdn-mn:~$ cp ~/.local/lib/python3.8/site-packages/ryu
~/VLAN_simple_switch_13.py
```

15.2.1 Base classes and libraries

Import the following base classes and libraries.

```
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

from ryu.ofproto import ether
from ryu.lib.packet import vlan
from ryu.lib.packet import ipv4
```

15.2.2 Mapping dictionaries

This dictionary is the switch port to VLAN mapping for access ports (tag/untag) that takes the following format. This says that switch 1, port 2 is in VLAN 100 while port 3 is in VLAN 200. In switch 2 port 3 is in VLAN 100 while port 4 is in VLAN 200. The final switch 3 has port 2 in VLAN 100 and port 2 in VLAN 200.

{dpid : {portno: vlanid, ...},....}

```
{
 1:{2:100,
    3:200},
 2:{3:100,
    4:200},
 3:{2:100,
    3:200}
}
```

There is also the switch port to VLAN mapping for trunk ports which must pass all packets whether tagged or not. Each switch port that is a trunk has an associated list of VLANs that can pass through the port. For example in switch 1 the VLANs 100 and 200 can pass through port 1. In switch 2 both port 1 and 2 are trunk and both pass VLAN 100 and 200 while the final switch 3 has port 1 as a trunk port for VLANs 100 and 200.

```
{dpid :  {portno: [vlanid1, vlanid2, vlanidn], ...},....}

{
 1:{1:[100,200]},
 2:{1:[100,200],
    2:[100,200]},
 3:{1:[100,200]}
}
```

Add mapping dictionary for access ports and trunk ports after the imports. In the trunk data dictionary the values consist of lists of VLAN IDs.

```
# Switch port to VLAN mapping for Access ports (tag/untag)
# { dpid :  {portno: vlanid,  ...},....}
access_data = {
               1:{2:100,
                  3:200},
               2:{3:100,
                  4:200},
               3:{2:100,
                  3:200}
              }

# Switch port to VLAN mapping for Trunk ports (pass all)
# { dpid :  {portno: [vlanid1, vlanid2, vlanidn], ...},....}
trunk_data = {
              1:{1:[100,200]},
              2:{1:[100,200],
                 2:[100,200]},
              3:{1:[100,200]}
             }
```

15.2.3 VLAN Utilities class

The 'VlanUtilities' class provide methods that are called from the VLANSwitch13 class. These methods use the 'datapath ID' and the 'VLAN ID':

- get access ports associated with a VLAN
- get trunk ports associated with a VLAN
- determine the access or trunk status of a port

```
# VLAN Utilities class
class VlanUtilities:
    """ Class of utilities used by the program """

    def __init__(self, *args):
        pass

    def get_access_ports(self, dpid, vlan_id):
        ports = []
        for port in access_data[dpid]:
            if access_data[dpid][port] == vlan_id:
                ports.append(port)
        return ports

    def get_trunk_ports(self, dpid, vlan_id):
        ports = []
        for port in trunk_data[dpid]:
            if vlan_id in trunk_data[dpid][port]:
                ports.append(port)
        return ports

    def is_it_access_port(self, dpid, vlan_id, port):
        if port in access_data[dpid]:
            if vlan_id == access_data[dpid][port]:
                return True
        return False

    def is_it_trunk_port(self, dpid, vlan_id, port):
        if port in trunk_data[dpid]:
            if vlan_id in trunk_data[dpid][port]:
                return True
        return False

# End - VlanUtilities class
```

After this class is defined it is necessary to instantiate an instance of the class, here as the object utilities.

```
# Create instance of the VlanUtilities class
utilities = VlanUtilities()
```

15.2.4 Main VLAN switching class add table ID

Change the main class name to a more appropriate name.

```
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 = {}
```

to:

```
class VLANSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(VLANSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
```

At the end of the ‘switch_features_handler()’ method add a zero as shown. This represents the table ID.

```
self.add_flow(datapath, 0, match, actions)
self.add_flow(datapath, 0, 0, match, actions)
```

Add table ID in the add_flow() method as shown in bold.

```
def add_flow(self, datapath, priority, table_id, match, actions, buffer_id=None):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                         actions)]
    if buffer_id:
        mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                priority=priority, table_id=0, match=match,
                                instructions=inst)
    else:
        mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                match=match, table_id=0, instructions=inst)
    datapath.send_msg(mod)
```

15.2.5 Switching in VLAN environment

In the Normal switching application, two types of forwarding are possible.

- **Flood to all ports**
- **Forward to a specific port**

There are a greater set of possibilities in a VLAN setup.

- **Flood to VLAN ports**
  - Access ports
    - Add OFPActionOutput for all VLAN ports
    - No ofproto.OFPP_FLOOD.
  - Trunk ports
    - Set the VLAN TAG
      - OFPActionPushVlan, OFPActionSetField
    - Add OFPActionOutput.
- **Forward to VLAN Port**
  - Both the in port and out port are access ports
    - No tagging required, simply forward.
  - The in port is an access port while the out port is a trunk port
    - Add a VLAN tag and output to trunk port.
  - The in port is a trunk port, while the out port is an access port
    - Remove the VLAN tag and output to the access port.
  - Both the in port and the out port are trunk ports
    - Forward the tagged packets to the out port.

In the VLAN environment flooding is limited within the VLAN domain so the normal switch behaviour of flooding on all ports when there is no match of MAC to port cannot happen. It must be limited within the ports associated with the VLAN of the in port. This new method ‘flood_packet_to_all_vlan_ports()’ handles this behaviour and is added to the ‘VLANSwitch13()’ class.

```
def flood_packet_to_all_vlan_ports(self, msg, vlan_id, tagged):
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']

    actions = []

    # if it is a tagged packet, untag first.
    if tagged == 1:
        actions.append(parser.OFPActionPopVlan())

    aports = utilities.get_vlan_ports(datapath.id, vlan_id)
    tports = utilities.get_trunk_ports(datapath.id, vlan_id)

    (a,b,c,d) = (datapath.id, vlan_id, aports, tports)
    self.logger.info("flood packet: dpid ({}) vlan_id ({}) access ports ({}) \
    trunk ports ({}) ".format(a,b,c,d))
```

```
    # don't sent it back to the received port
    for port in aports:
        if port != in_port:
            actions.append(parser.OFPActionOutput(port))

    # build actions for flow to add VLAN tag to packets egressing trunk ports
    # for trunk port, push a vlan tag onto packet and send it.
    actions.append(parser.OFPActionPushVlan(0x8100))

    # Include OpenFlow Port VLAN ID present (OFPVID_PRESENT)
    vid = vlan_id | ofproto_v1_3.OFPVID_PRESENT
    actions.append(parser.OFPActionSetField(vlan_vid=vid))

    for port in tports:
        if port != in_port:
            actions.append(parser.OFPActionOutput(port))

    # send packet out
    data = None
    if msg.buffer_id == ofproto.OFP_NO_BUFFER:
        data = msg.data

    out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                              in_port=in_port, actions=actions, data=data)
    datapath.send_msg(out)
```

Another method added to the 'VLANSwitch13()' class is the 'forward_packet_to_a_vlan_port()'.

This method handles 4 separate event types.

No	Event	Action
1	in port: access, out port: access	no tagging required, switch
2	in port: access, out port: trunk	push tag at out port, forward to port
3	in port: trunk, out port: access	pop tag at in port, forward to port
4	in port: trunk, out port: trunk	forward to port (tags remain intact)

```
def forward_packet_to_a_vlan_port(self, msg, vlan_id, out_port, dst, src, tagged):
    self.logger.info("forward packet to a port {}".format(out_port))
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']
    actions = []

    # in port: access, out port: access
    if (utilities.is_it_access_port(datapath.id, vlan_id, in_port) and
        utilities.is_it_access_port(datapath.id, vlan_id, out_port)):

        (a,b,c) = (in_port, out_port, vlan_id)
        self.logger.info("Access port {} to access port {} \
        : VLAN {}".format(a,b,c))

        # build match and actions for flow
        actions.append(parser.OFPActionOutput(out_port))
        match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)

        # add flow to flow table
        if msg.buffer_id != ofproto.OFP_NO_BUFFER:
            self.add_flow(datapath, 1, 0, match, actions, msg.buffer_id)
            return
        else:
```

```
            self.add_flow(datapath, 1, 0, match,  actions)

    # in port: access, out port: trunk
    elif (utilities.is_it_access_port(datapath.id, vlan_id, in_port) and
          utilities.is_it_trunk_port(datapath.id, vlan_id, out_port)):

        (a,b,c) = (in_port, out_port, vlan_id)
        self.logger.info("Access port {} to trunk port {} : VLAN \
        {}".format(a,b,c))

        # build match and actions for flow
        # OpenFlow Port VLAN ID (OFPVID) OFPVID_PRESENT must be included
        actions.append(parser.OFPActionPushVlan(0x8100))
        vid = vlan_id | ofproto_v1_3.OFPVID_PRESENT
        actions.append(parser.OFPActionSetField(vlan_vid=vid))
        actions.append(parser.OFPActionOutput(out_port))
        match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)

        # add flow to flow table
        if msg.buffer_id != ofproto.OFP_NO_BUFFER:
            self.add_flow(datapath, 1, 0, match, actions, msg.buffer_id)
            return
        else:
            self.add_flow(datapath, 1, 0, match,  actions)

    # in port: trunk, out port: access
    elif (utilities.is_it_trunk_port(datapath.id, vlan_id, in_port) and
          utilities.is_it_access_port(datapath.id, vlan_id, out_port)):
           (a,b,c) = (in_port, out_port, vlan_id)
        self.logger.info("Trunk port {} to access port {} : VLAN \
        {}".format(a,b,c))

        # build match and actions for flow
        actions.append(parser.OFPActionPopVlan())
        actions.append(parser.OFPActionOutput(out_port))
        vid = vlan_id | ofproto_v1_3.OFPVID_PRESENT
        match = parser.OFPMatch(in_port=in_port, vlan_vid=vid,
                                eth_dst=dst, eth_src=src)

        # add flow to flow table
        if msg.buffer_id != ofproto.OFP_NO_BUFFER:
            self.add_flow(datapath, 1, 0, match, actions, msg.buffer_id)
            return
        else:
            self.add_flow(datapath, 1, 0, match,  actions)

    # in port: trunk, out port: trunk
    elif (is_it_trunk_port(datapath.id, vlan_id, in_port) and
          is_it_trunk_port(datapath.id, vlan_id, out_port)):
        (a,b,c) = (in_port, out_port, vlan_id)
        self.logger.info("Trunk port {} to trunk port {} : VLAN \
        {}".format(a,b,c))

        # build match and actions for flow
        actions.append(parser.OFPActionOutput(out_port))
        vid = vlan_id | ofproto_v1_3.OFPVID_PRESENT
        match = parser.OFPMatch(in_port=in_port, vlan_vid=vid, eth_dst=dst,
                                eth_src=src)

        # add flow to flow table
        if msg.buffer_id != ofproto.OFP_NO_BUFFER:
            self.add_flow(datapath, 1, 0, match, actions, msg.buffer_id)
            return
        else:
            self.add_flow(datapath, 1, 0, match,  actions)
    else:
        (a,b,c) = (in_port, out_port, vlan_id)
        self.logger.info("Unknown event: in port {} to out port {} : VLAN \
        {}".format(a,b,c))
```

```
    # packet out - if no buffer
    data = None
    if msg.buffer_id == ofproto.OFP_NO_BUFFER:
        data = msg.data

    out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                              in_port=in_port, actions=actions, data=data)
    datapath.send_msg(out)
```

There are major changes required to the Packet IN handler '_packet_in_handler():' too. Add a new variable 'dpid' as shown.

```
msg = ev.msg
datapath = msg.datapath
ofproto = datapath.ofproto
parser = datapath.ofproto_parser
in_port = msg.match['in_port']
dpid = datapath.id
```

Also add a set of local variables to hold the VLAN ID, whether the packet is tagged or not and the source and destination MAC address.

```
pkt = packet.Packet(msg.data)
eth = pkt.get_protocols(ethernet.ethernet)[0]

# local variables
vlan_id = 0
tagged = 0
dst = eth.dst
src = eth.src

if eth.ethertype == ether_types.ETH_TYPE_LLDP:
    # ignore lldp packet
    return
dst = eth.dst
src = eth.src

# handles packets with VLAN tags
elif eth.ethertype == ether_types.ETH_TYPE_8021Q:
    vlan_hdr = pkt.get_protocols(vlan.vlan)[0]
    vlan_id = vlan_hdr.vid
    tagged = 1
    (a,b,c,d,e) = (dpid, src, dst, in_port, vlan_id)
    self.logger.info("vlan tagged packet in dp-id({}) src-mac({}) dst-mac({}) \
    in-port({}) vlan-id ({})".format(a,b,c,d,e))

else:
    # identify the VLAN access port
    vlan_id = access_data[dpid][in_port]
    (a,b,c,d) = (dpid, src, dst, in_port)
    self.logger.info("untagged packet in  dp-id({}) src-mac({}) dst-mac({}) \
    in-port({})".format(a,b,c,d))

self.mac_to_port.setdefault(dpid, {})

dpid = datapath.id
self.mac_to_port.setdefault(dpid, {})

self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

# learn a mac address to avoid FLOOD next time.
self.mac_to_port[dpid][src] = in_port

if dst in self.mac_to_port[dpid]:
    out_port = self.mac_to_port[dpid][dst]
```

```
    self.logger.info("found the matched out port {}".format(out_port))
    self.forward_packet_to_a_vlan_port(msg, vlan_id, out_port, dst, src, tagged)
else:
    out_port = ofproto.OFPP_FLOOD
    self.logger.info("no match found, flood to vlan ports")
    self.flood_packet_to_all_vlan_ports(msg, vlan_id, tagged)
```

Delete the remaining code to the end of the program.

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

# install a flow to avoid packet_in next time
if out_port != ofproto.OFPP_FLOOD:
    match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
    # verify if we have a valid buffer_id, if yes avoid to send both
    # flow_mod & packet_out
    if msg.buffer_id != ofproto.OFP_NO_BUFFER:
        self.add_flow(datapath, 1, match, actions, msg.buffer_id)
        return
    else:
        self.add_flow(datapath, 1, match, actions)
data = None
if msg.buffer_id == ofproto.OFP_NO_BUFFER:
    data = msg.data

out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                          in_port=in_port, actions=actions, data=data)
datapath.send_msg(out)
```

15.2.6 Testing the VLANs

Start the Ryu controller.

```
sdn@sdn-mn:~$ ryu-manager ./example_scripts/VLAN_simple_switch_13.py
loading app ./example_scripts/VLAN_simple_switch_13.py
loading app ryu.controller.ofp_handler
instantiating app ./example_scripts/VLAN_simple_switch_13.py of VLANSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
```

Start the mininet topology.

```
sdn@sdn-mn:~$ cp ~/example_scripts/custom-OvS-VLAN.py ~/mininet/custom/
sdn@sdn-mn:~$ sudo ~/mininet/custom/custom-OvS-VLAN.py
*** Creating network
*** Adding hosts:
h1 h2 h3 h4 h5 h6
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (h5, s3) (h6, s3) (s1, s2) (s2, s3)
*** Configuring hosts
h1 h2 h3 h4 h5 h6
*** Starting network ***
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Running CLI ***
*** Starting CLI:
mininet>
```

Carry out a ‘pingall’ on the mininet topology and monitor the Ryu controller terminal.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X h3 X h5 X
h2 -> X X h4 X h6
h3 -> h1 X X h5 X
h4 -> X h2 X X h6
h5 -> h1 X h3 X X
h6 -> X h2 X h4 X
*** Results: 60% dropped (12/30 received)


{{ Ryu terminal output extract }}

untagged packet in  dp-id(3) src-mac(00:00:00:00:00:06) dst-
mac(00:00:00:00:00:04) in-port(3)
found the matched out port 1
forward packet to a port 1
Access port 3 to trunk port 1 : VLAN 200
vlan tagged packet in dp-id(2) src-mac(00:00:00:00:00:06) dst-
mac(00:00:00:00:00:04) in-port(2) vlan-id (200)
found the matched out port 4
forward packet to a port 4
Trunk port 2 to access port 4 : VLAN 200
untagged packet in  dp-id(2) src-mac(00:00:00:00:00:04) dst-
mac(00:00:00:00:00:06) in-port(4)
found the matched out port 2
forward packet to a port 2
Access port 4 to trunk port 2 : VLAN 200
vlan tagged packet in dp-id(3) src-mac(00:00:00:00:00:04) dst-
mac(00:00:00:00:00:06) in-port(1) vlan-id (200)
found the matched out port 3
forward packet to a port 3
Trunk port 1 to access port 3 : VLAN 200
untagged packet in  dp-id(3) src-mac(00:00:00:00:00:05) dst-
mac(ff:ff:ff:ff:ff:ff) in-port(2)
no match found, flood to vlan ports
flood packet: dpid (3) vlan_id (100) access ports ([2]) trunk ports ([1])
vlan tagged packet in dp-id(2) src-mac(00:00:00:00:00:05) dst-
mac(ff:ff:ff:ff:ff:ff) in-port(2) vlan-id (100)
no match found, flood to vlan ports
```

```
flood packet: dpid (2) vlan_id (100) access ports ([3]) trunk ports ([1, 2])
vlan tagged packet in dp-id(1) src-mac(00:00:00:00:00:05) dst-
mac(ff:ff:ff:ff:ff:ff) in-port(1) vlan-id (100)
no match found, flood to vlan ports
flood packet: dpid (1) vlan_id (100) access ports ([2]) trunk ports ([1])
untagged packet in  dp-id(3) src-mac(00:00:00:00:00:05) dst-
mac(ff:ff:ff:ff:ff:ff) in-port(2)
no match found, flood to vlan ports
flood packet: dpid (3) vlan_id (100) access ports ([2]) trunk ports ([1])
vlan tagged packet in dp-id(2) src-mac(00:00:00:00:00:05) dst-
mac(ff:ff:ff:ff:ff:ff) in-port(2) vlan-id (100)
```

```
sdn@sdn-mn:~$ cat ~/mininet/custom/custom-OvS-VLAN.py
```

```
1 #! /usr/bin/env python2
2 # -*- coding: utf-8 -*-
3
4 from mininet.topo import Topo
5 from mininet.net import Mininet
6 from mininet.node import RemoteController
7 from mininet.cli import CLI
8 from mininet.log import setLogLevel, info
9
10 """ Custom topology example to demonstrate VLANs """
11
12 # Declare variables
13 ryu_ip = '127.0.0.1'
14 ryu_port = 6653
15
16 class MyTopo( Topo ):
17     """Create a custom network class"""
18
19     def __init__(self):
20         """ Constructor method """
21
22         # Initialise topology
23         Topo.__init__(self)
24
25         # Add switches
26         s1,s2,s3 = [ self.addSwitch(s) for s in ('s1', 's2', 's3') ]
27
28         # Add hosts
29         h1 = self.addHost('h1', ip='192.168.1.1/24', mac='00:00:00:00:00:01')
30         h2 = self.addHost('h2', ip='192.168.1.2/24', mac='00:00:00:00:00:02')
31         h3 = self.addHost('h3', ip='192.168.1.3/24', mac='00:00:00:00:00:03')
32         h4 = self.addHost('h4', ip='192.168.1.4/24', mac='00:00:00:00:00:04')
33         h5 = self.addHost('h5', ip='192.168.1.5/24', mac='00:00:00:00:00:05')
34         h6 = self.addHost('h6', ip='192.168.1.6/24', mac='00:00:00:00:00:06')
35
36         # Add links
37         for sa, sb in [(s1, s2), (s2, s3)]:
38             self.addLink( sa, sb )
39
40         for h, s in [(h1, s1), (h2, s1), (h3, s2), (h4, s2), (h5, s3), (h6, s3)]:
41             self.addLink( h, s )
42
43 # main() function
44 def main():
45     """Test custom network"""
46
47     topo = MyTopo()
48     net = Mininet(topo=topo, controller=None)
49     net.addController('c0', controller=RemoteController,ip=ryu_ip, port=ryu_port)
50
51     info('*** Starting network ***\n')
52     net.start()
53
54     info('*** Running CLI ***\n' )
55     CLI(net)
56
57     info('*** Stopping network ***' )
58     net.stop()
59
60 # Call run_net() function
61 if __name__ == '__main__':
62     setLogLevel('info')
63     main()
64
65 exit(0)
```

Illustration 39: Mininet custom VLAN topology

Review the OvS switch flow tables.

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=370.388s, table=0, n_packets=4, n_bytes=296,
priority=1,in_port=1,dl_vlan=100,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01
actions=pop_vlan,output:2
 cookie=0x0, duration=367.338s, table=0, n_packets=4, n_bytes=296,
priority=1,in_port=1,dl_vlan=100,dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:01
actions=pop_vlan,output:2
 cookie=0x0, duration=358.268s, table=0, n_packets=4, n_bytes=296,
priority=1,in_port=1,dl_vlan=200,dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:02
actions=pop_vlan,output:3
 cookie=0x0, duration=355.154s, table=0, n_packets=4, n_bytes=296,
priority=1,in_port=1,dl_vlan=200,dl_src=00:00:00:00:00:06,dl_dst=00:00:00:00:00:02
actions=pop_vlan,output:3
 cookie=0x0, duration=370.385s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port=2,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03
actions=push_vlan:0x8100,set_field:4196->vlan_vid,output:1
 cookie=0x0, duration=367.333s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port=2,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:05
actions=push_vlan:0x8100,set_field:4196->vlan_vid,output:1
 cookie=0x0, duration=358.241s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port=3,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:04
actions=push_vlan:0x8100,set_field:4296->vlan_vid,output:1
 cookie=0x0, duration=355.149s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port=3,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:06
actions=push_vlan:0x8100,set_field:4296->vlan_vid,output:1
 cookie=0x0, duration=403.116s, table=0, n_packets=121, n_bytes=9403, priority=0
actions=CONTROLLER:65535

sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s2
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=372.787s, table=0, n_packets=4, n_bytes=280,
priority=1,in_port=3,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01
actions=push_vlan:0x8100,set_field:4196->vlan_vid,output:1
 cookie=0x0, duration=360.669s, table=0, n_packets=4, n_bytes=280,
priority=1,in_port=4,dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:02
actions=push_vlan:0x8100,set_field:4296->vlan_vid,output:1
 cookie=0x0, duration=351.437s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port=3,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:05
actions=push_vlan:0x8100,set_field:4196->vlan_vid,output:2
 cookie=0x0, duration=339.288s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port=4,dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:06
actions=push_vlan:0x8100,set_field:4296->vlan_vid,output:2
 cookie=0x0, duration=372.776s, table=0, n_packets=3, n_bytes=250,
priority=1,in_port=1,dl_vlan=100,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03
actions=pop_vlan,output:3
 cookie=0x0, duration=369.752s, table=0, n_packets=4, n_bytes=296,
priority=1,in_port=2,dl_vlan=100,dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:01
actions=output:1
 cookie=0x0, duration=369.724s, table=0, n_packets=3, n_bytes=250,
priority=1,in_port=1,dl_vlan=100,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:05
actions=output:2
 cookie=0x0, duration=360.628s, table=0, n_packets=3, n_bytes=250,
priority=1,in_port=1,dl_vlan=200,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:04
actions=pop_vlan,output:4
 cookie=0x0, duration=357.568s, table=0, n_packets=4, n_bytes=296,
priority=1,in_port=2,dl_vlan=200,dl_src=00:00:00:00:00:06,dl_dst=00:00:00:00:00:02
actions=output:1
 cookie=0x0, duration=357.543s, table=0, n_packets=3, n_bytes=250,
priority=1,in_port=1,dl_vlan=200,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:06
actions=output:2
 cookie=0x0, duration=351.478s, table=0, n_packets=4, n_bytes=296,
priority=1,in_port=2,dl_vlan=100,dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:03
actions=pop_vlan,output:3
 cookie=0x0, duration=339.326s, table=0, n_packets=4, n_bytes=296,
priority=1,in_port=2,dl_vlan=200,dl_src=00:00:00:00:00:06,dl_dst=00:00:00:00:00:04
actions=pop_vlan,output:4
 cookie=0x0, duration=405.510s, table=0, n_packets=142, n_bytes=12695, priority=0
actions=CONTROLLER:65535
```

```
sdn@sdn-mn:~$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s3
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=372.463s, table=0, n_packets=4, n_bytes=280,
priority=1,in_port=2,dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:01
actions=push_vlan:0x8100,set_field:4196->vlan_vid,output:1
 cookie=0x0, duration=360.276s, table=0, n_packets=4, n_bytes=280,
priority=1,in_port=3,dl_src=00:00:00:00:00:06,dl_dst=00:00:00:00:00:02
actions=push_vlan:0x8100,set_field:4296->vlan_vid,output:1
 cookie=0x0, duration=354.202s, table=0, n_packets=4, n_bytes=280,
priority=1,in_port=2,dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:03
actions=push_vlan:0x8100,set_field:4196->vlan_vid,output:1
 cookie=0x0, duration=342.040s, table=0, n_packets=4, n_bytes=280,
priority=1,in_port=3,dl_src=00:00:00:00:00:06,dl_dst=00:00:00:00:00:04
actions=push_vlan:0x8100,set_field:4296->vlan_vid,output:1
 cookie=0x0, duration=372.407s, table=0, n_packets=3, n_bytes=250,
priority=1,in_port=1,dl_vlan=100,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:05
actions=pop_vlan,output:2
 cookie=0x0, duration=360.224s, table=0, n_packets=3, n_bytes=250,
priority=1,in_port=1,dl_vlan=200,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:06
actions=pop_vlan,output:3
 cookie=0x0, duration=354.126s, table=0, n_packets=3, n_bytes=250,
priority=1,in_port=1,dl_vlan=100,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:05
actions=pop_vlan,output:2
 cookie=0x0, duration=341.968s, table=0, n_packets=3, n_bytes=250,
priority=1,in_port=1,dl_vlan=200,dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:06
actions=pop_vlan,output:3
 cookie=0x0, duration=408.210s, table=0, n_packets=121, n_bytes=9415, priority=0
actions=CONTROLLER:65535
```

16. Building a simple L3 and L4 switches

To this point, matches and flows have been based on L2 MAC addressing and in ports. However OpenFlow will work with matches to many fields. The following section demonstrates how the L2 simple switch can be modified to switch based on IP addressing at L3, the network layer and segment protocols at L4, the transport layer.

16.1 The simple network layer (L3) switch

Copy the simple switch code as follows:

```
sdn@sdn-mn:~$ cp ~/.local/lib/python3.8/site-packages/ryu
~/L3_simple_switch_13.py
```

Include the IPv4 library in the base classes and libraries.

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

This import introduces the IPv4 (RFC 791) header encoder/decoder class which includes ‘parser’ method that unpacks the IPv4 header into its constituent parts and a ‘serialize’ method that manipulates IPv4 packet headers.

The major change is the match line. The L2 simple switch matches the ‘in_port’, the ‘eth_dst’ and ‘eth_src’ values of the frame header in this line.

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

Replace this line with L3 logic. First the frame type field in the header is checked to determine it is an IP packet that is embedded in the frame, this is indicated by the Ethernet protocol type field indicates ‘0x800’. In this case the application extracts the source and destination IP addresses as a new match and send as a flow to the switch.

```
# install a flow to avoid packet_in next time
if out_port != ofproto.OFPP_FLOOD:

    # Original L2 switch match statement
    #match = parser.OFPMatch(in_port=in_port,
                            # eth_dst=dst,
                            # eth_src=src
                            #)

    # Check if frame header protocol type indicates IPv4 i.e. 0x800
    # If so extract the source and destination IP addresses as a new
    # match and send as a flow to the switch.
    if eth.ethertype == ether_types.ETH_TYPE_IP:
        ip = pkt.get_protocol(ipv4.ipv4)
        src_ip = ip.src
        dst_ip = ip.dst
        match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP,
                                ipv4_src=src_ip,
                                ipv4_dst=dst_ip
                                )
```

Illustration 40: L3 switch match logic

16.1.1 Running the L3 application

```
sdn@sdn-mn:~$ ryu-manager ./example_scripts/L3_simple_switch_13.py
loading app ./example_scripts/L3_simple_switch_13.py
loading app ryu.controller.ofp_handler
instantiating app ./example_scripts/L3_simple_switch_13.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
```

Running a simple topology of a single switch with two hosts connected for demonstration.

```
sdn@sdn-mn:~$ sudo mn --controller remote,ip=127.0.0.1  --switch
ovsk,protocols=OpenFlow13 --mac --ipbase=10.1.1.0/24 --topo single,2
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
```

16.1.2 Testing the L3 switch

Test connectivity between hosts.

```
mininet> pingall

*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
```

Review the flows sent to the OpenFlow switch.

```
sdn@sdn-mn:~/Desktop$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=84.556s, table=0, n_packets=2, n_bytes=196,
priority=1,ip,nw_src=10.1.1.2,nw_dst=10.1.1.1 actions=output:1
 cookie=0x0, duration=84.553s, table=0, n_packets=2, n_bytes=196,
priority=1,ip,nw_src=10.1.1.1,nw_dst=10.1.1.2 actions=output:2
 cookie=0x0, duration=90.252s, table=0, n_packets=26, n_bytes=1844, priority=0
actions=CONTROLLER:65535
```

Note the matches for the two flow entries and compare to the L2 switch. In this case the match is the IP addresses and not the in_port and MAC addresses.

- **Flow 1**
  - ip
    - nw_src=10.1.1.2
    - nw_dst=10.1.1.1
- **Flow 2**
  - ip
    - nw_src=10.1.1.1
    - nw_dst=10.1.1.2

16.2 The simple transport layer (L4) switch

Copy the simple switch code again and give it a L4 name:

```
sdn@sdn-mn:~$ cp ~/.local/lib/python3.8/site-packages/ryu
~/L4_simple_switch_13.py
```

Include the following libraries in the base classes and libraries section of the L4 application.

- **in_proto**: is a list of IP protocols. i.e. IPPROTO_TCP = 6 and IPPROTO_UDP = 17.
- **ipv4**: IPv4 header encoder/decoder class, as described earlier.
- **icmp**: ICMP (RFC 792) header encoder/decoder class.
- **tcp**: TCP (RFC 793) header encoder/decoder class
- **udp**: UDP (RFC 768) header encoder/decoder class

```
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
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
```

The major change is the match section. The L2 simple switch matches the ‘in_port’, the ‘eth_dst’ and ‘eth_src’ values of the frame header in this line block, the L3 switch matched IPv4.

```
# install a flow to avoid packet_in next time
if out_port != ofproto.OFPP_FLOOD:
    match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
    # verify if we have a valid buffer_id, if yes avoid to send both
    # flow_mod & packet_out
    if msg.buffer_id != ofproto.OFP_NO_BUFFER:
        self.add_flow(datapath, 1, match, actions, msg.buffer_id)
        return
    else:
        self.add_flow(datapath, 1, match, actions)
```

Replace this line with L4 logic. As there is a lot of code it is described here in small blocks.

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

16.2.1 Internet protocol

This block is similar to the adjustments made for the L3 switch. If the Ethernet protocol type field indicates ‘0x800’. The items extracted are:

- **ip**: IPv4 packet header
  - csum=18381,dst='10.1.1.2',flags=2,header_length=5,identification=56535,offset=0,option=None,proto=1,src='10.1.1.1',tos=0,total_length=84,ttl=64,version=4
- **src_ip:** Source IP address
- **dst_ip**: Destination IP address
- **protocol**: L4 protocol, for example IPPROTO_ICMP = 1.

16.2.2 Internet Control Message Protocol

The next block is for dealing with ICMP messages. If the protocol is ICMP

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

                           )
```

- **ipv4_src**: Source IPv4 address
- **ipv4_dst**: Destination IPv4 address
- **ip_proto**: IP protocol, in this case IPPROTO_TCP = 1

16.2.3 Transmission Control Protocol

This block deals with the TCP segments.

```
#  if TCP Protocol
elif protocol == in_proto.IPPROTO_TCP:
    _tcp = pkt.get_protocol(tcp.tcp)
    match = parser.OFPMatch(eth_type = ether_types.ETH_TYPE_IP,
                            ipv4_src = src_ip,
                            ipv4_dst = dst_ip,
                            ip_proto = protocol,
                            tcp_src = _tcp.src_port,
                            tcp_dst = _tcp.dst_port
                           )
```

- **_tcp**: TCP segment header
    - ack=2450347376, bits=18, csum=54898, dst_port=42830, offset=10,
    - option=[TCPOptionMaximumSegmentSize(kind=2, length=4, max_seg_size=1460),
      TCPOptionSACKPermitted(kind=4, length=2),
      TCPOptionTimestamps(kind=8, length=10, ts_ecr=24206664, ts_val=24206667),
      TCPOptionNoOperation(kind=1, length=1),
      TCPOptionWindowScale(kind=3, length=3, shift_cnt=9)]
    - seq=3770006798, src_port=80, urgent=0, window_size=28960)
- **ipv4_src**: Source IPv4 address
- **ipv4_dst**: Destination IPv4 address
- **ip_proto**: IP protocol, in this case IPPROTO_TCP = 6
- **tcp_src**: TCP source port
- **tcp_dst**: TCP destination port

16.2.4 User Datagram Protocol

This block deals with the UDP segments.

```
#  if UDP Protocol
elif protocol == in_proto.IPPROTO_UDP:
_udp = pkt.get_protocol(udp.udp)
match = parser.OFPMatch(eth_type = ether_types.ETH_TYPE_IP,
                        ipv4_src = src_ip,
                        ipv4_dst = dst_ip,
                        ip_proto = protocol,
                        udp_src = _udp.src_port,
                        udp_dst = _udp.dst_port
                       )
```

- **_udp**: UDP segment header
    - csum=53278,dst_port=5001,src_port=48634,total_length=1478
- **ipv4_src**: Source IPv4 address
- **ipv4_dst**: Destination IPv4 address
- **ip_proto**: IP protocol, in this case IPPROTO_TCP = 17
- **udp_src**: UDP source port
- **udp_dst**: UDP destination port

16.2.5 Putting it all together

Illustration 41 puts these new blocks together. If the packet is not a flood packet then match if it is IPv4 as for the L3 switch. If it is IPv4 then test if it is ICMP, TCP or UDP and handle as described.

16.2.6 Run the L4 application

Run the new L4 application.

```
sdn@sdn-mn:~$ ryu-manager ./example_scripts/L4_simple_switch_13.py
loading app ./example_scripts/L4_simple_switch_13.py
loading app ryu.controller.ofp_handler
instantiating app ./example_scripts/L4_simple_switch_13.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
```

Running a simple topology of a single switch with two hosts connected for demonstration.

```
# install a flow to avoid packet_in next time
if out_port != ofproto.OFPP_FLOOD:

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

        # Handle ICMP Protocol
        if protocol == in_proto.IPPROTO_ICMP:
            match = parser.OFPMatch(eth_type = ether_types.ETH_TYPE_IP,
                                    ipv4_src = src_ip,
                                    ipv4_dst = dst_ip,
                                    ip_proto = protocol
                                   )

        #  Handle TCP Protocol
        elif protocol == in_proto.IPPROTO_TCP:
            _tcp = pkt.get_protocol(tcp.tcp)
            match = parser.OFPMatch(eth_type = ether_types.ETH_TYPE_IP,
                                    ipv4_src = src_ip,
                                    ipv4_dst = dst_ip,
                                    ip_proto = protocol,
                                    tcp_src = _tcp.src_port,
                                    tcp_dst = _tcp.dst_port
                                   )

        #  Handle UDP Protocol
        elif protocol == in_proto.IPPROTO_UDP:
            _udp = pkt.get_protocol(udp.udp)
            match = parser.OFPMatch(eth_type = ether_types.ETH_TYPE_IP,
                                    ipv4_src = src_ip,
                                    ipv4_dst = dst_ip,
                                    ip_proto = protocol,
                                    udp_src = _udp.src_port,
                                    udp_dst = _udp.dst_port
                                   )
```

Illustration 41: L4 switch match logic

```
sdn@sdn-mn:~$ sudo mn --controller remote,ip=127.0.0.1  --switch
ovsk,protocols=OpenFlow13 --mac --ipbase=10.1.1.0/24 --topo single,2
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
```

16.2.7 Testing ICMP

Test connectivity between hosts, in other words test ICMP.

```
mininet> pingall

*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
```

Review the flows sent to the OpenFlow switch.

```
sdn@sdn-mn:~/Desktop$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):

 cookie=0x0, duration=9.429s, table=0, n_packets=2, n_bytes=196,
priority=1,icmp,nw_src=10.1.1.1,nw_dst=10.1.1.2 actions=output:2

 cookie=0x0, duration=9.428s, table=0, n_packets=2, n_bytes=196,
priority=1,icmp,nw_src=10.1.1.2,nw_dst=10.1.1.1 actions=output:1

 cookie=0x0, duration=12.987s, table=0, n_packets=20, n_bytes=1480, priority=0
actions=CONTROLLER:65535
```

In this case the matches are:

- **Flow 1**
  - icmp
    - nw_src=10.1.1.1
    - nw_dst=10.1.1.2
- **Flow 2**
  - icmp
    - nw_src=10.1.1.2
    - nw_dst=10.1.1.1

16.2.8 Testing TCP

Use ‘iperf’ to generate TCP traffic.

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['14.7 Gbits/sec', '14.8 Gbits/sec']
```

Review the flows sent to the OpenFlow switch.

```
sdn@sdn-mn:~/Desktop$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):

 cookie=0x0, duration=20.311s, table=0, n_packets=1, n_bytes=74,
priority=1,tcp,nw_src=10.1.1.1,nw_dst=10.1.1.2,tp_src=41978,tp_dst=5001
actions=output:2

 cookie=0x0, duration=20.300s, table=0, n_packets=1, n_bytes=54,
priority=1,tcp,nw_src=10.1.1.2,nw_dst=10.1.1.1,tp_src=5001,tp_dst=41978
actions=output:1

 cookie=0x0, duration=35.172s, table=0, n_packets=22, n_bytes=1624, priority=0
actions=CONTROLLER:65535
```

In this case the matches are:

- **Flow 1**
  - tcp
    - nw_src=10.1.1.1
    - nw_dst=10.1.1.2
    - tp_src=41978
    - tp_dst=5001
- **Flow 2**
  - tcp
    - nw_src=10.1.1.2
    - nw_dst=10.1.1.1
    - tp_src=5001
    - tp_dst=41978

16.2.9 Testing UDP

Use ‘iperfudp’ to generate TCP traffic.

```
mininet> iperfudp 10m h1 h2
*** Iperf: testing UDP bandwidth between h1 and h2
*** Results: ['10m', '10.0 Mbits/sec', '10.0 Mbits/sec']
```

Review the flows sent to the OpenFlow switch.

```
sdn@sdn-mn:~/Desktop$ sudo ovs-ofctl --protocols OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):

 cookie=0x0, duration=35.527s, table=0, n_packets=4252, n_bytes=6429024,
priority=1,udp,nw_src=10.1.1.1,nw_dst=10.1.1.2,tp_src=42475,tp_dst=5001
actions=output:2

 cookie=0x0, duration=30.509s, table=0, n_packets=1, n_bytes=1512,
priority=1,udp,nw_src=10.1.1.2,nw_dst=10.1.1.1,tp_src=5001,tp_dst=42475
actions=output:1

 cookie=0x0, duration=43.011s, table=0, n_packets=21, n_bytes=4898, priority=0
actions=CONTROLLER:65535
```

In this case the matches are:

- **Flow 1**
  - udp
    - nw_src=10.1.1.1
    - nw_dst=10.1.1.2
    - tp_src= 42475
    - tp_dst=5001
- **Flow 2**
  - udp
    - nw_src=10.1.1.2
    - nw_dst=10.1.1.1
    - tp_src=5001
    - tp_dst=42475

17. Bibliography

[1] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, ‘Ethane: Taking control of the enterprise’, in *ACM SIGCOMM Computer Communication Review*, 2007, vol. 37, pp. 1–12.
[2] R. Fielding, ‘Fielding dissertation: Chapter 5: Representational state transfer (rest)’, *Recuperado el*, vol. 8, 2000.
[3] S. Raza and D. Lenrow, ‘North Bound Interface Working Group (NBI-WG) Charter’. Open Networking Foundation, 06-Oct-2013.
[4] Open Networking Foundation (ONF), ‘OpenFlow Switch Specification, Version 1.5.1 (Protocol version 0x06)’. Open Networking Foundation (ONF), 26-Mar-2015.
[5] Open Networking Foundation (ONF), ‘OpenFlow Switch Specification, Version 1.3.0 (Protocol version 0x04)’. Open Networking Foundation (ONF), 25-Jun-2012.
[6] ‘Mininet Overview’, *Mininet*. [Online]. Available: http://mininet.org/overview/. [Accessed: 02-Oct-2019].
[7] RYU project team, ‘ryu Documentation’, *Ryubook documentation Release 4.34*. [Online]. Available: https://readthedocs.org/projects/ryu/downloads/pdf/latest/. [Accessed: 13-May-2020].

This page is intentionally blank