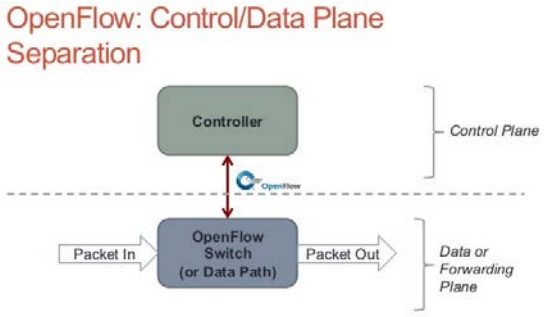
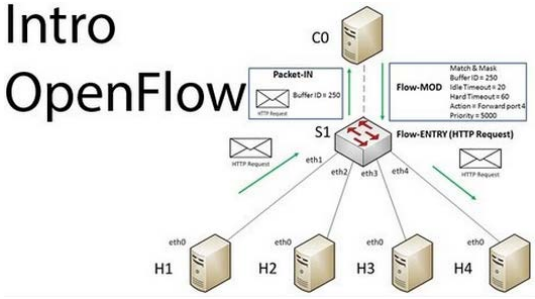
Lab 3 - Build a Learning Switch on Ryu

Additional Remarks

1. Openflow protocol (packet-in & packet out)





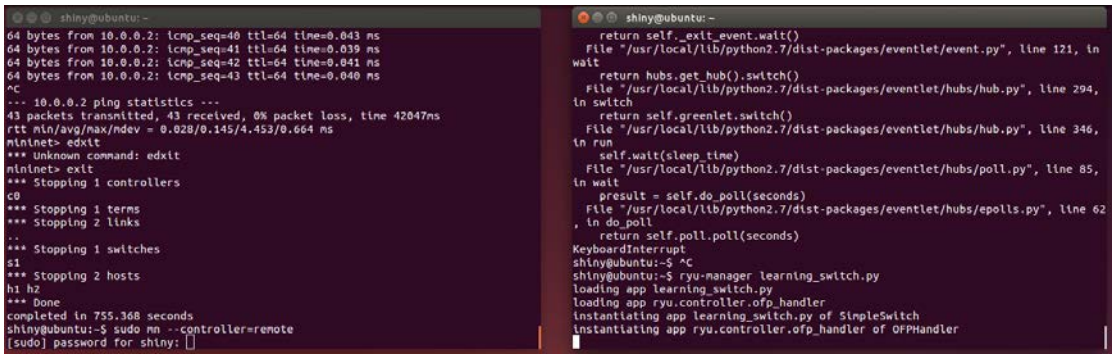
2. OpenFlow Datapath ID (DPID), datapath(dp) ~= switch

How are OpenFlow switches identified?

Each OpenFlow instance on a switch is identified by a Datapath Identifier. This is a 64 bit number determined as follows according to the OpenFlow specification:

"The datapath_id fleld uniquely identifies a datapath. The lower 48 bits are intended for the switch MAC address, while the top 16 bits are up to the implementer. An example use of the top 16 bits would be a VLAN ID to distinguish multiple virtual switch instances on a single physical switch."

3. You need to run controller (ryu) and mininet in two different terminal



4. You could acquire info of how to use the ryu's API here:

http://ryu.readthedocs.org/en/latest/search.html

ex: OFPActionOutput、OFPPacketOut……

5. OFPActionOutput

class ryu.ofproto.ofproto_v1_2_parser.OFPActionOutput(port, max_len=65509, type_=None, len_=None)

Output action This action indicates output a packet to the switch port.

Attribute	Description
port	Output port
max_len	Max length to send to controller

6. OFPPacketOut

class ryu.ofproto.ofproto_v1_2_parser.**OFPPacketOut**(*datapath, buffer_id=None, in_port=None, actions=None, data=None, actions_len=None*)

Packet-Out message The controller uses this message to send a packet out throught the switch.

Attribute	**Description**
buffer_id	ID assigned by datapath (OFP_NO_BUFFER if none)
in_port	Packet's input port or OFPP_CONTROLLER
actions	list of OpenFlow action class
data	Packet data

Example:

```
def send_packet_out(self, datapath, buffer_id, in_port):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser
    actions = [ofp_parser.OFPActionOutput(ofp.OFPP_FLOOD, 0)]
    req = ofp_parser.OFPPacketOut(datapath, buffer_id,
                                  in_port, actions)
    datapath.send_msg(req)
```

7. How to add flow entry in switch

install a flow to avoid packet_in next time

Use the function in the skeleton code: **add_flow(self, datapath, in_port, dst, actions)**

Add_flow function use the **API :OFPFlowMod** to modify the flow entry in the SDN switch

(Already implemented for you in the function add_flow)

Here is just reference of OFPFlowMod which you could also find in the website

class ryu.ofproto.ofproto_v1_2_parser.**OFPFlowMod**(*datapath, cookie=0, cookie_mask=0, table_id=0, command=0, idle_timeout=0, hard_timeout=0, priority=0,buffer_id=4294967295, out_port=0, out_group=0, flags=0, match=None, instructions=[]*)

Modify Flow entry message

The controller sends this message to modify the flow table.

Attribute	**Description**
cookie	Opaque controller-issued identifier
cookie_mask	Mask used to restrict the cookie bits that must match when the command is OPFFC_MODIFY* or OFPFC_DELETE*
table_id	ID of the table to put the flow in
command	One of the following values. OFPFC_ADD OFPFC_MODIFY OFPFC_MODIFY_STRICT OFPFC_DELETE OFPFC_DELETE_STRICT
idle_timeout	Idle time before discarding (seconds)
hard_timeout	Max time before discarding (seconds)
priority	Priority level of flow entry
buffer_id	Buffered packet to apply to (or OFP_NO_BUFFER)
out_port	For OFPFC_DELETE* commands, require matching entries to include this as an output port
out_group	For OFPFC_DELETE* commands, require matching entries to include this as an output group
flags	One of the following values. OFPFF_SEND_FLOW_REM OFPFF_CHECK_OVERLAP OFPFF_RESET_COUNTS
match	Instance of OFPMatch
instructions	list of OFPInstruction* instance

http://mininet.org/sample-workflow/

8. Using a Remote Controller

Note: this step is not part of the default walkthrough; it is primarily useful if you have a controller running outside of the VM, such as on the VM host, or a different physical PC. The OpenFlow tutorial uses ***`controller --remote`*** *for starting up a simple learning switch that you create using a controller framework like POX, NOX, Beacon or Floodlight.*

When you start a Mininet network, each switch can be connected to a remote controller - which could be in the VM, outside the VM and on your local machine, or anywhere in the world.

This setup may be convenient if you already have a custom version of a controller framework and development tools (such as Eclipse) installed on the local machine, or you want to test a controller running on a different physical machine (maybe even in the cloud).

If you want to try this, fill in the host IP and/or listening port:

```
$ sudo mn --controller=remote,ip=[controller IP],port=[controller listening port]
```

Default controller port = 6633

9. Changing Topology Size and Type

The default topology is a single switch connected to two hosts. You could change this to a different topo with `--topo`, and pass parameters for that topology's creation. For example, to verify all-pairs ping connectivity with one switch and three hosts:

Run a regression test:

```
$ sudo mn --test pingall --topo single,3
```

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

```
$ sudo mn --test pingall --topo linear,4
```