Lab5: Observing Flow Table Overflows

# Name: Sicong Chen ID: N10969883 Date: April, 2017

# Name: Sijia Jiang ID: N11642702 Date: April, 2017

# Objectives

* Fully understand the operation of OpenFlow and observe the operations.
* Learn basic skill of configuring an OpenFlow Switch.
* Trigger and observe the behavior of flow table overflow.
* Learn to manage flows based on limited network resources

# Equipment Needs

* Computer/Laptop/VM (Linux highly recommended)

# Experiments

## Preparation

1. Choose one controller stack from below to complete the rest of this lab

Ryu OpenFlow tutorial:

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

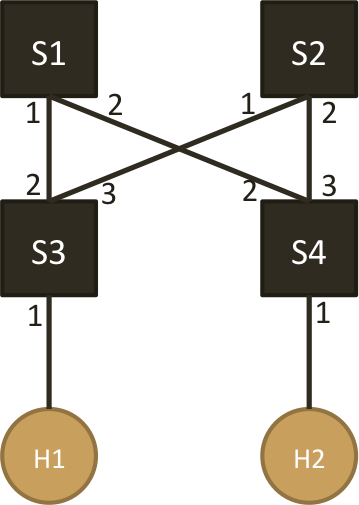
Beacon OpenFlow tutorial <http://archive.openflow.org/wk/index.php/OpenFlow_Tutorial#Create_Learning_Switch>

OpenDaylight controller developing wiki:

\* Note: this controller stack is quite complicated. As a starter at programming, I do not suggest you use it. But anyone realizing this project using ODL controller gets bonus points.

<http://www.projectfloodlight.org/getting-started/>

## Observing the table overflow in OVS

1. Use mininet to create the topology shown below.
2. Set the flow table size of S1 and S2 to 100
3. Provide a controller that
   1. upon receiving a new TCP flow, the switch (S3 or S4) forwards the first packet to the controller
   2. controller installs a path S3-S1-S4 for this TCP flow on both directions (2 rules per flow)
4. Use hping3 to send 60 TCP flow with varied destination ports from H1 to H2 and observe flow table overflow
5. Modify your controller so that
   1. 50% of TCP flows follow path S3-S1-S4
   2. 50% of TCP flows follow path S3-S2-S4
6. Observe the flow table overflow still happens

**4. Lab Reports**

1. What is the command for setting the flow-table size in Open vSwitch. Please explain the meaning of each option.

The command for setting the flow-table size is

**sh ovs-vsctl add Bridge s1 flow\_tables 0=@sicong -- --id=@sicong create flow\_table flow\_limit=100**

**ovs-vsctl:** The ovs-vsctl program configures ovs-virture switchd by providing a high-level interface to its configuration database

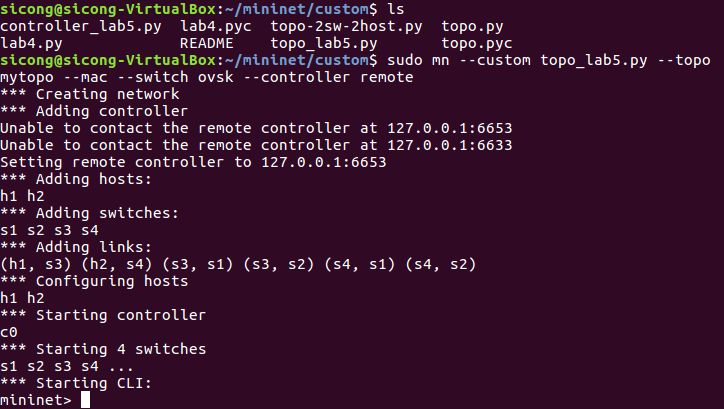
**add Bridge s1:** add switch S1

**flow\_tables 0=@siocng -- --id=@sicong create flow\_table: C**reate flow table

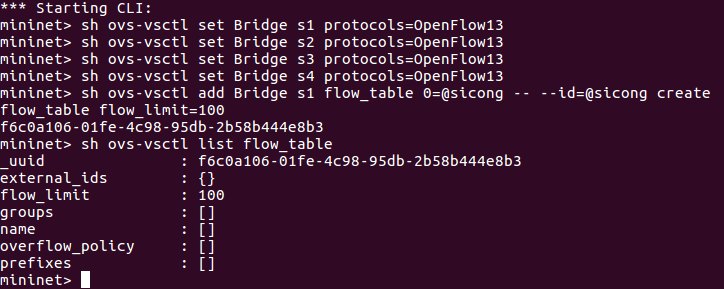
as 0.

**flow\_limit=100: s**et flow table size to 100

1. Submit the screenshot(s) showing that the flow tables of S1 & S2 are configured to a required size.

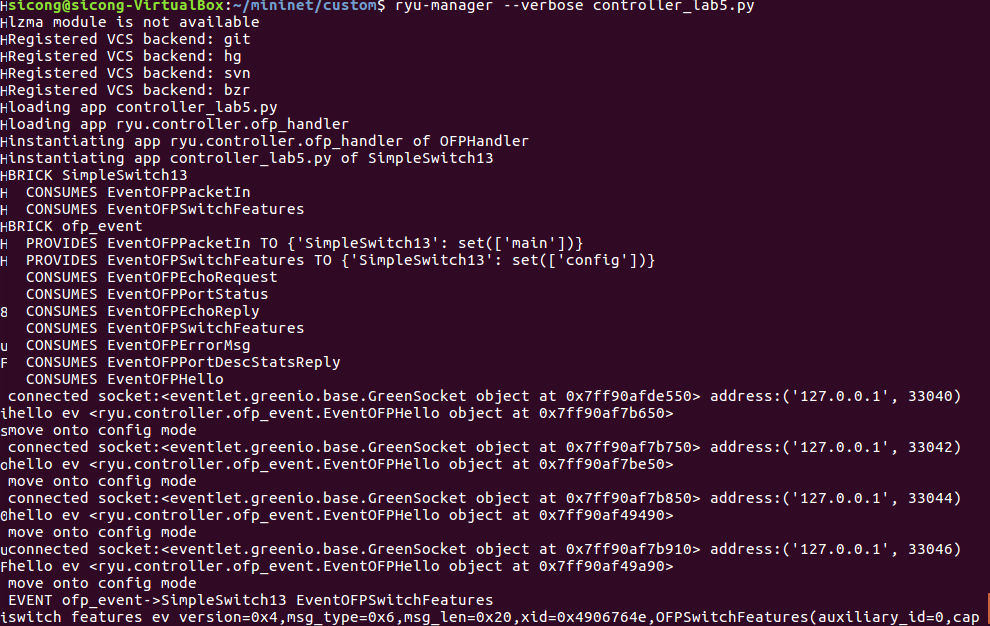


**set flow table size to 100:**

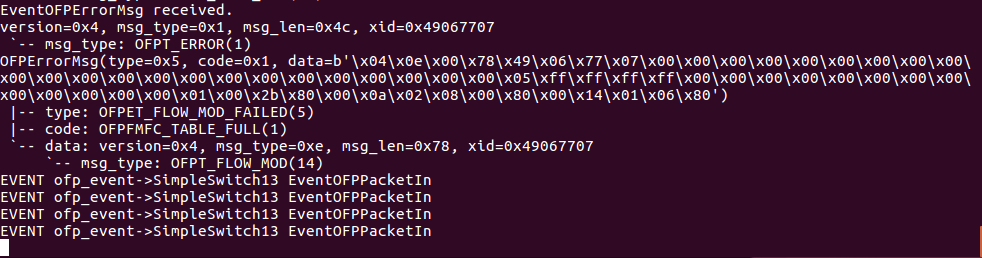


1. Submit the screenshot(s) showing flow table overflow message(s) sent by the controller.

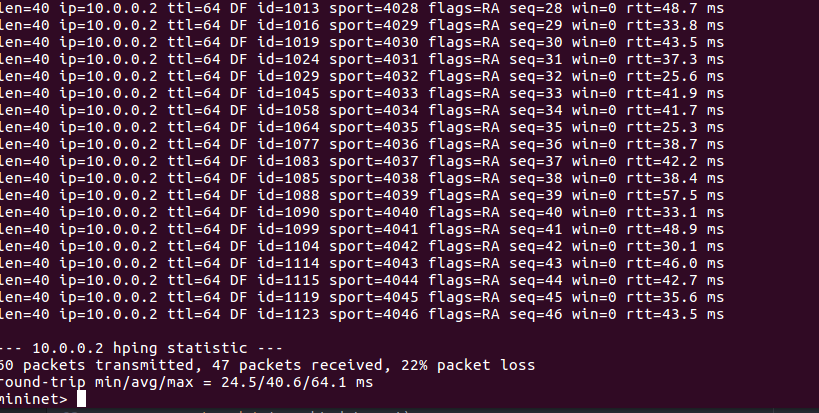
**When there is no overflow. After ryu-manager:**



when **mininet> h1 hping3 –c 60 –p ++3000 –i u50000 h2** overflow:

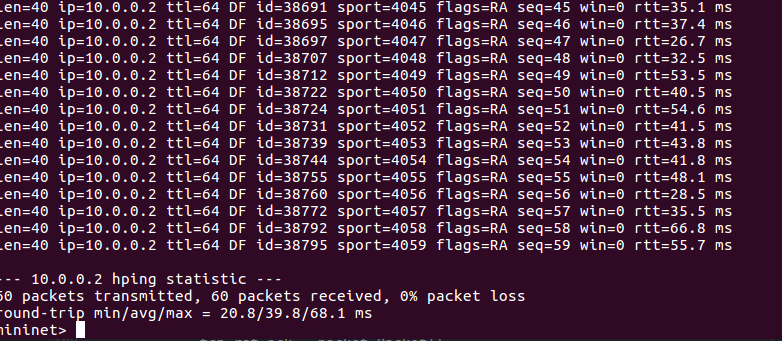


h1 hping3 h2:



1. Submit the screenshot(s) showing there is no flow table overflow after modifying the controller.

**50%-50% multipath. 0 packet loss:**



1. Submit your controller code, both single path and 50%-50% multipath. Add comments to clarify the logic if necessary.

**Code single path:**

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.ofproto import ether

from ryu.ofproto import inet

from ryu.lib.packet import packet

from ryu.lib.packet import ethernet

from ryu.lib.packet import arp

from ryu.lib.packet import ipv4

from ryu.lib.packet import tcp

from ryu.lib.packet import udp

class SimpleSwitch13(app\_manager.RyuApp):

OFP\_VERSIONS = [ofproto\_v1\_3.OFP\_VERSION]

def \_\_init\_\_(self, \*args, \*\*kwargs):

super(SimpleSwitch13, self).\_\_init\_\_(\*args, \*\*kwargs)

# arp table: for searching

self.arp\_table={}

self.arp\_table['10.0.0.1'] = '00:00:00:00:00:01'

self.arp\_table['10.0.0.2'] = '00:00:00:00:00:02'

@set\_ev\_cls(ofp\_event.EventOFPSwitchFeatures, CONFIG\_DISPATCHER)

def switch\_features\_handler(self, ev):

datapath = ev.msg.datapath

ofproto = datapath.ofproto

parser = datapath.ofproto\_parser

# Insert Static rule

match = parser.OFPMatch()

actions = [parser.OFPActionOutput(ofproto.OFPP\_CONTROLLER,

ofproto.OFPCML\_NO\_BUFFER)]

self.add\_flow(datapath, 0, match, actions)

# Installing static rules to process TCP/UDP and ICMP and ACL

dpid = datapath.id # classifying the switch ID

if dpid == 1: # switch S1

self.add\_layer4\_rules(datapath, inet.IPPROTO\_ICMP, '10.0.0.1', 10, 1)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_ICMP, '10.0.0.2', 10, 2)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.1', 20, 1)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.2', 20, 2)

elif dpid == 3: # switch S3

self.add\_layer4\_rules(datapath, inet.IPPROTO\_ICMP, '10.0.0.1', 10, 1)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_ICMP, '10.0.0.2', 10, 2)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.1', 20, 1)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.2', 20, 2)

elif dpid == 4: # switch S4

self.add\_layer4\_rules(datapath, inet.IPPROTO\_ICMP, '10.0.0.1', 20, 2)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_ICMP, '10.0.0.2', 20, 1)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.1', 20, 2)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.2', 20, 1)

else:

print "wrong switch"

@set\_ev\_cls(ofp\_event.EventOFPPacketIn, MAIN\_DISPATCHER)

def \_packet\_in\_handler(self, ev):

msg = ev.msg

datapath = msg.datapath

ofproto = datapath.ofproto

parser = datapath.ofproto\_parser

in\_port = msg.match['in\_port']

pkt = packet.Packet(msg.data)

eth = pkt.get\_protocol(ethernet.ethernet)

ethertype = eth.ethertype

# process ARP

if ethertype == ether.ETH\_TYPE\_ARP:

self.handle\_arp(datapath, in\_port, pkt)

return

eth\_pkt = pkt.get\_protocol(ethernet.ethernet)

ipv4\_pkt = pkt.get\_protocol(ipv4.ipv4)

tcp\_pkt=pkt.get\_protocol(tcp.tcp)

self.logger.info("packet is %s"%(pkt,))

if datapath.id == 1:

tcp\_rst\_ack = packet.Packet()

# 314

match = parser.OFPMatch(eth\_type=0X0800,ip\_proto=6,

ipv4\_src='10.0.0.1',

ipv4\_dst='10.0.0.2',

tcp\_src=tcp\_pkt.src\_port,

tcp\_dst=tcp\_pkt.dst\_port)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(2)]

self.add\_flow(datapath,5,match,actions)

# 413

match = parser.OFPMatch(eth\_type=0X0800,ip\_proto=6,

ipv4\_src='10.0.0.2',

ipv4\_dst='10.0.0.1',

tcp\_src=tcp\_pkt.dst\_port,

tcp\_dst=tcp\_pkt.src\_port)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(1)]

self.add\_flow(datapath,5,match,actions)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(2)]

out = parser.OFPPacketOut(datapath, ofproto.OFP\_NO\_BUFFER,

ofproto.OFPP\_CONTROLLER, actions,

msg.data)

datapath.send\_msg(out)

elif datapath.id == 3:

tcp\_rst\_ack = packet.Packet()

# 314

match = parser.OFPMatch(eth\_type=0X0800,ip\_proto=6,

ipv4\_src='10.0.0.1',

ipv4\_dst='10.0.0.2',

tcp\_src=tcp\_pkt.src\_port,

tcp\_dst=tcp\_pkt.dst\_port)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(2)]

self.add\_flow(datapath,5,match,actions)

# 413

match = parser.OFPMatch(eth\_type=0X0800,ip\_proto=6,

ipv4\_src='10.0.0.2',

ipv4\_dst='10.0.0.1',

tcp\_src=tcp\_pkt.dst\_port,

tcp\_dst=tcp\_pkt.src\_port)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(1)]

self.add\_flow(datapath,5,match,actions)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(2)]

out = parser.OFPPacketOut(datapath, ofproto.OFP\_NO\_BUFFER,

ofproto.OFPP\_CONTROLLER, actions,

msg.data)

datapath.send\_msg(out)

elif datapath.id == 4:

tcp\_rst\_ack = packet.Packet()

# 314

match = parser.OFPMatch(eth\_type=0X0800,ip\_proto=6,

ipv4\_src='10.0.0.1',

ipv4\_dst='10.0.0.2',

tcp\_src=tcp\_pkt.src\_port,

tcp\_dst=tcp\_pkt.dst\_port)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(1)]

self.add\_flow(datapath,5,match,actions)

# 413

match = parser.OFPMatch(eth\_type=0X0800,ip\_proto=6,

ipv4\_src='10.0.0.2',

ipv4\_dst='10.0.0.1',

tcp\_src=tcp\_pkt.dst\_port,

tcp\_dst=tcp\_pkt.src\_port)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(2)]

self.add\_flow(datapath,5,match,actions)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(1)];

out = parser.OFPPacketOut(datapath, ofproto.OFP\_NO\_BUFFER,

ofproto.OFPP\_CONTROLLER, actions,

msg.data)

datapath.send\_msg(out)

else:

print "wrong switch"

# Member methods you can call to install TCP/UDP/ICMP fwding rules

def add\_layer4\_rules(self, datapath, ip\_proto, ipv4\_dst = None, priority = 1, fwd\_port = None):

parser = datapath.ofproto\_parser

actions = [parser.OFPActionOutput(fwd\_port)]

match = parser.OFPMatch(eth\_type = ether.ETH\_TYPE\_IP,

ip\_proto = ip\_proto,

ipv4\_dst = ipv4\_dst)

self.add\_flow(datapath, priority, match, actions)

# Member methods you can call to install general rules

def add\_flow(self, datapath, priority, match, actions):

ofproto = datapath.ofproto

parser = datapath.ofproto\_parser

inst = [parser.OFPInstructionActions(ofproto.OFPIT\_APPLY\_ACTIONS,

actions)]

mod = parser.OFPFlowMod(datapath=datapath, priority=priority,

match=match, instructions=inst)

datapath.send\_msg(mod)

def handle\_arp(self, datapath, in\_port, pkt):

ofproto = datapath.ofproto

parser = datapath.ofproto\_parser

# parse out the ethernet and arp packet

eth\_pkt = pkt.get\_protocol(ethernet.ethernet)

arp\_pkt = pkt.get\_protocol(arp.arp)

# obtain the MAC of dst IP

arp\_resolv\_mac = self.arp\_table[arp\_pkt.dst\_ip]

### generate the ARP reply msg, please refer RYU documentation

### the packet library section

ether\_hd = ethernet.ethernet(dst = eth\_pkt.src,

src = arp\_resolv\_mac,

ethertype = ether.ETH\_TYPE\_ARP);

arp\_hd = arp.arp(hwtype=1, proto = 2048, hlen = 6, plen = 4,

opcode = 2, src\_mac = arp\_resolv\_mac,

src\_ip = arp\_pkt.dst\_ip, dst\_mac = eth\_pkt.src,

dst\_ip = arp\_pkt.src\_ip);

arp\_reply = packet.Packet()

arp\_reply.add\_protocol(ether\_hd)

arp\_reply.add\_protocol(arp\_hd)

arp\_reply.serialize()

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(in\_port)];

out = parser.OFPPacketOut(datapath, ofproto.OFP\_NO\_BUFFER,

ofproto.OFPP\_CONTROLLER, actions,

arp\_reply.data)

datapath.send\_msg(out)

**code 50%-50% multipath:**

(For multipath , because the source port start from 3000. We can separate the flow from different port to go to different path.

We modules port by 2. If the residues is 0, flow take path s3-s1-s4.

If the residues is 1, flow take path s3-s2-s4.)

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.ofproto import ether

from ryu.ofproto import inet

from ryu.lib.packet import packet

from ryu.lib.packet import ethernet

from ryu.lib.packet import arp

from ryu.lib.packet import ipv4

from ryu.lib.packet import tcp

from ryu.lib.packet import udp

class SimpleSwitch13(app\_manager.RyuApp):

OFP\_VERSIONS = [ofproto\_v1\_3.OFP\_VERSION]

"""

Constructor:

You can define some globally used variables inside the class

"""

def \_\_init\_\_(self, \*args, \*\*kwargs):

super(SimpleSwitch13, self).\_\_init\_\_(\*args, \*\*kwargs)

# arp table: for searching

self.arp\_table={}

self.arp\_table['10.0.0.1'] = '00:00:00:00:00:01'

self.arp\_table['10.0.0.2'] = '00:00:00:00:00:02'

@set\_ev\_cls(ofp\_event.EventOFPSwitchFeatures, CONFIG\_DISPATCHER)

def switch\_features\_handler(self, ev):

datapath = ev.msg.datapath

ofproto = datapath.ofproto

parser = datapath.ofproto\_parser

# Insert Static rule

match = parser.OFPMatch()

actions = [parser.OFPActionOutput(ofproto.OFPP\_CONTROLLER,

ofproto.OFPCML\_NO\_BUFFER)]

self.add\_flow(datapath, 0, match, actions)

# Installing static rules to process TCP/UDP and ICMP and ACL

dpid = datapath.id # classifying the switch ID

if dpid == 1: # switch S1

self.add\_layer4\_rules(datapath, inet.IPPROTO\_ICMP, '10.0.0.1', 10, 1)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_ICMP, '10.0.0.2', 10, 2)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.1', 20, 1)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.2', 20, 2)

elif dpid == 3 : # switch S3

self.add\_layer4\_rules(datapath, inet.IPPROTO\_ICMP, '10.0.0.1', 10, 1)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_ICMP, '10.0.0.2', 10, 2)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.1', 20, 1)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.2', 20, 2)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_ICMP, '10.0.0.1', 20, 3)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_ICMP, '10.0.0.2', 20, 1)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.1', 20, 3)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.2', 20, 1)

elif dpid == 4 : # switch S4

self.add\_layer4\_rules(datapath, inet.IPPROTO\_ICMP, '10.0.0.1', 20, 2)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_ICMP, '10.0.0.2', 20, 1)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.1', 20, 2)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.2', 20, 1)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_ICMP, '10.0.0.1', 20, 3)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_ICMP, '10.0.0.2', 20, 1)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.1', 20, 3)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.2', 20, 1)

else:

print "wrong switch"

@set\_ev\_cls(ofp\_event.EventOFPPacketIn, MAIN\_DISPATCHER)

def \_packet\_in\_handler(self, ev):

msg = ev.msg

datapath = msg.datapath

ofproto = datapath.ofproto

parser = datapath.ofproto\_parser

in\_port = msg.match['in\_port']

pkt = packet.Packet(msg.data)

eth = pkt.get\_protocol(ethernet.ethernet)

ethertype = eth.ethertype

# process ARP

if ethertype == ether.ETH\_TYPE\_ARP:

self.handle\_arp(datapath, in\_port, pkt)

return

eth\_pkt = pkt.get\_protocol(ethernet.ethernet)

ipv4\_pkt = pkt.get\_protocol(ipv4.ipv4)

tcp\_pkt=pkt.get\_protocol(tcp.tcp)

self.logger.info("packet is %s"%(pkt,))

if datapath.id == 1:

tcp\_rst\_ack = packet.Packet()

# 314

match = parser.OFPMatch(eth\_type=0X0800,ip\_proto=6,

ipv4\_src='10.0.0.1',

ipv4\_dst='10.0.0.2',

tcp\_src=tcp\_pkt.src\_port,

tcp\_dst=tcp\_pkt.dst\_port)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(2)];

self.add\_flow(datapath,5,match,actions)

# 413

match = parser.OFPMatch(eth\_type=0X0800,ip\_proto=6,

ipv4\_src='10.0.0.2',

ipv4\_dst='10.0.0.1',

tcp\_src=tcp\_pkt.dst\_port,

tcp\_dst=tcp\_pkt.src\_port)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(1)]

self.add\_flow(datapath,5,match,actions)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(2)];

out = parser.OFPPacketOut(datapath, ofproto.OFP\_NO\_BUFFER,

ofproto.OFPP\_CONTROLLER, actions,

msg.data)

datapath.send\_msg(out)

elif datapath.id == 2:

tcp\_rst\_ack = packet.Packet()

# 324

match = parser.OFPMatch(eth\_type=0X0800,ip\_proto=6,

ipv4\_src='10.0.0.1',

ipv4\_dst='10.0.0.2',

tcp\_src=tcp\_pkt.src\_port,

tcp\_dst=tcp\_pkt.dst\_port)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(2)];

self.add\_flow(datapath,5,match,actions)

# 423

match = parser.OFPMatch(eth\_type=0X0800,ip\_proto=6,

ipv4\_src='10.0.0.2',

ipv4\_dst='10.0.0.1',

tcp\_src=tcp\_pkt.dst\_port,

tcp\_dst=tcp\_pkt.src\_port)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(1)]

self.add\_flow(datapath,5,match,actions)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(2)];

out = parser.OFPPacketOut(datapath, ofproto.OFP\_NO\_BUFFER,

ofproto.OFPP\_CONTROLLER, actions,

msg.data)

datapath.send\_msg(out)

elif datapath.id == 3:

if tcp\_pkt.src\_port % 2 == 0:

tcp\_rst\_ack = packet.Packet()

# 314

match = parser.OFPMatch(eth\_type=0X0800,ip\_proto=6,

ipv4\_src='10.0.0.1',

ipv4\_dst='10.0.0.2',

tcp\_src=tcp\_pkt.src\_port,

tcp\_dst=tcp\_pkt.dst\_port)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(2)]

self.add\_flow(datapath,5,match,actions)

# 413

match = parser.OFPMatch(eth\_type=0X0800,ip\_proto=6,

ipv4\_src='10.0.0.2',

ipv4\_dst='10.0.0.1',

tcp\_src=tcp\_pkt.dst\_port,

tcp\_dst=tcp\_pkt.src\_port)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(1)]

self.add\_flow(datapath,5,match,actions)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(2)]

out = parser.OFPPacketOut(datapath, ofproto.OFP\_NO\_BUFFER,

ofproto.OFPP\_CONTROLLER, actions,

msg.data)

datapath.send\_msg(out)

else:

tcp\_rst\_ack = packet.Packet()

# 324

match = parser.OFPMatch(eth\_type=0X0800,ip\_proto=6,

ipv4\_src='10.0.0.1',

ipv4\_dst='10.0.0.2',

tcp\_src=tcp\_pkt.src\_port,

tcp\_dst=tcp\_pkt.dst\_port)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(3)]

self.add\_flow(datapath,5,match,actions)

# 423

match = parser.OFPMatch(eth\_type=0X0800,ip\_proto=6,

ipv4\_src='10.0.0.2',

ipv4\_dst='10.0.0.1',

tcp\_src=tcp\_pkt.dst\_port,

tcp\_dst=tcp\_pkt.src\_port)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(1)]

self.add\_flow(datapath,5,match,actions)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(3)];

out = parser.OFPPacketOut(datapath, ofproto.OFP\_NO\_BUFFER,

ofproto.OFPP\_CONTROLLER, actions,

msg.data)

datapath.send\_msg(out)

elif datapath.id == 4:

if tcp\_pkt.src\_port % 2 == 0:

tcp\_rst\_ack = packet.Packet()

# 314

match = parser.OFPMatch(eth\_type=0X0800,ip\_proto=6,

ipv4\_src='10.0.0.1',

ipv4\_dst='10.0.0.2',

tcp\_src=tcp\_pkt.src\_port,

tcp\_dst=tcp\_pkt.dst\_port)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(1)]

self.add\_flow(datapath,5,match,actions)

# 413

match = parser.OFPMatch(eth\_type=0X0800,ip\_proto=6,

ipv4\_src='10.0.0.2',

ipv4\_dst='10.0.0.1',

tcp\_src=tcp\_pkt.dst\_port,

tcp\_dst=tcp\_pkt.src\_port)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(2)]

self.add\_flow(datapath,5,match,actions)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(1)]

out = parser.OFPPacketOut(datapath, ofproto.OFP\_NO\_BUFFER,

ofproto.OFPP\_CONTROLLER, actions,

msg.data)

datapath.send\_msg(out)

else:

tcp\_rst\_ack = packet.Packet()

# 324

match = parser.OFPMatch(eth\_type=0X0800,ip\_proto=6,

ipv4\_src='10.0.0.1',

ipv4\_dst='10.0.0.2',

tcp\_src=tcp\_pkt.src\_port,

tcp\_dst=tcp\_pkt.dst\_port)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(1)]

self.add\_flow(datapath,5,match,actions)

# 423

match = parser.OFPMatch(eth\_type=0X0800,ip\_proto=6,

ipv4\_src='10.0.0.2',

ipv4\_dst='10.0.0.1',

tcp\_src=tcp\_pkt.dst\_port,

tcp\_dst=tcp\_pkt.src\_port)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(3)]

self.add\_flow(datapath,5,match,actions)

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(1)];

out = parser.OFPPacketOut(datapath, ofproto.OFP\_NO\_BUFFER,

ofproto.OFPP\_CONTROLLER, actions,

msg.data)

datapath.send\_msg(out)

else:

print "wrong switch"

# Member methods you can call to install TCP/UDP/ICMP fwding rules

def add\_layer4\_rules(self, datapath, ip\_proto, ipv4\_dst = None, priority = 1, fwd\_port = None):

parser = datapath.ofproto\_parser

actions = [parser.OFPActionOutput(fwd\_port)]

match = parser.OFPMatch(eth\_type = ether.ETH\_TYPE\_IP,

ip\_proto = ip\_proto,

ipv4\_dst = ipv4\_dst)

self.add\_flow(datapath, priority, match, actions)

# Member methods you can call to install general rules

def add\_flow(self, datapath, priority, match, actions):

ofproto = datapath.ofproto

parser = datapath.ofproto\_parser

inst = [parser.OFPInstructionActions(ofproto.OFPIT\_APPLY\_ACTIONS,

actions)]

mod = parser.OFPFlowMod(datapath=datapath, priority=priority,

match=match, instructions=inst)

datapath.send\_msg(mod)

def handle\_arp(self, datapath, in\_port, pkt):

ofproto = datapath.ofproto

parser = datapath.ofproto\_parser

# parse out the ethernet and arp packet

eth\_pkt = pkt.get\_protocol(ethernet.ethernet)

arp\_pkt = pkt.get\_protocol(arp.arp)

# obtain the MAC of dst IP

arp\_resolv\_mac = self.arp\_table[arp\_pkt.dst\_ip]

### generate the ARP reply msg, please refer RYU documentation

### the packet library section

ether\_hd = ethernet.ethernet(dst = eth\_pkt.src,

src = arp\_resolv\_mac,

ethertype = ether.ETH\_TYPE\_ARP);

arp\_hd = arp.arp(hwtype=1, proto = 2048, hlen = 6, plen = 4,

opcode = 2, src\_mac = arp\_resolv\_mac,

src\_ip = arp\_pkt.dst\_ip, dst\_mac = eth\_pkt.src,

dst\_ip = arp\_pkt.src\_ip);

arp\_reply = packet.Packet()

arp\_reply.add\_protocol(ether\_hd)

arp\_reply.add\_protocol(arp\_hd)

arp\_reply.serialize()

# send the Packet Out mst to back to the host who is initilaizing the ARP

actions = [parser.OFPActionOutput(in\_port)];

out = parser.OFPPacketOut(datapath, ofproto.OFP\_NO\_BUFFER,

ofproto.OFPP\_CONTROLLER, actions,

arp\_reply.data)

datapath.send\_msg(out)

6. Consider the implementation of flow table memory in Open vSwitch. What will happen if the table size is not set while millions of flow entries are inserted into the vSwitch?

Even if the flow table size is not set, the switch still has its maximum memory size. So, the flow table will be

overflowed while millions of flow entries are inserted into vSwitch. In addition, even flow table is so big

that it can keep all the entries, switch might need a long time to find the match.

1. Name at least 3 consequences when flow table gets overflowed.

(1)Controller will keep inserting new table entries but keep receiving error message from switch. Because the controller which will continuously attempt to add new flow entries and overflow the switch's flow table.

(2) Even controller can insert the entries into switch, the switch must delete some old entries. Then next time the flow of this old entries has to ask controller to insert the entry into switch again. In short, the switch has to keep swapping the flow entries.

(3) If flow table gets over owed and controller cannot insert the new entries, the packets of this flow will be dropped.