Lab 6: Network Monitoring with SDN

Sijia Jiang N11642702

# Objectives

* Learn developing SDN traffic monitoring applications.
* Gain real time traffic measurement experience.
* Manage flows in real-time based on traffic measurement.

# Equipment

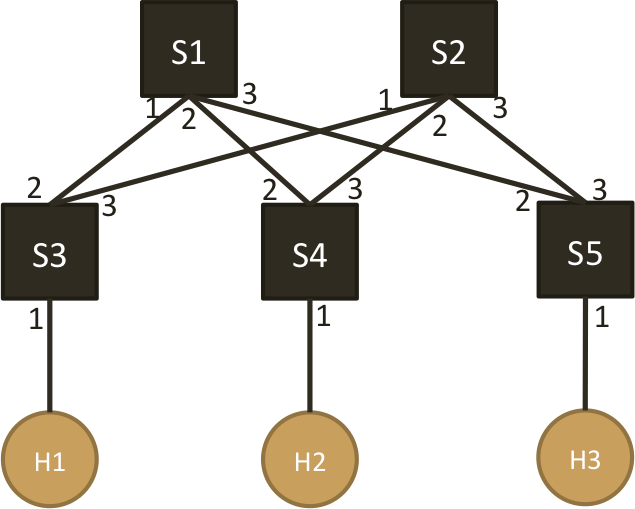
* Computer with Internet Connectivity.

# References

* RYU programming guide: <https://osrg.github.io/ryu-book/en/html/>

# Experiments

* 1. Go through the traffic monitoring example provided by RYU website: <https://osrg.github.io/ryu-book/en/html/traffic_monitor.html>
  2. Create the following topology using the topology file provided on NYU Classes to create 5 switches and 3 hosts. The topology .py file also includes a costumed command: “runTraffic” to use in the Mininet prompt. This command generates UDP traffic among H1, H2 and H3 for 10 mins.



* 1. Write two controllers to,
     1. Controller 1: When a new flow arrives, always manage the flow to follow the paths as instructed below,   
         H1 🡺 H2: S3 – S1 – S4   
         H1 🡺 H3: S3 – S1 – S5  
         H2 🡺 H3: S4 – S1 – S5  
         Measure (every 5 or 10 seconds) the traffic rate (in bps) of each link for 10 mins and draw a figure of ( time vs. traffic rate ) for each link
     2. Controller 2: Use the measured traffic (every 5 or 10 seconds) to manage flows so that all the links in the network is as evenly loaded as possible. Flow management can be done through either arranging new arriving flow’s path or re-locating existing flows. Draw a figure of ( time vs. traffic rate ) for each link.
  2. Start the controller, and run the “runTraffic” command in Mininet prompt to monitor the traffic and manage flows for 10 mins.

# Reports

1. Explain your approach used in Controller 2 to evenly distribute the traffic onto different links.

Write rules in controller. Record bytes for both ports. If the received and transmitted bytes is in port2 is larger than port3, designate packet to out\_port 3, vice versa.

1. Submit your controller codes for both controller 1 and controller 2
2. **controller1**

from operator import attrgetter

from ryu.app import simple\_switch\_13

from ryu.controller import ofp\_event

from ryu.controller.handler import MAIN\_DISPATCHER, DEAD\_DISPATCHER

from ryu.controller.handler import set\_ev\_cls

from ryu.lib import hub

from ryu.base import app\_manager

from ryu.controller.handler import CONFIG\_DISPATCHER, MAIN\_DISPATCHER

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 SimpleMonitor13(app\_manager.RyuApp):

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

super(SimpleMonitor13, self).\_\_init\_\_(\*args, \*\*kwargs)

self.datapaths = {}

self.monitor\_thread = hub.spawn(self.\_monitor)

self.s3\_2 = 0

self.s3\_2\_old = 0

self.s3\_2\_new = 0

self.s3\_3 = 0

self.s3\_3\_old = 0

self.s3\_3\_new = 0

self.s4\_2 = 0

self.s4\_2\_old = 0

self.s4\_2\_new = 0

self.s4\_3 = 0

self.s4\_3\_old = 0

self.s4\_3\_new = 0

self.s5\_2 = 0

self.s5\_2\_old = 0

self.s5\_2\_new = 0

self.s5\_3 = 0

self.s5\_3\_old = 0

self.s5\_3\_new = 0

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"

self.arp\_table["10.0.0.3"]="00:00:00:00:00:03"

@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

#implement TCP fwding

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.1', 10, 1) #tcp to h1

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.2', 10, 2) #tcp to h2

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.3', 10, 3) #tcp to h

elif dpid == 3: # switch S2

#implement TCP fwding

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.1', 10, 1) #tcp to h1

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.2', 10, 2) #tcp to h2

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.3', 10, 2) #tcp to h2

elif dpid == 4: # switch S3

#implement TCP fwding

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.1', 10, 2) #tcp to h1

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.2', 10, 1) #tcp to h2

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.3', 10, 2) #tcp to h1

elif dpid == 5: # switch S4

### implement tcp fwding

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.1', 10, 2) #tcp to h1

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.2', 10, 2) #tcp to h2

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.3', 10, 1) #tcp to h2

@set\_ev\_cls(ofp\_event.EventOFPStateChange,

[MAIN\_DISPATCHER, DEAD\_DISPATCHER])

def \_state\_change\_handler(self, ev):

datapath = ev.datapath

if ev.state == MAIN\_DISPATCHER:

if datapath.id not in self.datapaths:

self.logger.debug('register datapath: %016x', datapath.id)

self.datapaths[datapath.id] = datapath

elif ev.state == DEAD\_DISPATCHER:

if datapath.id in self.datapaths:

self.logger.debug('unregister datapath: %016x', datapath.id)

del self.datapaths[datapath.id]

def \_monitor(self):

while True:

for dp in self.datapaths.values():

self.\_request\_stats(dp)

hub.sleep(10)

def \_request\_stats(self, datapath):

self.logger.debug('send stats request: %016x', datapath.id)

ofproto = datapath.ofproto

parser = datapath.ofproto\_parser

req = parser.OFPFlowStatsRequest(datapath)

datapath.send\_msg(req)

req = parser.OFPPortStatsRequest(datapath, 0, ofproto.OFPP\_ANY)

datapath.send\_msg(req)

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

def \_flow\_stats\_reply\_handler(self, ev):

body = ev.msg.body

self.logger.info('datapath '

'in-port eth-dst '

'out-port packets bytes')

self.logger.info('---------------- '

'-------- ----------------- '

'-------- -------- --------')

for stat in sorted([flow for flow in body if flow.priority == 1],

key=lambda flow: (flow.match['in\_port'],

flow.match['eth\_dst'])):

self.logger.info('%016x %8x %17s %8x %8d %8d',

ev.msg.datapath.id,

stat.match['in\_port'], stat.match['eth\_dst'],

stat.instructions[0].actions[0].port,

stat.packet\_count, stat.byte\_count)

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

def \_port\_stats\_reply\_handler(self, ev):

body = ev.msg.body

self.logger.info('datapath port '

'rx-pkts rx-bytes rx-error '

'tx-pkts tx-bytes tx-error')

self.logger.info('---------------- -------- '

'-------- -------- -------- '

'-------- -------- --------')

s3s1rx = open("s3s1rx.txt", "a")

s3s1tx = open("s3s1tx.txt", "a")

s4s1rx = open("s4s1rx.txt", "a")

s4s1tx = open("s4s1tx.txt", "a")

s5s1rx = open("s5s1rx.txt", "a")

s5s1tx = open("s5s1tx.txt", "a")

s3s2rx = open("s3s2rx.txt", "a")

s3s2tx = open("s3s2tx.txt", "a")

s4s2rx = open("s4s2rx.txt", "a")

s4s2tx = open("s4s2tx.txt", "a")

s5s2rx = open("s5s2rx.txt", "a")

s5s2tx = open("s5s2tx.txt", "a")

for stat in sorted(body, key=attrgetter('port\_no')):

self.logger.info('%016x %8x %8d %8d %8d %8d %8d %8d',

ev.msg.datapath.id, stat.port\_no,

stat.rx\_packets, stat.rx\_bytes, stat.rx\_errors,

stat.tx\_packets, stat.tx\_bytes, stat.tx\_errors)

self.logger.info('reply port')

if ev.msg.datapath.id == 3 and stat.port\_no == 2:

str1 = str(stat.rx\_bytes)

s3s1rx.write(str1+'\n')

str2 = str(stat.tx\_bytes)

s3s1tx.write(str2+'\n')

#for each switch, saving the old record,

#and add received bytes and transmitted bytes to the new one

#let the difference between these two variant to be the determining factor

self.s3\_2\_old = self.s3\_2\_new

self.s3\_2\_new = stat.rx\_bytes + stat.tx\_bytes

self.s3\_2 = self.s3\_2\_new - self.s3\_2\_old

if ev.msg.datapath.id == 4 and stat.port\_no == 2:

str3 = str(stat.rx\_bytes)

s4s1rx.write(str3+'\n')

str4 = str(stat.tx\_bytes)

s4s1tx.write(str4+'\n')

self.s4\_2\_old = self.s4\_2\_new

self.s4\_2\_new = stat.rx\_bytes + stat.tx\_bytes

self.s4\_2 = self.s4\_2\_new - self.s4\_2\_old

if ev.msg.datapath.id == 5 and stat.port\_no == 2:

str5 = str(stat.rx\_bytes)

s5s1rx.write(str5+'\n')

str6 = str(stat.tx\_bytes)

s5s1tx.write(str6+'\n')

self.s5\_2\_old = self.s5\_2\_new

self.s5\_2\_new = stat.rx\_bytes + stat.tx\_bytes

self.s5\_2 = self.s5\_2\_new - self.s5\_2\_old

if ev.msg.datapath.id == 3 and stat.port\_no == 3:

str7 = str(stat.rx\_bytes)

s3s2rx.write(str7+'\n')

str8 = str(stat.tx\_bytes)

s3s2tx.write(str8+'\n')

self.s3\_3\_old = self.s3\_3\_new

self.s3\_3\_new = stat.rx\_bytes + stat.tx\_bytes

self.s3\_3 = self.s3\_3\_new - self.s3\_3\_old

if ev.msg.datapath.id == 4 and stat.port\_no == 3:

str9 = str(stat.rx\_bytes)

s4s2rx.write(str9+'\n')

str10 = str(stat.tx\_bytes)

s4s2tx.write(str10+'\n')

self.s4\_3\_old = self.s4\_3\_new

self.s4\_3\_new = stat.rx\_bytes + stat.tx\_bytes

self.s4\_3 = self.s4\_3\_new - self.s4\_3\_old

if ev.msg.datapath.id == 5 and stat.port\_no == 3:

str11 = str(stat.rx\_bytes)

s5s2rx.write(str11+'\n')

str12 = str(stat.tx\_bytes)

s5s2tx.write(str12+'\n')

self.s5\_3\_old = self.s5\_3\_new

self.s5\_3\_new = stat.rx\_bytes + stat.tx\_bytes

self.s5\_3 = self.s5\_3\_new - self.s5\_3\_old

s3s1rx.close()

s3s1tx.close()

s4s1rx.close()

s4s1tx.close()

s5s1rx.close()

s5s1tx.close()

s3s2rx.close()

s3s2tx.close()

s4s2rx.close()

s4s2tx.close()

s5s2rx.close()

s5s2tx.close()

@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

# process IP

# 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]

ether\_hd = ethernet.ethernet(dst = eth\_pkt.src,src = arp\_resolv\_mac,ethertype = ether.ETH\_TYPE\_ARP);

arp\_hd = arp.arp(hwtype=1, proto=0x0800, hlen=6, plen=4, opcode=2,src\_mac=arp\_resolv\_mac, src\_ip=arp\_pkt.dst\_ip,dst\_mac=arp\_pkt.src\_mac, 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)

**2) controller2**

from ryu.base import app\_manager

from ryu.controller import ofp\_event

from ryu.controller.handler import CONFIG\_DISPATCHER, MAIN\_DISPATCHER, DEAD\_DISPATCHER

from ryu.controller.handler import set\_ev\_cls

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

from operator import attrgetter

from ryu.lib import hub

from ryu.ofproto import ofproto\_v1\_3

class Lab4ryu(app\_manager.RyuApp):

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

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

super(Lab4ryu, self).\_\_init\_\_(\*args, \*\*kwargs)

self.datapaths = {}

self.monitor\_thread = hub.spawn(self.\_monitor)

# record of previous and new flow to determine transfer route

self.s3\_2 = 0

self.s3\_2\_old = 0

self.s3\_2\_new = 0

self.s3\_3 = 0

self.s3\_3\_old = 0

self.s3\_3\_new = 0

self.s4\_2 = 0

self.s4\_2\_old = 0

self.s4\_2\_new = 0

self.s4\_3 = 0

self.s4\_3\_old = 0

self.s4\_3\_new = 0

self.s5\_2 = 0

self.s5\_2\_old = 0

self.s5\_2\_new = 0

self.s5\_3 = 0

self.s5\_3\_old = 0

self.s5\_3\_new = 0

# 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";

self.arp\_table["10.0.0.3"] = "00:00:00:00:00:03";

@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

match = parser.OFPMatch()

actions = [parser.OFPActionOutput(ofproto.OFPP\_CONTROLLER,ofproto.OFPCML\_NO\_BUFFER)]

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

dpid = datapath.id

if dpid == 1:

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

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

self.add\_layer4\_rules(datapath, inet.IPPROTO\_TCP, '10.0.0.3', 10, 3)

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\_ICMP, '10.0.0.3', 10, 3)

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

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

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.3', 10, 3)

elif dpid == 2:

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

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

self.add\_layer4\_rules(datapath, inet.IPPROTO\_TCP, '10.0.0.3', 10, 3)

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\_ICMP, '10.0.0.3', 10, 3)

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

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

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.3', 10, 3)

# delete the output port information for the edge switchs

elif dpid == 3:

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

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

self.add\_layer4\_rules(datapath, inet.IPPROTO\_TCP, '10.0.0.3', 10, 2)

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\_ICMP, '10.0.0.3', 10, 2)

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

elif dpid == 4:

self.add\_layer4\_rules(datapath, inet.IPPROTO\_TCP, '10.0.0.1', 10, 2)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_TCP, '10.0.0.2', 10, 1)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_TCP, '10.0.0.3', 10, 2)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_ICMP, '10.0.0.1', 10, 2)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_ICMP, '10.0.0.2', 10, 1)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_ICMP, '10.0.0.3', 10, 2)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.2', 10, 1)

elif dpid == 5:

self.add\_layer4\_rules(datapath, inet.IPPROTO\_TCP, '10.0.0.1', 10, 2)

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

self.add\_layer4\_rules(datapath, inet.IPPROTO\_TCP, '10.0.0.3', 10, 1)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_ICMP, '10.0.0.1', 10, 2)

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

self.add\_layer4\_rules(datapath, inet.IPPROTO\_ICMP, '10.0.0.3', 10, 1)

self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.3', 10, 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

# process IP

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

self.handle\_ip(datapath, in\_port, pkt, msg)

return

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)

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

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

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

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

ether\_hd = ethernet.ethernet(dst = eth\_pkt.src,src = arp\_resolv\_mac,ethertype = ether.ETH\_TYPE\_ARP)

arp\_hd = arp.arp(opcode=arp.ARP\_REPLY,src\_mac=arp\_resolv\_mac,dst\_mac=arp\_pkt.src\_mac,src\_ip=arp\_pkt.dst\_ip,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()

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

out = parser.OFPPacketOut(datapath, ofproto.OFP\_NO\_BUFFER,ofproto.OFPP\_CONTROLLER, actions,arp\_reply.data)

datapath.send\_msg(out)

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

ofproto = datapath.ofproto

parser = datapath.ofproto\_parser

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

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

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

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

actions = []

if ipv4\_pkt.proto == inet.IPPROTO\_UDP:

match = parser.OFPMatch(eth\_type=0x0800,

ip\_proto=17,

ipv4\_src=ipv4\_pkt.src,

ipv4\_dst=ipv4\_pkt.dst,

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

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

output2 = [parser.OFPActionOutput(2)]

output3 = [parser.OFPActionOutput(3)]

# compare the increment of two output link, and add flow into the smaller one

if datapath.id == 3:

if self.s3\_2 < self.s3\_3:

self.add\_flow(datapath, 20, match, output2)

actions = [parser.OFPActionOutput(2)]

else:

self.add\_flow(datapath, 20, match, output3)

actions = [parser.OFPActionOutput(3)]

if datapath.id == 4:

if self.s4\_2 < self.s4\_3:

self.add\_flow(datapath, 20, match, output2)

actions = [parser.OFPActionOutput(2)]

else:

self.add\_flow(datapath, 20, match, output3)

actions = [parser.OFPActionOutput(3)]

if datapath.id == 5:

if self.s5\_2 < self.s5\_3:

self.add\_flow(datapath, 20, match, output2)

actions = [parser.OFPActionOutput(2)]

else:

self.add\_flow(datapath, 20, match, output3)

actions = [parser.OFPActionOutput(3)]

out = parser.OFPPacketOut(datapath, ofproto.OFP\_NO\_BUFFER, in\_port, actions, msg.data)

datapath.send\_msg(out)

@set\_ev\_cls(ofp\_event.EventOFPStateChange,

[MAIN\_DISPATCHER, DEAD\_DISPATCHER])

def \_state\_change\_handler(self, ev):

datapath = ev.datapath

if ev.state == MAIN\_DISPATCHER:

if datapath.id not in self.datapaths:

self.logger.debug('register datapath: %016x', datapath.id)

self.datapaths[datapath.id] = datapath

elif ev.state == DEAD\_DISPATHER:

if datapath.id in self.datapaths:

self.logger.debug('unregister datapath: %016x', datapath.id)

del self.datapaths[datapath.id]

def \_monitor(self):

while True:

for dp in self.datapaths.values():

self.\_request\_states(dp)

hub.sleep(10)

def \_request\_states(self, datapath):

self.logger.debug('send stats request: %016x', datapath.id)

ofproto = datapath.ofproto

parser = datapath.ofproto\_parser

#req = parser.OFPFlowStatsRequest(datapath)

#datapath.send\_msg(req)

req = parser.OFPPortStatsRequest(datapath, 0, ofproto.OFPP\_ANY)

datapath.send\_msg(req)

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

def \_table\_stats\_reply\_handler(self, ev):

body = ev.msg.body

self.logger.info('datapath '

'in-port eth-dst '

'out-port packets bytes')

self.logger.info('---------------- '

'-------- ----------------- '

'-------- -------- --------')

for stat in sorted([flow for flow in body if flow.priority == 10],

key=lambda flow: (flow.match['in\_port'],

flow.match['eth\_dst'])):

self.logger.info('%016x %8x %17s %8x %8d %8d',

ev.msg.datapath.id,

stat.match['in\_port'], stat.match['eth\_dst'],

stat.instructions[0].actions[0].port,

stat.packet\_count, stat.byte\_count)

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

def \_port\_stats\_reply\_handler(self, ev):

body = ev.msg.body

self.logger.info('datapath port '

'rx-pkts rx-bytes rx-error '

'tx-pkts tx-bytes tx-error')

self.logger.info('---------------- -------- '

'-------- -------- -------- '

'-------- -------- --------')

s3s1rx = open("s3s1rx.txt", "a")

s3s1tx = open("s3s1tx.txt", "a")

s4s1rx = open("s4s1rx.txt", "a")

s4s1tx = open("s4s1tx.txt", "a")

s5s1rx = open("s5s1rx.txt", "a")

s5s1tx = open("s5s1tx.txt", "a")

s3s2rx = open("s3s2rx.txt", "a")

s3s2tx = open("s3s2tx.txt", "a")

s4s2rx = open("s4s2rx.txt", "a")

s4s2tx = open("s4s2tx.txt", "a")

s5s2rx = open("s5s2rx.txt", "a")

s5s2tx = open("s5s2tx.txt", "a")

for stat in sorted(body, key=attrgetter('port\_no')):

self.logger.info('%016x %8x %8d %8d %8d %8d %8d %8d',

ev.msg.datapath.id, stat.port\_no,

stat.rx\_packets, stat.rx\_bytes, stat.rx\_errors,

stat.tx\_packets, stat.tx\_bytes, stat.tx\_errors)

self.logger.info('reply port')

if ev.msg.datapath.id == 3 and stat.port\_no == 2:

str1 = str(stat.rx\_bytes)

s3s1rx.write(str1+'\n')

str2 = str(stat.tx\_bytes)

s3s1tx.write(str2+'\n')

#for each switch, saving the old record,

#and add received bytes and transmitted bytes to the new one

#let the difference between these two variant to be the determining factor

self.s3\_2\_old = self.s3\_2\_new

self.s3\_2\_new = stat.rx\_bytes + stat.tx\_bytes

self.s3\_2 = self.s3\_2\_new - self.s3\_2\_old

if ev.msg.datapath.id == 4 and stat.port\_no == 2:

str3 = str(stat.rx\_bytes)

s4s1rx.write(str3+'\n')

str4 = str(stat.tx\_bytes)

s4s1tx.write(str4+'\n')

self.s4\_2\_old = self.s4\_2\_new

self.s4\_2\_new = stat.rx\_bytes + stat.tx\_bytes

self.s4\_2 = self.s4\_2\_new - self.s4\_2\_old

if ev.msg.datapath.id == 5 and stat.port\_no == 2:

str5 = str(stat.rx\_bytes)

s5s1rx.write(str5+'\n')

str6 = str(stat.tx\_bytes)

s5s1tx.write(str6+'\n')

self.s5\_2\_old = self.s5\_2\_new

self.s5\_2\_new = stat.rx\_bytes + stat.tx\_bytes

self.s5\_2 = self.s5\_2\_new - self.s5\_2\_old

if ev.msg.datapath.id == 3 and stat.port\_no == 3:

str7 = str(stat.rx\_bytes)

s3s2rx.write(str7+'\n')

str8 = str(stat.tx\_bytes)

s3s2tx.write(str8+'\n')

self.s3\_3\_old = self.s3\_3\_new

self.s3\_3\_new = stat.rx\_bytes + stat.tx\_bytes

self.s3\_3 = self.s3\_3\_new - self.s3\_3\_old

if ev.msg.datapath.id == 4 and stat.port\_no == 3:

str9 = str(stat.rx\_bytes)

s4s2rx.write(str9+'\n')

str10 = str(stat.tx\_bytes)

s4s2tx.write(str10+'\n')

self.s4\_3\_old = self.s4\_3\_new

self.s4\_3\_new = stat.rx\_bytes + stat.tx\_bytes

self.s4\_3 = self.s4\_3\_new - self.s4\_3\_old

if ev.msg.datapath.id == 5 and stat.port\_no == 3:

str11 = str(stat.rx\_bytes)

s5s2rx.write(str11+'\n')

str12 = str(stat.tx\_bytes)

s5s2tx.write(str12+'\n')

self.s5\_3\_old = self.s5\_3\_new

self.s5\_3\_new = stat.rx\_bytes + stat.tx\_bytes

self.s5\_3 = self.s5\_3\_new - self.s5\_3\_old

s3s1rx.close()

s3s1tx.close()

s4s1rx.close()

s4s1tx.close()

s5s1rx.close()

s5s1tx.close()

s3s2rx.close()

s3s2tx.close()

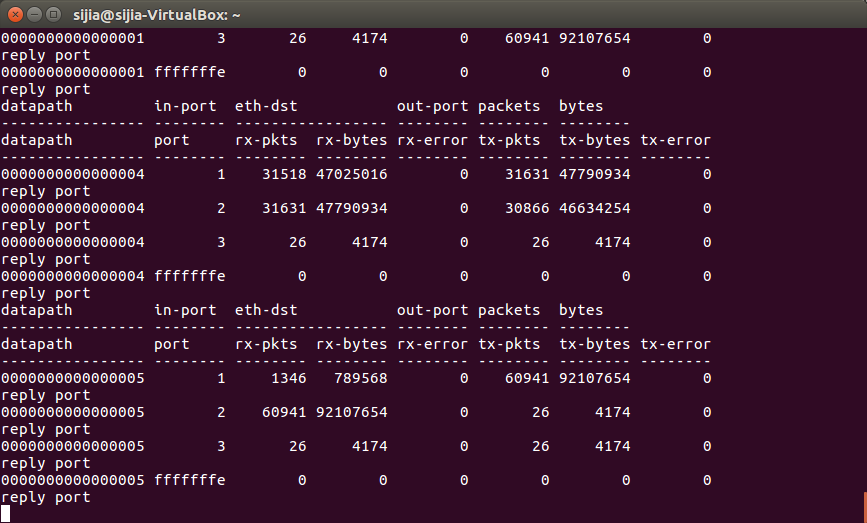
s4s2rx.close()

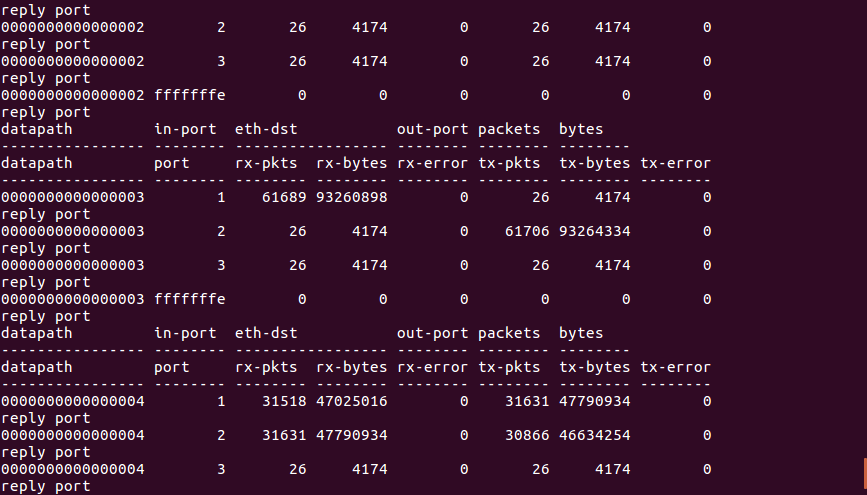
s4s2tx.close()

s5s2rx.close()

s5s2tx.close()

1. Figures of **( Time vs. Traffic Rate (bps) )** on each link over 10 mins from both controller 1 and controller 2
2. **controller 1**



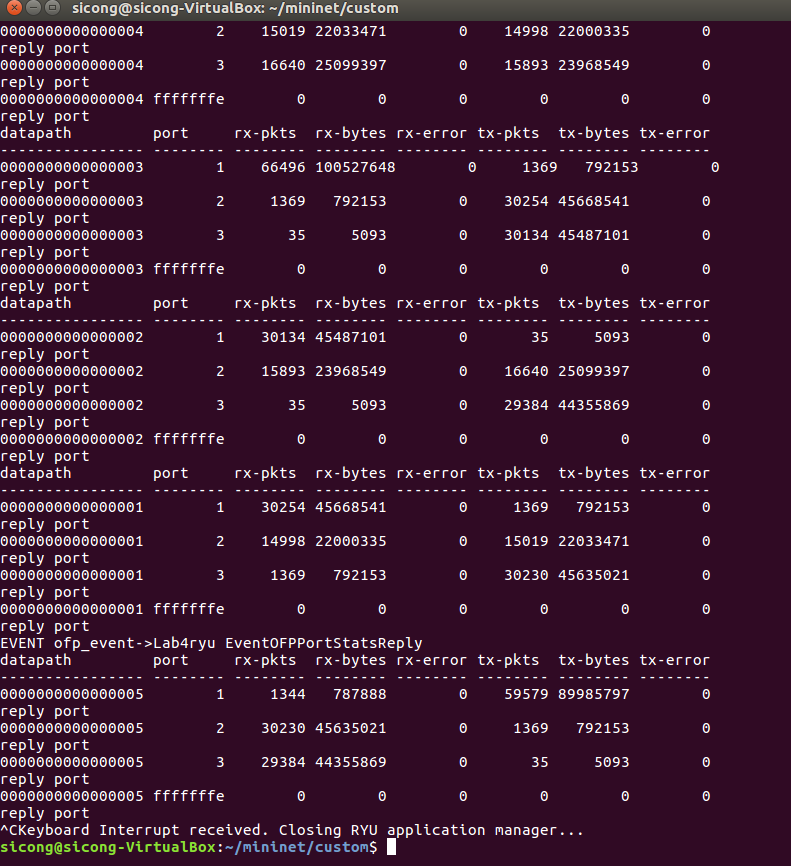


All flow use port2, the rate goes up as traffic start to enter and goes down as traffic go away. The rate is roughly constant in the middle. There is almost no flow goes through port3.

All flow use port2, the rate goes up as traffic start to enter and goes down as traffic go away. The rate is roughly constant in the middle. There is almost no flow goes through port3.

All flow use port2, the rate goes up as traffic start to enter and goes down as traffic go away. The rate is roughly constant in the middle. There is almost no flow goes through port3

1. **controller 2**



Both ports have alternative peaks and valleys in rate-time chart. When port2 reach peak, port3 reach valley. When port2 reach valley, port3 reach peak. Vice versa.

Both ports have alternative peaks and valleys in rate-time chart. When port2 reach peak, port3 reach valley. When port2 reach valley, port3 reach peak. Vice versa.

Both ports have alternative peaks and valleys in rate-time chart. When port2 reach peak, port3 reach valley. When port2 reach valley, port3 reach peak. Vice versa.

**We have zero tolerance to forged or fabricated data!!** A single piece of forged/fabricated data would bring the total score down to zero.