Lab4: SDN Open Virtual Switches

Sicong Chen N10969883

Sijia Jiang N11642702

# Objectives

* Emulate a functional SDN network.
* Understand and get familiar with OVS.
* Understand and get familiar with controllers.

# Equipment

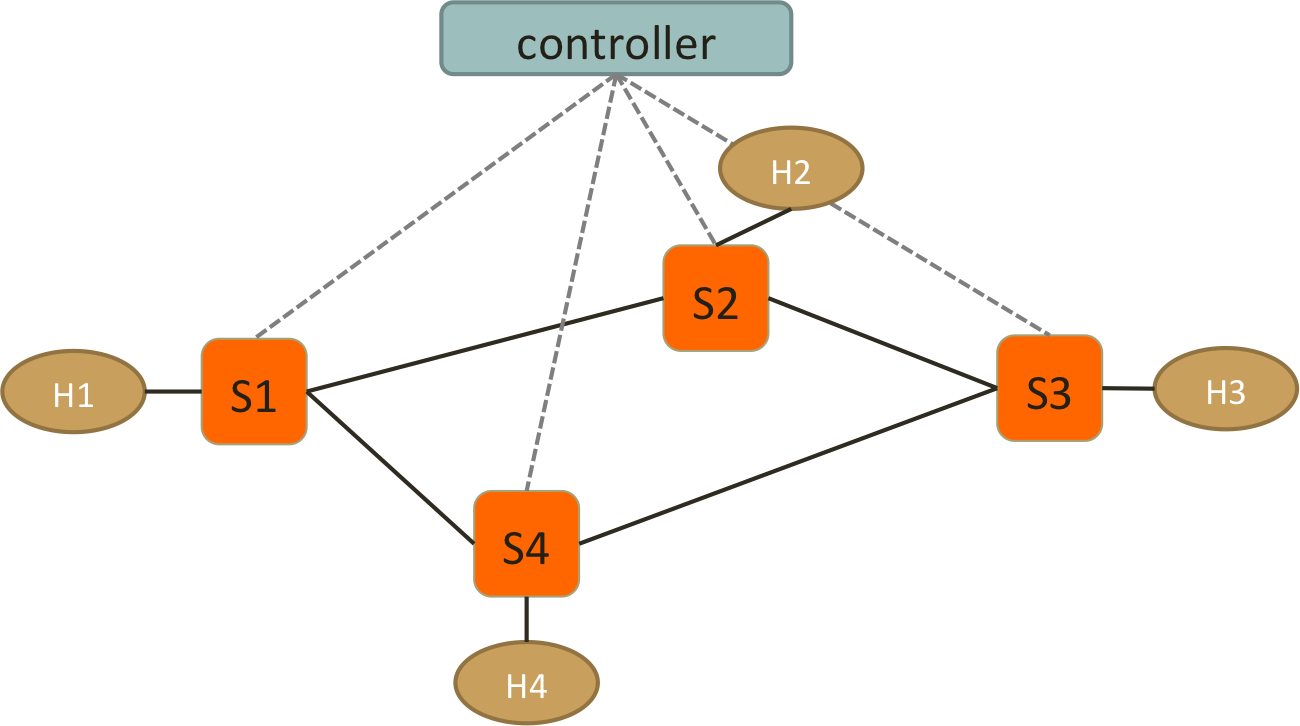
* Computers
* Internet

# References

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

# Experiments

* 1. Use Mininet to create the following topology: (4 Hosts, 4 OVSes ) with a remote controller
  2. Use RYU to implement the controller (you can use other controller such as BEACON, POX, etc...)



* 1. Test Connectivity using ping. (Hint: take care of ARP packets in the controller and install proper rules for them.)
  2. Enforce these policies:
* Everything follows shortest path
* When there are two shortest paths available
  + ICMP and TCP packets take the upper/right path
    - S1-S2-S3 and S2-S3-S4
  + UDP packets take the lower/left path
    - S1-S4-S3 and S2-S1-S4
  + H1 and H3 cannot have HTTP traffic (TCP with port:80)
    - New connections are dropped with a TCP RST sent back to H1 or H3
    - To be more specific, when the first TCP packet (SYN) arrives S1 or S3, forwarded it to controller, controller then create a RST packet and send it back to the host.
  + H2 and H3 cannot have UDP traffic
    - simply drop packets at switches

# Reports

1. Write a pseudo code to implement spanning tree in SDN network.

1. T (the final spanning tree) is defined to be the empty set;

2. For each vertex v of G, make the empty set out of v;

3. Sort the edges of G in ascending (non-decreasing) order;

4. For each edge (u, v) from the sored list of step 3.

If u and v belong to different sets

Add (u,v) to T;

Get together u and v in one single set;

5. Return T

1. List the advantages of using OpenVSwitch and SDN controller compared to IP network.
2. Centralized network provisioning.
3. Holistic enterprise management.
4. More granular security.
5. Lower operating costs.
6. Hardware savings and reduced capital expenditures.
7. Cloud abstraction.
8. Guaranteed content delivery.
9. Include the controller’s code.

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={}

        ### fill in the table for arp searching

        ###

        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'

        self.arp\_table['10.0.0.4'] = '00:00:00:00:00:04'

    """

        Hand-shake event call back method

        This is the very initial method where the switch hand shake with the controller

        It checks whether both are using the same protocol version: OpenFlow 1.3 in this case

        Therefore in this method, you can setup some static rules.

        e.g. the rules which sends unknown packets to the controller

             the rules directing TCP/UDP/ICMP traffic

             ACL rules

    """

    @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](http://datapath.id/)  # classifying the switch ID

        if dpid == 1: # switch S1

            ### implement tcp fwding   hint: please use the add\_layer4\_rules() methods

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

            #10.0.0.1 is dst ip address, 1 is switch port number

            # for switch 1, what port number should choose to forward that dst address

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

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

            self.add\_layer4\_rules(datapath, inet.IPPROTO\_TCP, '10.0.0.3', 20, 2)

            ### implement icmp fwding

            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)

            ### implement udp fwding1

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

            self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.4', 20, 3)

            self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.3', 20, 3)

            ### implement udp fwding2

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

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

            self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.4', 20, 3)

            # implement ACL rules

            # this rule directs the TCP packets from h1 to h3 to the controller

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

                                    ipv4\_src = '10.0.0.1',

                                    ipv4\_dst = '10.0.0.3',

                                    ip\_proto = inet.IPPROTO\_TCP,

                                    tcp\_src = 80)

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

                                              ofproto.OFPCML\_NO\_BUFFER)]

            self.add\_flow(datapath, 30, match, actions)

        elif dpid == 2: # switch S2

            ### implement tcp fwding1

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

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

            self.add\_layer4\_rules(datapath, inet.IPPROTO\_TCP, '10.0.0.3', 20, 3)

            ### implement tcp fwding2

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

            self.add\_layer4\_rules(datapath, inet.IPPROTO\_TCP, '10.0.0.3', 20, 3)

            self.add\_layer4\_rules(datapath, inet.IPPROTO\_TCP, '10.0.0.4', 20, 3)

            ### implement icmp fwding1

            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)

            ### implement icmp fwding2

            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\_ICMP, '10.0.0.4', 10, 3)

            ### implement udp fwding

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

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

            self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.4', 20, 1)

            ### implement ACL

            ### drop UDP from 10.0.0.2 to 10.0.0.3 without notification

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

                                    ipv4\_src = '10.0.0.2',

                                    ipv4\_dst = '10.0.0.3',

                                    ip\_proto = inet.IPPROTO\_UDP)

            actions = []

            self.add\_flow(datapath, 20, match, actions)  #add a flow to controller

        elif dpid == 3: # switch S3

            # fwding everthing between port 1 and port 2

            ### implement tcp fwding1

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

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

            self.add\_layer4\_rules(datapath, inet.IPPROTO\_TCP, '10.0.0.3', 20, 3)

            ### implement tcp fwding2

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

            self.add\_layer4\_rules(datapath, inet.IPPROTO\_TCP, '10.0.0.3', 20, 3)

            self.add\_layer4\_rules(datapath, inet.IPPROTO\_TCP, '10.0.0.4', 20, 2)

            ### implement icmp fwding1

            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, 1)

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

            ### implement icmp fwding2

            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, 3)

            self.add\_layer4\_rules(datapath, inet.IPPROTO\_ICMP, '10.0.0.4', 10, 2)

            ### implement udp fwding

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

            self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.4', 20, 2)

            self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.3', 20, 3)

            # implement ACL rules

            # this rule directs the TCP packets from h1 to h3 to the controller

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

                                    ipv4\_src = '10.0.0.3',

                                    ipv4\_dst = '10.0.0.1',

                                    ip\_proto = inet.IPPROTO\_TCP,

                                    tcp\_src = 80)

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

                                              ofproto.OFPCML\_NO\_BUFFER)]

            self.add\_flow(datapath, 30, match, actions)

            ### implement ACL

            ### drop UDP from 10.0.0.3 to 10.0.0.2 without notification

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

                                    ipv4\_src = '10.0.0.3',

                                    ipv4\_dst = '10.0.0.2',

                                    ip\_proto = inet.IPPROTO\_UDP)

            actions = []

            self.add\_flow(datapath, 20, match, actions)  #add a flow to controller

        elif dpid == 4: # switch S4

            ### implement tcp fwding

            self.add\_layer4\_rules(datapath, inet.IPPROTO\_TCP, '10.0.0.2', 20, 3)

            self.add\_layer4\_rules(datapath, inet.IPPROTO\_TCP, '10.0.0.3', 20, 3)

            self.add\_layer4\_rules(datapath, inet.IPPROTO\_TCP, '10.0.0.4', 20, 2)

            ### implement icmp fwding

            self.add\_layer4\_rules(datapath, inet.IPPROTO\_ICMP, '10.0.0.2', 10, 3)

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

            self.add\_layer4\_rules(datapath, inet.IPPROTO\_ICMP, '10.0.0.4', 10, 2)

            ### implement udp fwding1

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

            self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.4', 20, 2)

            self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.3', 20, 3)

            ### implement udp fwding2

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

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

            self.add\_layer4\_rules(datapath, inet.IPPROTO\_UDP, '10.0.0.4', 20, 2)

        else:

            print "wrong switch"

    """

        Call back method for PacketIn Message

        This is the call back method when a PacketIn Msg is sent

        from a switch to the controller

        It handles L3 classification in this function:

    """

    @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)

            return

    # 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)

    """

        Methods to handle ARP.

    """

    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]

    new\_packet = packet.Packet()

    new\_packet.add\_protocol(ethernet.ethernet(ethertype=eth\_pkt.ethertype, dst=eth\_pkt.src, src=arp\_resolv\_mac))

    new\_packet.add\_protocol(arp.arp(opcode=arp.ARP\_REPLY, src\_mac=arp\_resolv\_mac, src\_ip=arp\_pkt.dst\_ip, dst\_mac=arp\_pkt.src\_mac, dst\_ip=arp\_pkt.src\_ip))

    new\_packet.serialize()

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

    out = parser.OFPPacketOut(datapath,

    ofproto.OFP\_NO\_BUFFER, ofproto.OFPP\_CONTROLLER,

    actions, new\_packet.data)

    datapath.send\_msg(out)

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

        ofproto = datapath.ofproto

        parser = datapath.ofproto\_parser

        ipv4\_pkt = pkt.get\_protocol(ipv4.ipv4) # parse out the IPv4 pkt

        if [datapath.id](http://datapath.id/) == 1 and ipv4\_pkt.proto == inet.IPPROTO\_TCP:

            tcp\_pkt = ipv4\_pkt.get\_protocol(tcp.tcp) # parser out the TCP pkt

            ### generate the TCP packet with the RST flag set to 1

            ### packet generation is similar to ARP,

            ### but you need to generate ethernet->ip->tcp and serialize it

            eth\_pkt = ipv4\_dst.get\_protocol(ethernet, ethernet)

            tcp\_hd = tcp.tcp(ack = tcp\_pkt.seq + 1, src\_port = tcp\_pkt.dst\_port, dst\_port = tcp\_pkt.src\_port, bits = 20)

            ip\_hd = ipv4.ipv4(dst = ipv4\_pkt.src, src = ipv4\_pkt.dst, proto = ipv4\_pkt.proto)

            ether\_hd = ethernet.ethernet(ethertype = ether.ETH\_TYPE\_IP, dst = eth\_pkt.src, src = eth\_pkt.dst)

            tcp\_rst\_ack = packet.Packet()

            tcp\_rst\_ack.add\_protocol(ether\_hd)

            tcp\_rst\_ack.add\_protocol(ip\_hd)

            tcp\_rst\_ack.add\_protocol(tcp\_hd)

            tcp\_rst\_ack.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,

                                      tcp\_rst\_ack.data)

            datapath.send\_msg(out)

1. Include the topology file

from mininet.topo import Topo

class MyTopo( Topo ):

"Simple topology example."

def \_\_init\_\_( self ):

"Create custom topo."

# Initialize topology

Topo.\_\_init\_\_( self )

# Add hosts and switches

H1 = self.addHost('H1')

H2 = self.addHost('H2')

H3 = self.addHost('H3')

H4 = self.addHost('H4')

S1 = self.addSwitch('S1')

S2 = self.addSwitch('S2')

S3 = self.addSwitch('S3')

S4 = self.addSwitch('S4')

S5 = self.addSwitch('S5')

# Add links

self.addLink(H1, S1)

self.addLink(S1, S2)

self.addLink(S1, S4)

self.addLink(S2, H2)

self.addLink(S2, S3)

self.addLink(S4, H4)

self.addLink(S4, S3)

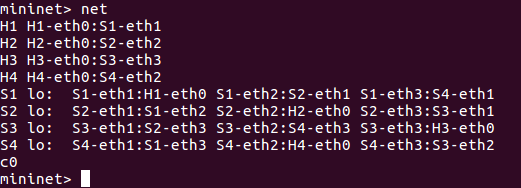
self.addLink(S3, H3)

topos = { 'mytopo': ( lambda: MyTopo() ) }

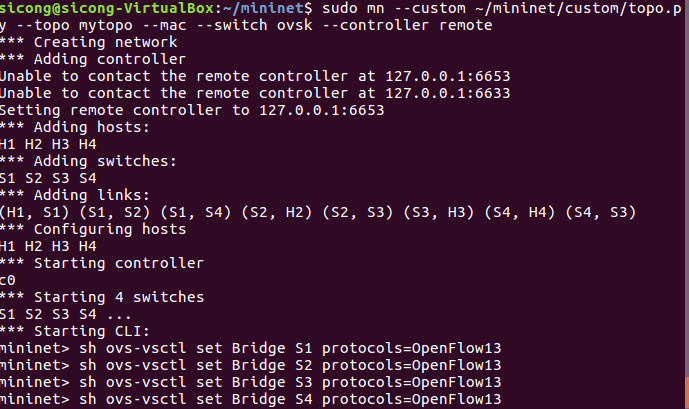
1. Describe how you generate traffic to test your controller and switch behavior: To test your controller-based Ethernet switch, first verify that when all packets arrive at the controller, only broadcast packets (like ARPs) and packets with unknown destination locations (like the first packet sent for a flow) go out all non-input ports. You can do this with tcpdump running on an xterm for each host. Once the switch no longer has hub behavior, work to push down a flow when the source and destination ports are known. You can use ovs-ofctl to verify the flow counters, and if subsequent pings complete much faster, you'll know that they're not passing through the controller. You can also verify this behavior by running iperf in Mininet and checking that no OpenFlow packet-in messages are getting sent. The reported iperf bandwidth should be much higher as well, and should match the number you got when using the reference learning switch controller earlier.
2. Screenshots:

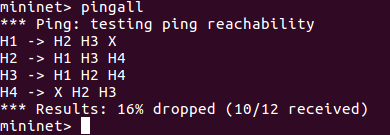
* Ping among all the hosts after setting up the platform.

Screenshots: first, creating networks

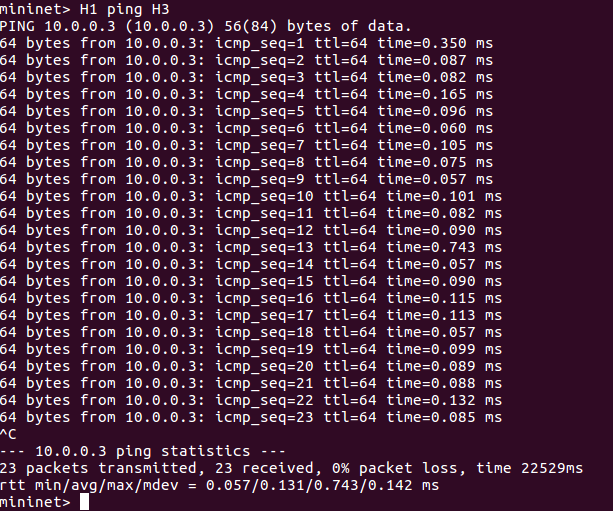


pingall

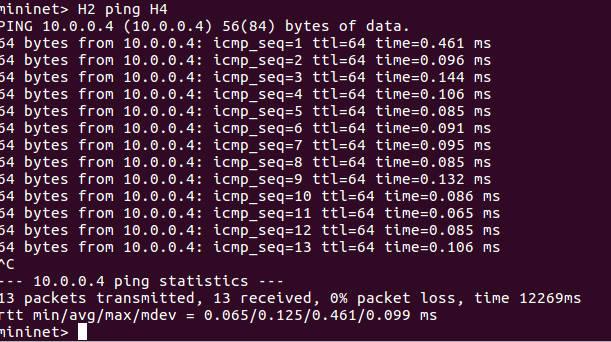




Ping H1 H3

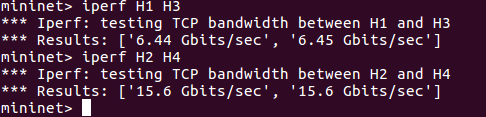


Ping H2 H4

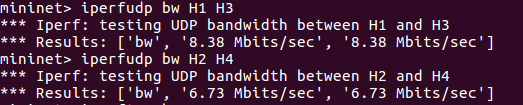


* TCP, UDP and ICMP packets on their respective paths.

Test TCP /ICMP bandwidth between H1 H3 and H2 H4

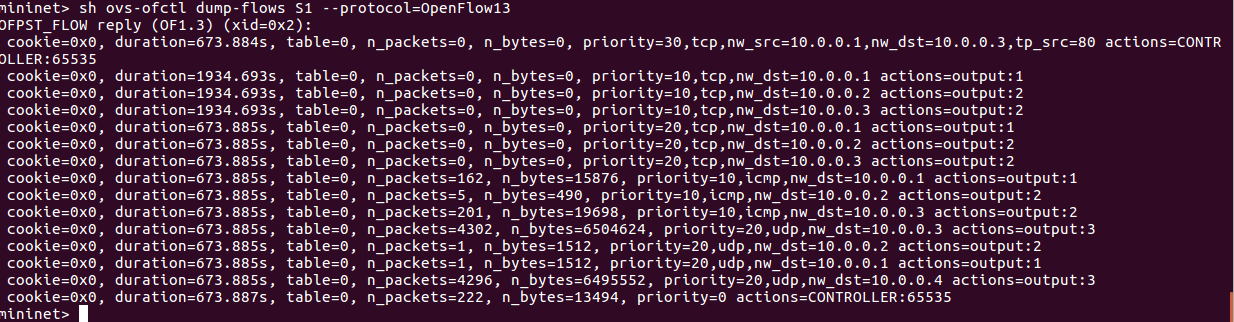


Test UDP bandwidth between H1 H3 and H2 H4

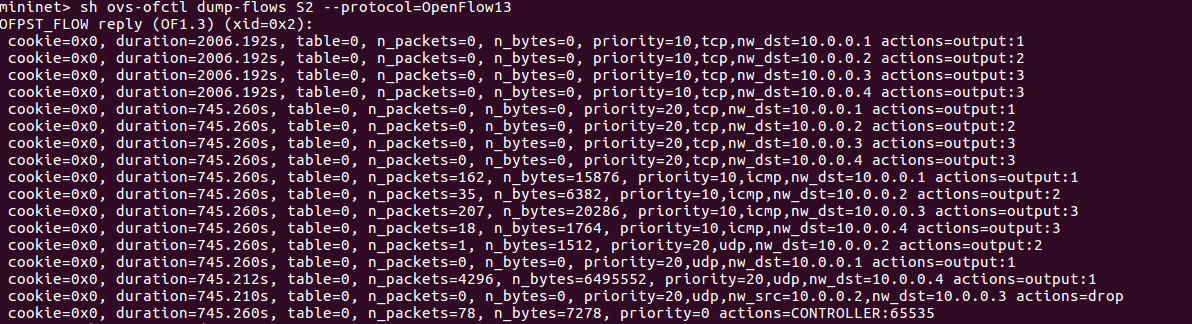


* Rules installed at each switch.

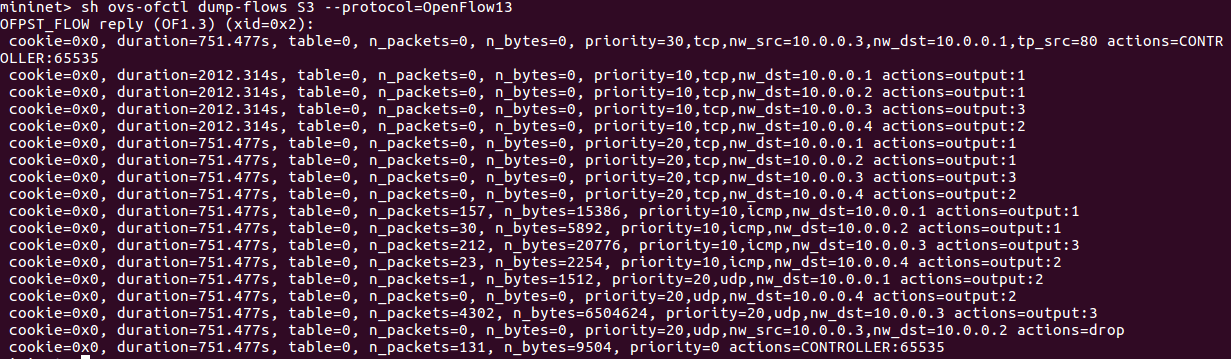
For Switch1:



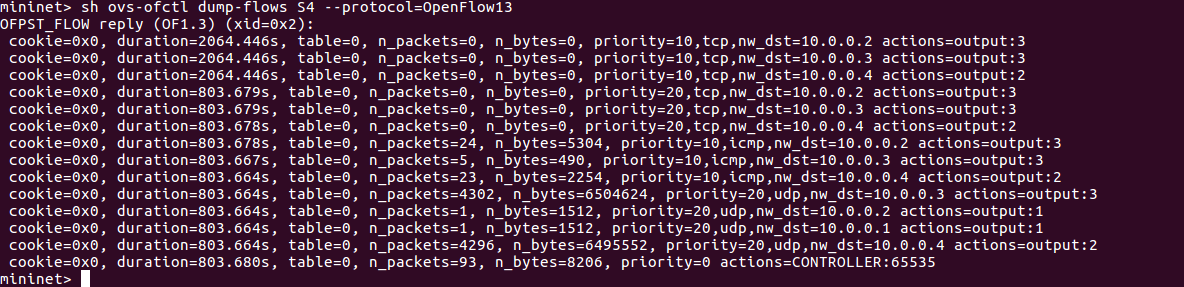
For Switch2:



For Switch3:



For Switch4:



1. Challenges you’ve encountered while doing this experiment, and explain how you manage to solve them. If you do not experience any problem, simply say no problems.

1.We first use mininet on macbook and use terminal to ssh to mininet. However, when we are ssh to mininet , the network setting for mininet is host-only. So we can not ‘sudo apt-get install ‘ unless we log out mininet and disable host-only. Therefore, for the sake of convenience, we run mininet on Ubuntu instead.

2. When we first run ryu to start controller, after ‘ryu-manager –verbose ‘, mininet reply controller.py syntax error. After checking code , we notice error in the way we express arp\_table dictionary syntax.

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