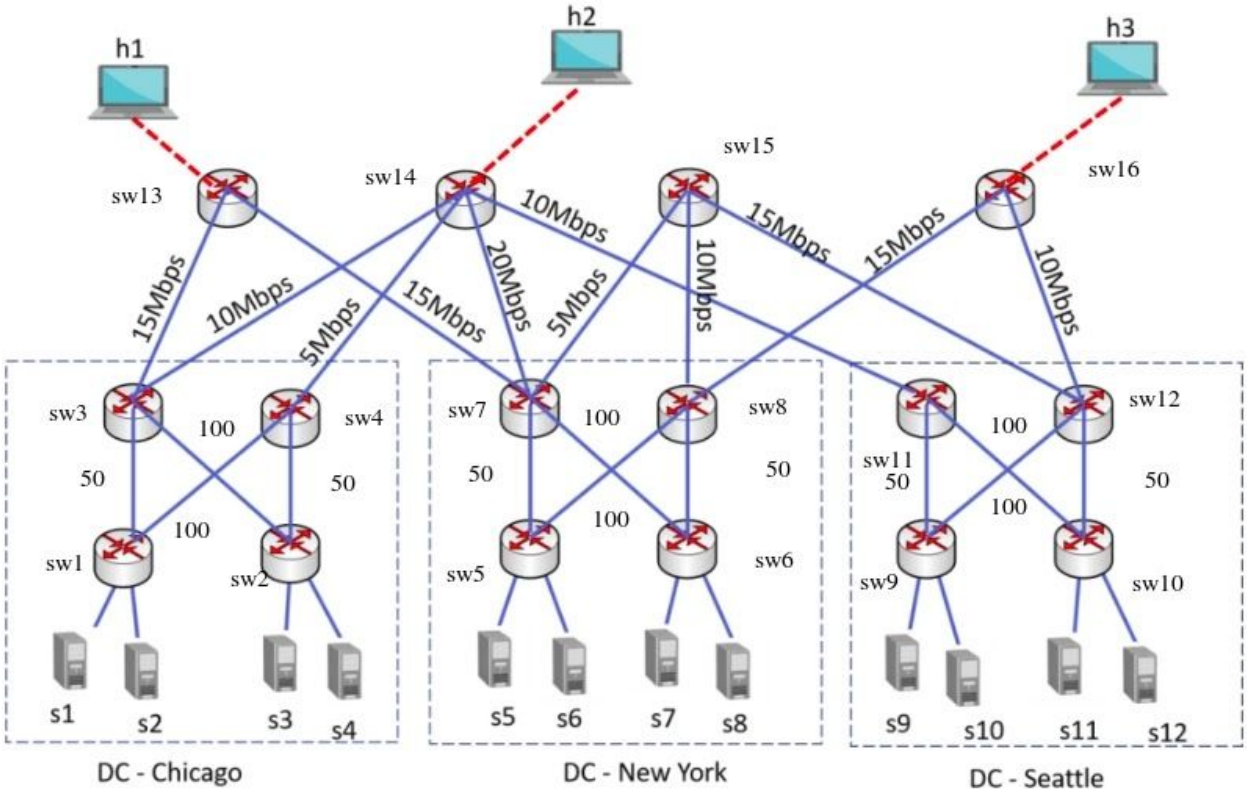
پروژه شماره 4 شبکه های کامپیوتری

(بعد از اصلاحیه)

محسن فیاض 810196650 ژیوار صورتی 810196502

SDN - Ryu



مراحل پیاده‌سازی

❖ نصب

در ابتدا با دستورات زیر نصب را شروع کردیم.

```
Cd
pip install eventlet msgpack-python netaddr oslo.config routes six
webobgit
clone git://github.com/osrg/ryu.gitcd
ryupython ./setup.py install
```

سپس دستورات تست را خواستیم اجرا کنیم که به مشکلات متعدد خوردیم و شروع به نصب نیازمندی هایی که ذکر نشده بود کردیم.

```
ERROR: ryu 4.34 requires
eventlet!=0.18.3,!=0.20.1,!=0.21.0,!=0.23.0,>=0.18.2, which is not
installed.
ERROR: ryu 4.34 requires msgpack<1.0.0,>=0.3.0, which is not installed.
ERROR: ryu 4.34 requires netaddr, which is not installed.
ERROR: ryu 4.34 requires oslo.config>=2.5.0, which is not installed.
ERROR: ryu 4.34 requires ovs>=2.6.0, which is not installed.
ERROR: ryu 4.34 requires routes, which is not installed.
ERROR: ryu 4.34 requires tinyrpc==0.9.4, which is not installed.
```

سپس برای نصب mininet از گیت فایل ها را گرفتیم و با کد های زیر نصبش کردیم.

```
util/install.sh -fnv
```

❖ پیاده سازی ساختار شبکه

```
net = Mininet(topo=None, ipBase='10.0.0.0/8')
```

ابتدا با خط بالا یک Mininet ایجاد کردیم.

```
c0 = net.addController(name='c0', controller=RemoteController,
                       ip='127.0.0.1', port=6633)
```

در خط بالا کنترلر را اضافه کردیم که بسیار حائز اهمیت است. چون این خط است که باعث می شود mininet به کنترلری که با ryu اجرا می کنیم متصل شود. پورت 6633 نیز در همین جهت است و عدد خاصی است.

```
sw1 = net.addSwitch('sw1', cls=OVSKernelSwitch)
h1 = net.addHost('h1', cls=Host, ip='10.1.0.1',
                 mac='00:00:00:00:00:01', defaultRoute=None)
net.addLink(sw1, sw3, bw=50)
```

با سه دستور بالا که نمونه ای از تمام هوست ها و سوییچ ها و لینک ها است شبکه را با توجه به bandwidth های نوشته شده ساختیم و س\س با دستورات مناسب اجزای این شبکه را build و start کردیم.

```
net.build()
controller.start()
net.get('sw1').start([c0])
```

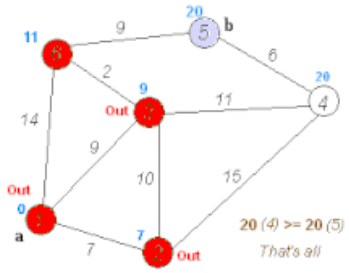
و در نهایت با دستور زیر mininet در cmd اجرا کردیم تا بتوان در ترمینال دستورات مناسب را وارد کرد.

```
CLI(net)
```

❖ بخش RYU با استفاده از dijkstra

❖ الگوریتم dijkstra

ابتدا به توضیح الگوریتم dijkstra می پردازیم تا کد واضح تر شود.



در زیر pseudocode این الگوریتم آمده است

```
1  function Dijkstra(Graph, source):
2
3      create vertex set Q
4
5      for each vertex v in Graph:
6          dist[v] ← INFINITY
7          prev[v] ← UNDEFINED
8          add v to Q
10     dist[source] ← 0
11
12     while Q is not empty:
13         u ← vertex in Q with min dist[u]
14
15         remove u from Q
16
17         for each neighbor v of u:           // only v that are still in Q
18             alt ← dist[u] + length(u, v)
19             if alt < dist[v]:
20                 dist[v] ← alt
21                 prev[v] ← u
22
23     return dist[], prev[]
```

در این الگوریتم از dist برای فاصله هر نود از مبدا جستجو استفاده می شود و همچنین برای اینکه بدانیم به جز طول کوتاه ترین مسیر باید از چه نود هایی بگذریم تا به مقصد برسیم از prev استفاده می شود تا نود قبلی که هر از آن به هر نود وارد شده ایم نگهداری شود.
عملکرد این الگوریتم به این شکل است که هر مرحله از نودی که کمترین dist را دارد که اولین بار خود مبدا با dist برابر 0 است فاصله تا هر نود دیگر را محاسبه می کند و اگر این فاصله جدید از dist قبلی نود دیگر کمتر بود dist آپدیت می شود و همچنین prev نود دیگر این نود انتخاب می شود. به این ترتیب ادامه می دهیم تا هیچ نودی در صف Q باقی نمانده باشد.

❖ توضیح کد RYU بخش dijkstra

```
def get_path(src, dst, first_port, final_port):
    # Dijkstra's algorithm
    print("get_path is called, src=", src, " dst=", dst,
          " first_port=", first_port, " final_port=", final_port)
    # INIT DIST AND PREV
    distance = {}
    previous = {}
    for dpid in switches:
        distance[dpid] = float('Inf')
        previous[dpid] = None
    # SRC DIST IS 0 TO IT'S SELF
    distance[src] = 0
    Q = set(switches)
    print("Q=", Q)
    while len(Q) > 0:
        # CHOOSE THE NODE WITH LEAST DIST
        u = minimum_distance(distance, Q)
        Q.remove(u)
        for p in switches:
            if adjacency[u][p] is not None:
                # FOR EVERY OTHER NODE
                # CALCULATE NEW DIST AND SET IF LESS THAN BEFORE
                w = 1
                if distance[u] + w < distance[p]:
                    distance[p] = distance[u] + w
                    previous[p] = u
    r = []
    p = dst
    r.append(p)
    q = previous[p]
```

```
    # BUILD THE PATH BACKWARDS BASED ON PREVs CREATED BEFORE
    while q is not None:
        if q == src:
            r.append(q)
            break
        p = q
        r.append(p)
        q = previous[p]
    # REVERSE THE PATH TO BE FROM SRC TO DST
    r.reverse()
    # IF THE PATH WAS A LOOP IGNORE PATH
    if src == dst:
        path = [src]
    else:
        path = r
    # Now add the ports
    r = []
    in_port = first_port
    # SET IN AND OUT PORTS OF SWITCHES BASED ON ADJACENCY LIST OF PORTS
    for s1, s2 in zip(path[:-1], path[1:]):
        out_port = adjacency[s1][s2]
        r.append((s1, in_port, out_port))
        in_port = adjacency[s2][s1]
    r.append((dst, in_port, final_port))
    return r
```

همانطور که دیده می شود این بخش کد دقیقا همان پیاده سازی پایتون pseudocodeی است که بالاتر توضیح داده شد. همچنین در کد کامنت هایی برای توضیح هر بخش با حروف بزرگ نوشته شده است.

تنها بخشی که این کد اضافه بر اصل dijkstra دارد یکی ساخت کامل path براساس prev است که صرفا یک while است و توضیح خاصی نیاز ندارد اما برای کامل تر بودن گزارش توضیح می دهیم که در این بخش مسیر به صورت برعکس از dst به سمت src ساخته می شود و در نهایت لیست آن برعکس می شود.

همچنین چک می شود مبدا و مقصد اگر یکی باشند یعنی مسیر یک حلقه است و به همین خاطر مسیر نادیده گرفته می شود.

و در آخر براساس دیکشنری adjacency که مشخص کننده پورت های هر سوییچ به سوییچ دیگر است این مسیر تبدیل به پورت ورودی و خروجی برای هر سوییچ می شود تا روی سوئیچ ها نصب شود.

```
def minimum_distance(distance, Q):
   # FIND THE NODE WITH MIN DIST IN Q
   min = float('Inf')
   node = 0
   for v in Q:
       if distance[v] < min:
           min = distance[v]
           node = v
   return node
```

در get_path از یک تابع کمکی هم استفاده شده که وظیفه اش یافتن نود با کمترین مقدار dist است تا براساس آن فاصله تا تمام نود های دیگر محاسبه شود.

❖ تاثیر دادن bandwidth در dijkstra

همانطور که در کد بالا دیده می شد وزن هر مسیر w = 1 در نظر گرفته شده بود. یعنی همه مسیر ها هزینه 1 داشتند.

اینجا خودمان یک دیکشنری جدید ایجاد کردیم به اسم bw که هزینه هر سوییچ به دیگری را در خودش دارد و اگر نداشته باشد 1 می دهد. ابتدا در 50 خط کد زدن مقادیری که در شکل توپولوژی داده شده بود را نوشتیم. سپس در خط زیر از dijkstra وزن مسیر مشخص شده را در آوردیم. همچنین چون bandwidth اندازه لینک است و ما هزینه می خواستیم مقدار 1 تقسیم بر bw را در دیکشنری ها قرار دادیم.

```
# BANDWIDTH
def reverse_bw(b):
   return 1/b
# DEFAULT BW = 1
bw = defaultdict(lambda: defaultdict(lambda: 1))
# Chicago
bw[1][3] = reverse_bw(50)
bw[3][1] = reverse_bw(50)
bw[2][4] = reverse_bw(50)
...
...
...
for p in switches:
           if adjacency[u][p] is not None:
               # FOR EVERY OTHER NODE
               # CALCULATE NEW DIST AND SET IF LESS THAN BEFORE
               # w = 1
               w = bw[u][p]
               if distance[u] + w < distance[p]:
                   distance[p] = distance[u] + w
                   previous[p] = u
```

خط

```
w = bw[u][p]
```

تغییر مهم ایجاد شده جهت تاثیر دادن bandwidth های موجود است. (همانطور که گفته شد bw در اصل معکوس bandwidth ها است.)

❖ اصلی ترین تابع RYU

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

تابع بالا هسته اصلی این کد است که از سوییچ ها بسته را می گیرد و طبق شبکه dijkstra را اجرا می کند و سپس تمام پورت های ورودی و خروجی که تشخیص داده شده را روی سوییچ های مورد نظر نصب می کند.

طبق لینک زیر اطلاعات فراخوانی این تابع به شکل زیر است.

https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html

Asynchronous Messages

Packet-In Message

class **ryu.ofproto.ofproto_v1_3_parser.OFPPacketIn**(*datapath, buffer_id=None, total_len=None, reason=None, table_id=None, cookie=None, match=None, data=None*)

Packet-In message

The switch sends the packet that received to the controller by this message.

Attribute	Description
buffer_id	ID assigned by datapath
total_len	Full length of frame
reason	Reason packet is being sent. OFPR_NO_MATCH OFPR_ACTION OFPR_INVALID_TTL
table_id	ID of the table that was looked up
cookie	Cookie of the flow entry that was looked up
match	Instance of `OFPMatch`
data	Ethernet frame

همانطور که دیده می شود این تابع به سه دلیل نوشته شده وقتی بسته ای به سوییچ برسد صدا زده می شود و این بسته را برای کنترلر می فرستد. در این زمان است که کنترلر ما مسیر مناسب را می یابد و جداول سوییچ ها را به روز رسانی می کند.

```
# msg: An object which describes the corresponding OpenFlow message.
        msg = ev.msg
        # msg.datapath: A ryu.controller.controller.Datapath instance which
describes an OpenFlow switch from which we received this OpenFlow message.
        datapath = msg.datapath
        # ofproto: A module which exports OpenFlow definitions, mainly constants
appeared in the specification, for the negotiated OpenFlow version. For example,
ryu.ofproto.ofproto_v1_0 for OpenFlow 1.0.
        ofproto = datapath.ofproto
        # ofproto_parser: A module which exports OpenFlow wire message encoder and
decoder for the negotiated OpenFlow version. For example,
ryu.ofproto.ofproto_v1_0_parser for OpenFlow 1.0.
        parser = datapath.ofproto_parser
        # INPUT PORT WHICH THIS PACKET CAME FROM
        in_port = msg.match['in_port']
        # CREATE PACKET FROM MSG.DATA TO ACCESS ETH TYPE
        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocol(ethernet.ethernet)
        # print "eth.ethertype=", eth.ethertype
        # avoid broadcast from LLDP Link Layer Discovery Protocol
        if eth.ethertype == 35020:
            return
```

در ابتدای این تابع اطلاعاتی که می توان از ev گرفت به تفکیک جدا شده اند. برای هر کدام کامنت گذاشته شده که چه محتوایی دارند.

ابتدا msg گرفته شده که محتویات آن دیده می شود.

https://ryu.readthedocs.io/en/latest/ryu_app_api.html#ryu.controller.ofp_event.EventOFPMsgBase

class **ryu.controller.ofp_event.EventOFPMsgBase**(*msg*)

The base class of OpenFlow event class.

OpenFlow event classes have at least the following attributes.

Attribute	Description
msg	An object which describes the corresponding OpenFlow message.
msg.datapath	A ryu.controller.controller.Datapath instance which describes an OpenFlow switch fro
timestamp	Timestamp when Datapath instance generated this event.

The msg object has some more additional members whose values are extracted from the original OpenFlow message.

سپس datapath گرفته می شود از msg

https://ryu.readthedocs.io/en/latest/ryu_app_api.html#ryu.controller.controller.Datapath

ryu.controller.controller.Datapath

class **ryu.controller.controller.Datapath**(*socket*, *address*)

A class to describe an OpenFlow switch connected to this controller.

An instance has the following attributes.

Attribute	Description
id	64-bit OpenFlow Datapath ID. Only available for ryu.controller
ofproto	A module which exports OpenFlow definitions, mainly constan
ofproto_parser	A module which exports OpenFlow wire message encoder and
ofproto_parser.OFPxxxx(datapath,...)	A callable to prepare an OpenFlow message for the given switc
set_xid(self, msg)	Generate an OpenFlow XID and put it in msg.xid.
send_msg(self, msg)	Queue an OpenFlow message to send to the corresponding sw
send_packet_out	deprecated
send_flow_mod	deprecated
send_flow_del	deprecated
send_delete_all_flows	deprecated
send_barrier	Queue an OpenFlow barrier message to send to the switch.
send_nxt_set_flow_format	deprecated
is_reserved_port	deprecated

```
# PACKET SRC AND DST SWITCH MAC
        dst = eth.dst
        src = eth.src

        # ID OF THIS SWITCH IN DATAPATH
        dpid = datapath.id
        if src not in mymac.keys():
            # IF WE DON'T KNOW SRC YET ADD IT
            mymac[src] = (dpid, in_port)
            print("NEW MAC: (DATAPATH_ID, ONE OF THIS SWITCH PORTS ID) =" , src,
mymac[src])
        if dst in mymac.keys():
            # IF WE KNOW IN WHICH SWITCH AND PORT DST IS LOCATED
            # ex. src = 5a:b2:d0:4f:af:45
            p = get_path(mymac[src][0], mymac[dst][0], mymac[src][1],
mymac[dst][1])
            print("***Path:", mymac[src][0], ":", mymac[src][1], "To",
mymac[dst][0], ":",  mymac[dst][1], "is ", end=" ")
            print(p)
            self.install_path(p, ev, src, dst)
            out_port = p[0][2]
        else:
            # IF WE DON'T KNOW WHERE DST IS THEN FLOOD
            # FLOODING
            out_port = ofproto.OFPP_FLOOD
```

سپس ادرس MAC مبدا و مقصد بسته گرفته می شود. از قبل یک mymac تعریف شده است که یک دیکشنری با کلید مک آدرس و محتوای شناسه سوییچ و کدام پورت از سوییچ به آن مک متصل است را دارد. اگر مبدا این بسته که آمده در دیکشنری نباشد به آن اضافه می شود.

بعد اگر مقصد در دیکشنری باشد مسیر با get_path و الگوریتم dijkstra پیدا می شود و سپس روی تمامی سوییچ های مسیر با install_path نصب می شود.
اما اگر ما ندانیم مک مقصد به کدام سوییچ متصل است طبیعتا نمی توانیم مسیر یابی کنیم و به همین دلیل flood رخ می دهد.

برای توضیح بیشتر می‌توان گفت کاری که انجام می‌شود به این شکل است که مسیر بین مبدا و مقصد تشخیص داده می‌شود و طبق آن مسیر مشخص شده و همینطور اطلاعاتی که از قبل داریم، flow table ها update می‌شوند.

در ابتدا طبق الگوریتم گفته شده بین مبدا و مقصد مسیر مناسب پیدا می‌شود که خروجی آن لسیتی از سوئیچ ها و همینطور پورت های ورودی و خروجی متناسب با آن‌ها می‌باشد.

این اطلاعات به صورت برعکس گرفته می‌شود تا بتوان از آن در مسیریابی مبدا به مقصد استفاده کرد.

در مرحله بعدی، با توجه به این اطلاعات استخراج شده و اطلاعات مربوط به هر سوئیچ و ID آن‌ها node مربوطه match شده که این کار با استفاده از اطلاعاتی که از هر node داریم بدست می‌آید.

در مرحله بعد با توجه به پورت های خروجی و ورودی که بدست آمده‌اند، برای هر سوئیچ match شده، اطلاعات action برای آن‌ها update می‌شود.

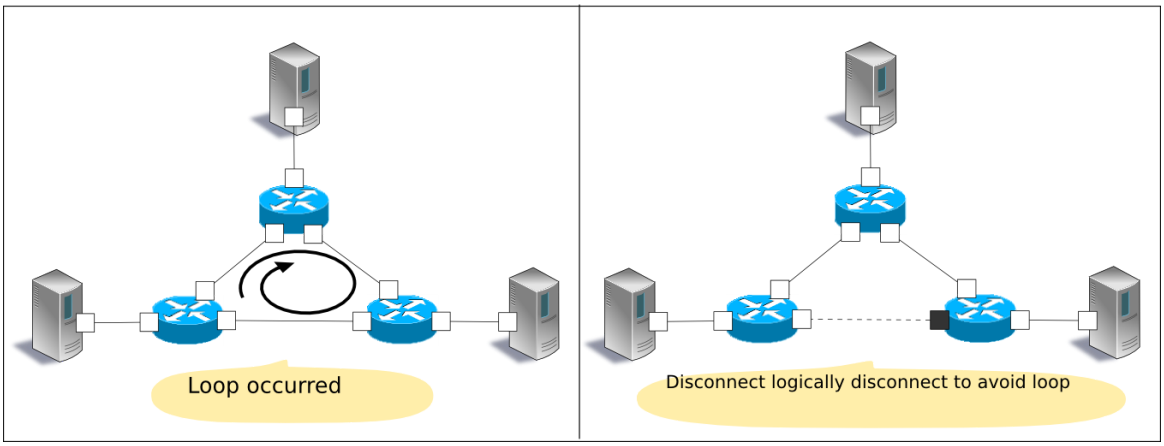
کارهای گفته شده طبق کدهای زیر انجام می‌گیرند.

```
for sw, in_port, out_port in p:

          print(src_mac,"->", dst_mac, "via ", sw, " in_port=", in_port, "
out_port=", out_port)

          # FIND THE SWITCH MATCHING OUR SETTING FOR SWITCH ID AND MAC ADDRESSES

          match = parser.OFPMatch(

              in_port=in_port, eth_src=src_mac, eth_dst=dst_mac)
```

```
            # GENERATE THE ACTION TO BE DONE IN FOREMENTIONED CIRCUMSTANCES WHICH IS
ADDING THE OUT PUT PORT

            actions = [parser.OFPActionOutput(out_port)]

            # FIND THE ACCORDING SWITCH

            datapath = self.datapath_list[int(sw) - 1]

            # APPLY THE GENERATED ACTION

            inst = [parser.OFPInstructionActions(

                ofproto.OFPIT_APPLY_ACTIONS, actions)]

            # Modify Flow entry message

            # The controller sends this message to modify the flow table.

            mod = datapath.ofproto_parser.OFPFlowMod(

                datapath=datapath, match=match, idle_timeout=0, hard_timeout=0,
priority=1, instructions=inst)

            # SEND THE OPENFLOW ITEM (BROADCASTING)

            datapath.send_msg(mod)
```

❖ مشکل دور در گراف

وجود دور در گراف شبکه وقتی مشکل ساز می شود که نودی شروع به flood کند. در این حالت این نود به یک نود دیگر می گوید و اگر دوری باشد در نهایت همین پیام به خودش می رسد و باز همین دور ادامه می یابد.



❖ رفع باگ کد

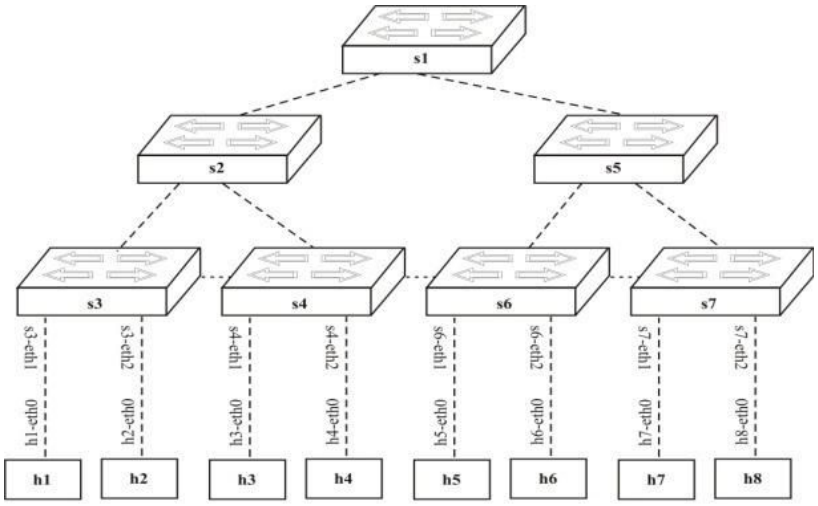
در خط زیر فرض شده بود که سوییچ ها به ترتیب اضافه می شوند که فرض درستی نیست به همین علت به جای برداشتن آخرین دیتاپس حالا آن را پیدا می کنیم.

```
# datapath = self.datapath_list[int(sw) - 1]
datapath = next(d for d in self.datapath_list if d.id == sw)
```

خطی که کامنت شده باگ داشته و خط دوم که خودمان زدیم به درستی datapath متناظر با sw را پیدا می کند.

نتایج:

❖ نتایج اجرای کد روی درخت دستی نوشته شده



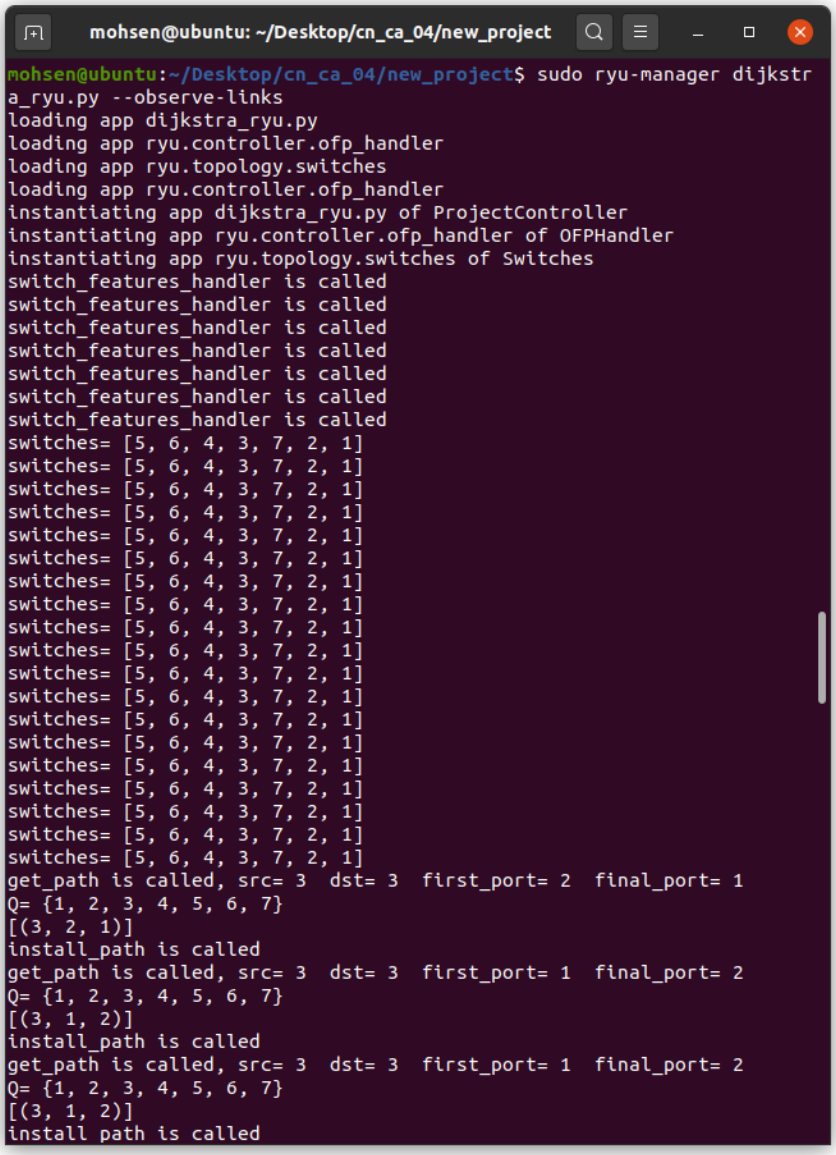

```
mininet> net
h1 h1-eth0:s3-eth1
h2 h2-eth0:s3-eth2
h3 h3-eth0:s4-eth1
h4 h4-eth0:s4-eth2
h5 h5-eth0:s6-eth1
h6 h6-eth0:s6-eth2
h7 h7-eth0:s7-eth1
h8 h8-eth0:s7-eth2
s1 lo:  s1-eth1:s2-eth3 s1-eth2:s5-eth3
s2 lo:  s2-eth1:s3-eth3 s2-eth2:s4-eth3 s2-eth3:s1-eth1
s3 lo:  s3-eth1:h1-eth0 s3-eth2:h2-eth0 s3-eth3:s2-eth1
s4 lo:  s4-eth1:h3-eth0 s4-eth2:h4-eth0 s4-eth3:s2-eth2
s5 lo:  s5-eth1:s6-eth3 s5-eth2:s7-eth3 s5-eth3:s1-eth2
s6 lo:  s6-eth1:h5-eth0 s6-eth2:h6-eth0 s6-eth3:s5-eth1
s7 lo:  s7-eth1:h7-eth0 s7-eth2:h8-eth0 s7-eth3:s5-eth2
c0
```

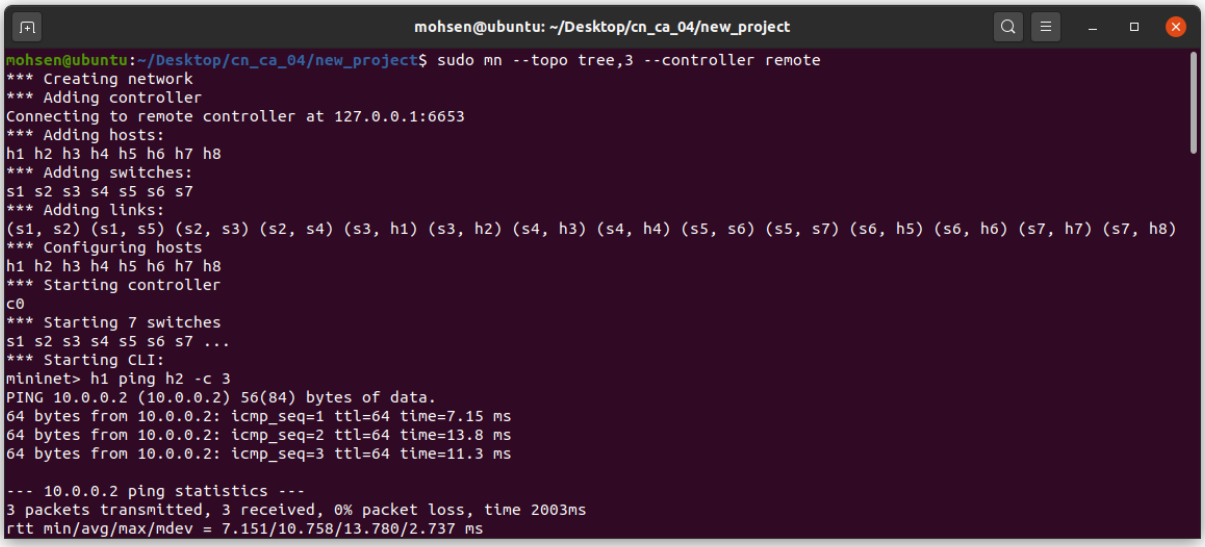
ابتدا کد این درخت را در mininet نوشتیم.

```
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
```

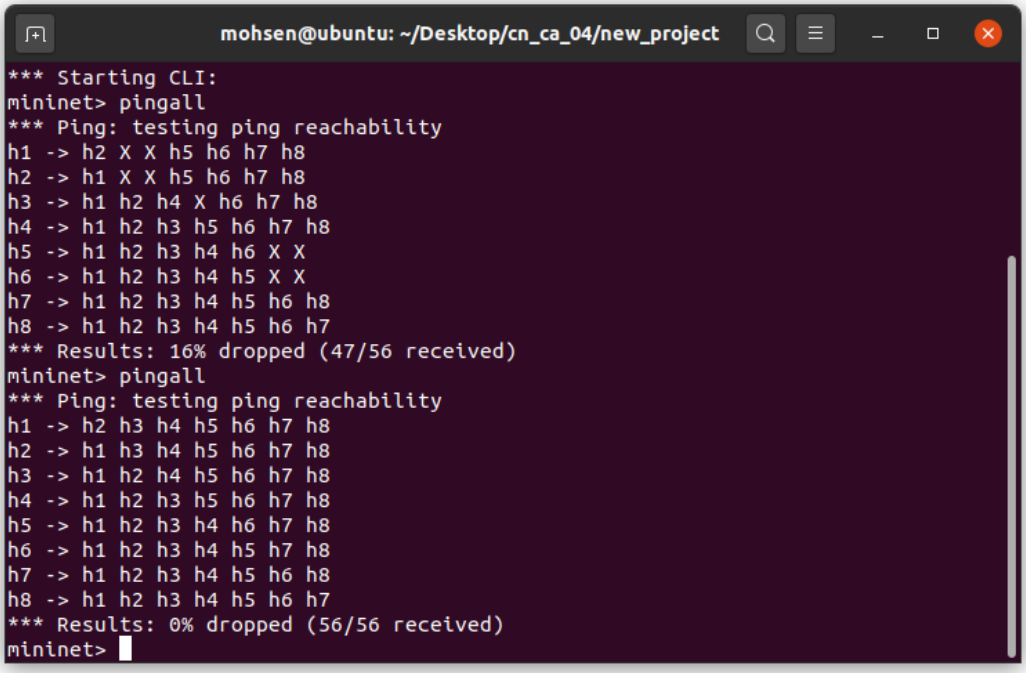
```
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call


def myNetwork():
   net = Mininet(topo=None,
                 build=False,
                 ipBase='10.0.0.0/8', autoStaticArp=False)

   info('*** Adding controller\n')
   c0 = net.addController(name='c0',
                          controller=RemoteController,
                          ip='127.0.0.1',
                          port=6633)
   info('*** Add switches\n')
   sw1 = net.addSwitch('sw1', cls=OVSKernelSwitch)
   sw2 = net.addSwitch('sw2', cls=OVSKernelSwitch)
   sw3 = net.addSwitch('sw3', cls=OVSKernelSwitch)
   sw4 = net.addSwitch('sw4', cls=OVSKernelSwitch)
   sw5 = net.addSwitch('sw5', cls=OVSKernelSwitch)
   sw6 = net.addSwitch('sw6', cls=OVSKernelSwitch)
   sw7 = net.addSwitch('sw7', cls=OVSKernelSwitch)
   info('*** Add hosts\n')
   h1 = net.addHost('h1', cls=Host, ip='10.1.0.1',
                    mac='00:00:00:00:00:01', defaultRoute=None)
   h2 = net.addHost('h2', cls=Host, ip='10.1.0.2',
                    mac='00:00:00:00:00:02', defaultRoute=None)
   h3 = net.addHost('h3', cls=Host, ip='10.1.0.3',
                    mac='00:00:00:00:00:03', defaultRoute=None)
   h4 = net.addHost('h4', cls=Host, ip='10.1.0.4',
                    mac='00:00:00:00:00:04', defaultRoute=None)
   h5 = net.addHost('h5', cls=Host, ip='10.1.0.5',
                    mac='00:00:00:00:00:05', defaultRoute=None)
   h6 = net.addHost('h6', cls=Host, ip='10.1.0.6',
                    mac='00:00:00:00:00:06', defaultRoute=None)
   h7 = net.addHost('h7', cls=Host, ip='10.1.0.7',
                    mac='00:00:00:00:00:07', defaultRoute=None)
   h8 = net.addHost('h8', cls=Host, ip='10.1.0.8',
                    mac='00:00:00:00:00:08', defaultRoute=None)

   info('*** Add links\n')
   net.addLink(h1, sw3)
   net.addLink(h2, sw3)
   net.addLink(h3, sw4)
   net.addLink(h4, sw4)
   net.addLink(h5, sw6)
   net.addLink(h6, sw6)
   net.addLink(h7, sw7)
```

```
    net.addLink(h8, sw7)
    net.addLink(sw3, sw2)
    net.addLink(sw4, sw2)
    net.addLink(sw6, sw5)
    net.addLink(sw7, sw5)
    net.addLink(sw2, sw1)
    net.addLink(sw5, sw1)

    info('*** Starting network\n')
    net.build()
    info('*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()

    info('*** Starting switches\n')
    net.get('sw1').start([c0])
    net.get('sw2').start([c0])
    net.get('sw3').start([c0])
    net.get('sw4').start([c0])
    net.get('sw5').start([c0])
    net.get('sw6').start([c0])
    net.get('sw7').start([c0])

    info('*** Post configure switches and hosts\n')

    CLI(net)
    net.stop()


if __name__ == '__main__':
    setLogLevel('info')
    myNetwork()
```

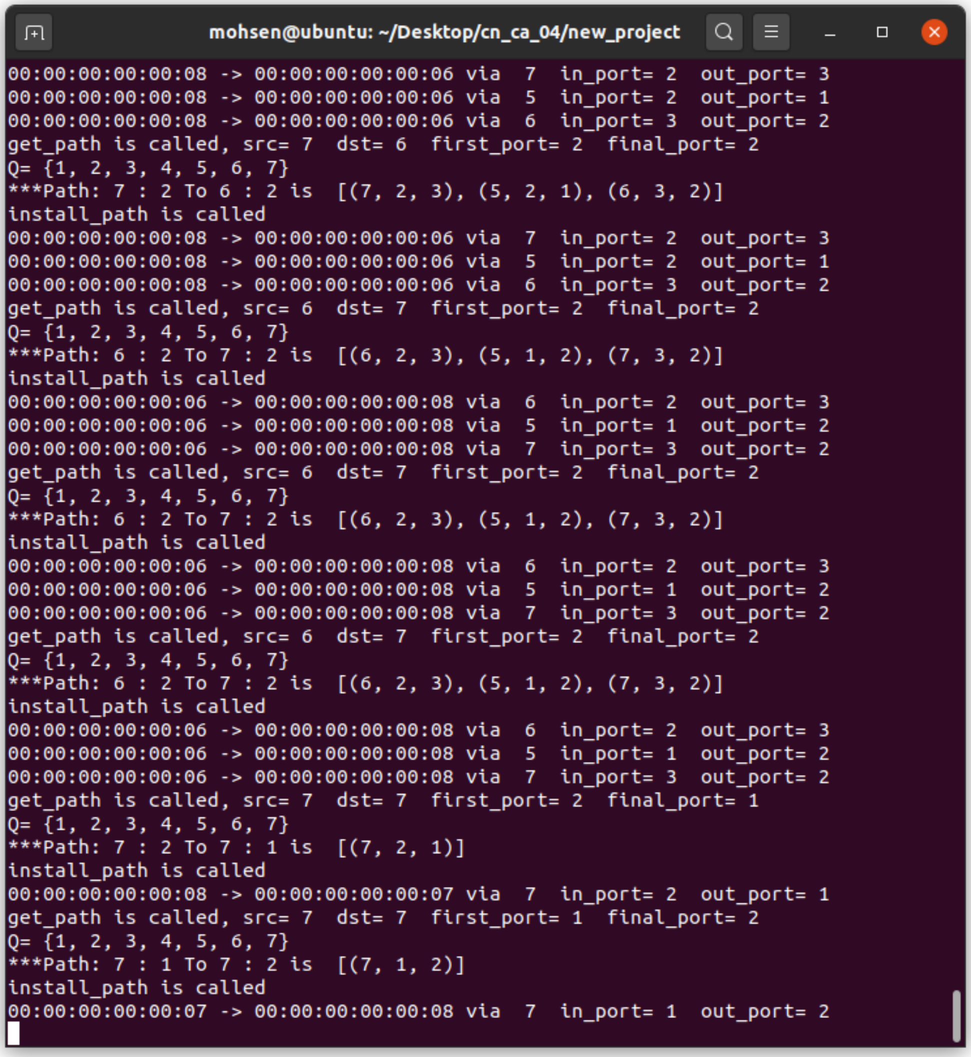

mohsen@ubuntu: ~/Desktop/cn_ca_04/new_project
mohsen@ubuntu:~/Desktop/cn_ca_04/new_project$ sudo ryu-manager dijkstra_ryu.py --observe-links
loading app dijkstra_ryu.py
loading app ryu.controller.ofp_handler
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app dijkstra_ryu.py of ProjectController
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu.topology.switches of Switches
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switches= [5, 6, 4, 3, 7, 2, 1]
switches= [5, 6, 4, 3, 7, 2, 1]
switches= [5, 6, 4, 3, 7, 2, 1]
switches= [5, 6, 4, 3, 7, 2, 1]
switches= [5, 6, 4, 3, 7, 2, 1]
switches= [5, 6, 4, 3, 7, 2, 1]
switches= [5, 6, 4, 3, 7, 2, 1]
switches= [5, 6, 4, 3, 7, 2, 1]
switches= [5, 6, 4, 3, 7, 2, 1]
switches= [5, 6, 4, 3, 7, 2, 1]
switches= [5, 6, 4, 3, 7, 2, 1]
switches= [5, 6, 4, 3, 7, 2, 1]
switches= [5, 6, 4, 3, 7, 2, 1]
switches= [5, 6, 4, 3, 7, 2, 1]
switches= [5, 6, 4, 3, 7, 2, 1]
switches= [5, 6, 4, 3, 7, 2, 1]
switches= [5, 6, 4, 3, 7, 2, 1]
switches= [5, 6, 4, 3, 7, 2, 1]
get_path is called, src= 3 dst= 3 first_port= 2 final_port= 1
Q= {1, 2, 3, 4, 5, 6, 7}
[(3, 2, 1)]
install_path is called
get_path is called, src= 3 dst= 3 first_port= 1 final_port= 2
Q= {1, 2, 3, 4, 5, 6, 7}
[(3, 1, 2)]
install_path is called
get_path is called, src= 3 dst= 3 first_port= 1 final_port= 2
Q= {1, 2, 3, 4, 5, 6, 7}
[(3, 1, 2)]
install path is called


mohsen@ubuntu: ~/Desktop/cn_ca_04/new_project
mohsen@ubuntu:~/Desktop/cn_ca_04/new_project$ sudo mn --topo tree,3 --controller remote
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(s1, s2) (s1, s5) (s2, s3) (s2, s4) (s3, h1) (s3, h2) (s4, h3) (s4, h4) (s5, s6) (s5, s7) (s6, h5) (s6, h6) (s7, h7) (s7, h8)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7 ...
*** Starting CLI:
mininet> h1 ping h2 -c 3
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=7.15 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=13.8 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=11.3 ms
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 7.151/10.758/13.780/2.737 ms

نتیجه اجرای pingall



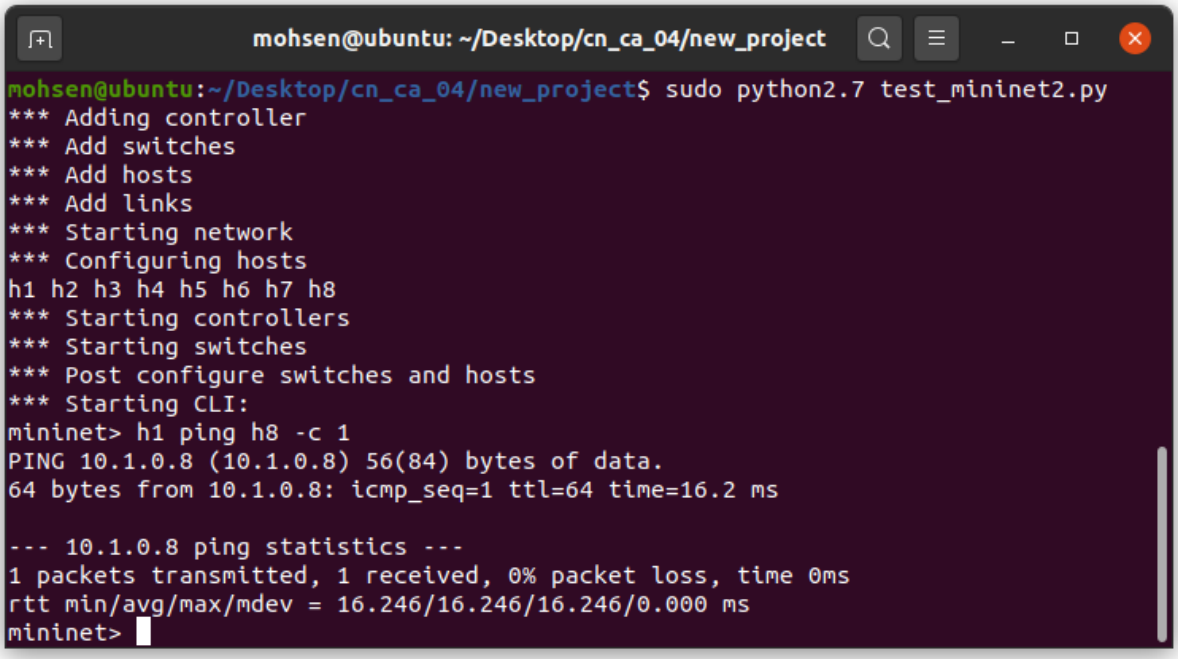
همانطور که دیده می شود اولین بار در حین نصب بعضی از بسته ها نرسیدند اما برای دومین بار که مسیر ها اکثرا از قبل نصب شده بودند دیگر هیچ مشکلی نبود و همه بسته ها به راحتی رسیدند.



```
mohsen@ubuntu: ~/Desktop/cn_ca_04/new_project
00:00:00:00:00:08 -> 00:00:00:00:00:06 via  7  in_port= 2  out_port= 3
00:00:00:00:00:08 -> 00:00:00:00:00:06 via  5  in_port= 2  out_port= 1
00:00:00:00:00:08 -> 00:00:00:00:00:06 via  6  in_port= 3  out_port= 2
get_path is called, src= 7  dst= 6  first_port= 2  final_port= 2
Q= {1, 2, 3, 4, 5, 6, 7}
***Path: 7 : 2 To 6 : 2 is  [(7, 2, 3), (5, 2, 1), (6, 3, 2)]
install_path is called
00:00:00:00:00:08 -> 00:00:00:00:00:06 via  7  in_port= 2  out_port= 3
00:00:00:00:00:08 -> 00:00:00:00:00:06 via  5  in_port= 2  out_port= 1
00:00:00:00:00:08 -> 00:00:00:00:00:06 via  6  in_port= 3  out_port= 2
get_path is called, src= 6  dst= 7  first_port= 2  final_port= 2
Q= {1, 2, 3, 4, 5, 6, 7}
***Path: 6 : 2 To 7 : 2 is  [(6, 2, 3), (5, 1, 2), (7, 3, 2)]
install_path is called
00:00:00:00:00:06 -> 00:00:00:00:00:08 via  6  in_port= 2  out_port= 3
00:00:00:00:00:06 -> 00:00:00:00:00:08 via  5  in_port= 1  out_port= 2
00:00:00:00:00:06 -> 00:00:00:00:00:08 via  7  in_port= 3  out_port= 2
get_path is called, src= 6  dst= 7  first_port= 2  final_port= 2
Q= {1, 2, 3, 4, 5, 6, 7}
***Path: 6 : 2 To 7 : 2 is  [(6, 2, 3), (5, 1, 2), (7, 3, 2)]
install_path is called
00:00:00:00:00:06 -> 00:00:00:00:00:08 via  6  in_port= 2  out_port= 3
00:00:00:00:00:06 -> 00:00:00:00:00:08 via  5  in_port= 1  out_port= 2
00:00:00:00:00:06 -> 00:00:00:00:00:08 via  7  in_port= 3  out_port= 2
get_path is called, src= 6  dst= 7  first_port= 2  final_port= 2
Q= {1, 2, 3, 4, 5, 6, 7}
***Path: 6 : 2 To 7 : 2 is  [(6, 2, 3), (5, 1, 2), (7, 3, 2)]
install_path is called
00:00:00:00:00:06 -> 00:00:00:00:00:08 via  6  in_port= 2  out_port= 3
00:00:00:00:00:06 -> 00:00:00:00:00:08 via  5  in_port= 1  out_port= 2
00:00:00:00:00:06 -> 00:00:00:00:00:08 via  7  in_port= 3  out_port= 2
get_path is called, src= 7  dst= 7  first_port= 2  final_port= 1
Q= {1, 2, 3, 4, 5, 6, 7}
***Path: 7 : 2 To 7 : 1 is  [(7, 2, 1)]
install_path is called
00:00:00:00:00:08 -> 00:00:00:00:00:07 via  7  in_port= 2  out_port= 1
get_path is called, src= 7  dst= 7  first_port= 1  final_port= 2
Q= {1, 2, 3, 4, 5, 6, 7}
***Path: 7 : 1 To 7 : 2 is  [(7, 1, 2)]
install_path is called
00:00:00:00:00:07 -> 00:00:00:00:00:08 via  7  in_port= 1  out_port= 2
```

همینطور در ryu میبینیم که مثلا برای رفتن از h6 به h8 مسیر سوییچ های 6 به 5 به 7 را پیشنهاد داده که صحیح است

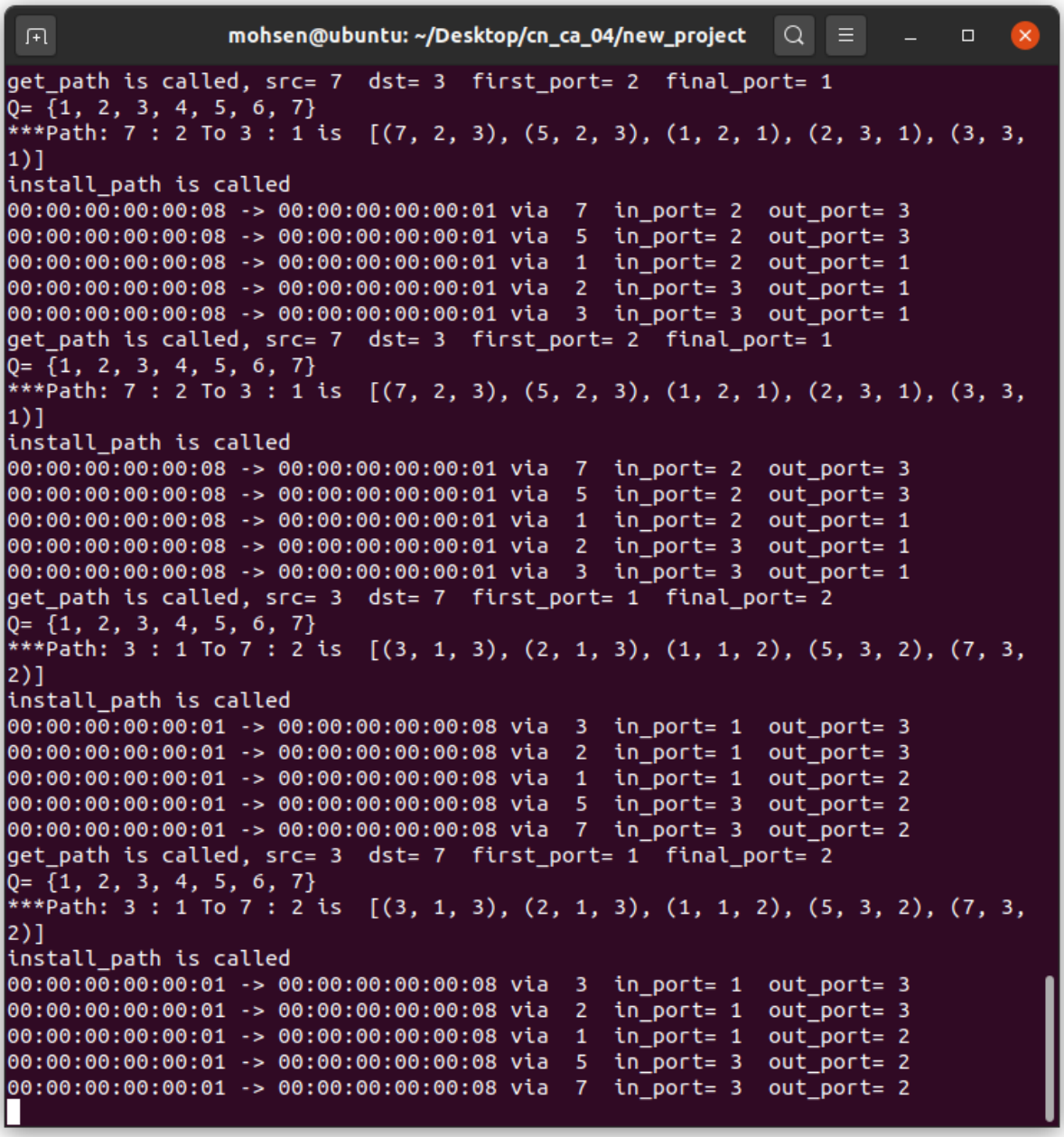
تست ارسال از h1 به h8



برای این کار دستور

h1 ping h2 -c 1

را اجرا کردیم که همانطور که دیده می شود با موفقیت دریافت شد.

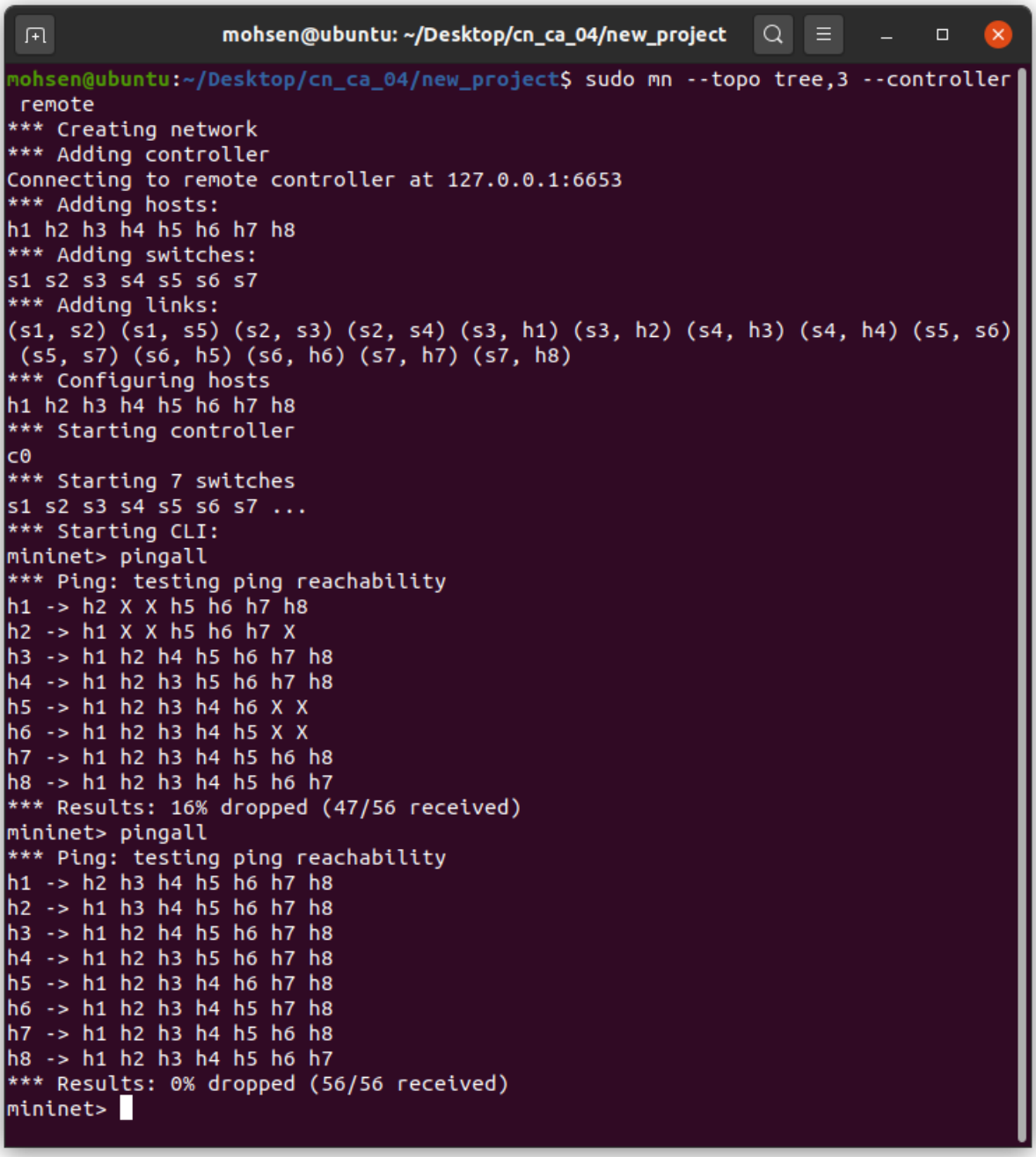

mohsen@ubuntu: ~/Desktop/cn_ca_04/new_project
get_path is called, src= 7 dst= 3 first_port= 2 final_port= 1
Q= {1, 2, 3, 4, 5, 6, 7}
***Path: 7 : 2 To 3 : 1 is [(7, 2, 3), (5, 2, 3), (1, 2, 1), (2, 3, 1), (3, 3, 1)]
install_path is called
00:00:00:00:00:08 -> 00:00:00:00:00:01 via 7 in_port= 2 out_port= 3
00:00:00:00:00:08 -> 00:00:00:00:00:01 via 5 in_port= 2 out_port= 3
00:00:00:00:00:08 -> 00:00:00:00:00:01 via 1 in_port= 2 out_port= 1
00:00:00:00:00:08 -> 00:00:00:00:00:01 via 2 in_port= 3 out_port= 1
00:00:00:00:00:08 -> 00:00:00:00:00:01 via 3 in_port= 3 out_port= 1
get_path is called, src= 7 dst= 3 first_port= 2 final_port= 1
Q= {1, 2, 3, 4, 5, 6, 7}
***Path: 7 : 2 To 3 : 1 is [(7, 2, 3), (5, 2, 3), (1, 2, 1), (2, 3, 1), (3, 3, 1)]
install_path is called
00:00:00:00:00:08 -> 00:00:00:00:00:01 via 7 in_port= 2 out_port= 3
00:00:00:00:00:08 -> 00:00:00:00:00:01 via 5 in_port= 2 out_port= 3
00:00:00:00:00:08 -> 00:00:00:00:00:01 via 1 in_port= 2 out_port= 1
00:00:00:00:00:08 -> 00:00:00:00:00:01 via 2 in_port= 3 out_port= 1
00:00:00:00:00:08 -> 00:00:00:00:00:01 via 3 in_port= 3 out_port= 1
get_path is called, src= 3 dst= 7 first_port= 1 final_port= 2
Q= {1, 2, 3, 4, 5, 6, 7}
***Path: 3 : 1 To 7 : 2 is [(3, 1, 3), (2, 1, 3), (1, 1, 2), (5, 3, 2), (7, 3, 2)]
install_path is called
00:00:00:00:00:01 -> 00:00:00:00:00:08 via 3 in_port= 1 out_port= 3
00:00:00:00:00:01 -> 00:00:00:00:00:08 via 2 in_port= 1 out_port= 3
00:00:00:00:00:01 -> 00:00:00:00:00:08 via 1 in_port= 1 out_port= 2
00:00:00:00:00:01 -> 00:00:00:00:00:08 via 5 in_port= 3 out_port= 2
00:00:00:00:00:01 -> 00:00:00:00:00:08 via 7 in_port= 3 out_port= 2
get_path is called, src= 3 dst= 7 first_port= 1 final_port= 2
Q= {1, 2, 3, 4, 5, 6, 7}
***Path: 3 : 1 To 7 : 2 is [(3, 1, 3), (2, 1, 3), (1, 1, 2), (5, 3, 2), (7, 3, 2)]
install_path is called
00:00:00:00:00:01 -> 00:00:00:00:00:08 via 3 in_port= 1 out_port= 3
00:00:00:00:00:01 -> 00:00:00:00:00:08 via 2 in_port= 1 out_port= 3
00:00:00:00:00:01 -> 00:00:00:00:00:08 via 1 in_port= 1 out_port= 2
00:00:00:00:00:01 -> 00:00:00:00:00:08 via 5 in_port= 3 out_port= 2
00:00:00:00:00:01 -> 00:00:00:00:00:08 via 7 in_port= 3 out_port= 2

همینطور در ryu میبینیم که برای ارسال از h1 به h8 مسیر گذشتن از سوییچ های 3 به 2 به 1 به 5 به 7 را می دهد که با توجه به گراف درخت شبکه که بالاتر وجود دارد دیده می شود که این مسیر دقیقا مسیر درستی است که از ریشه نیز می گذرد.

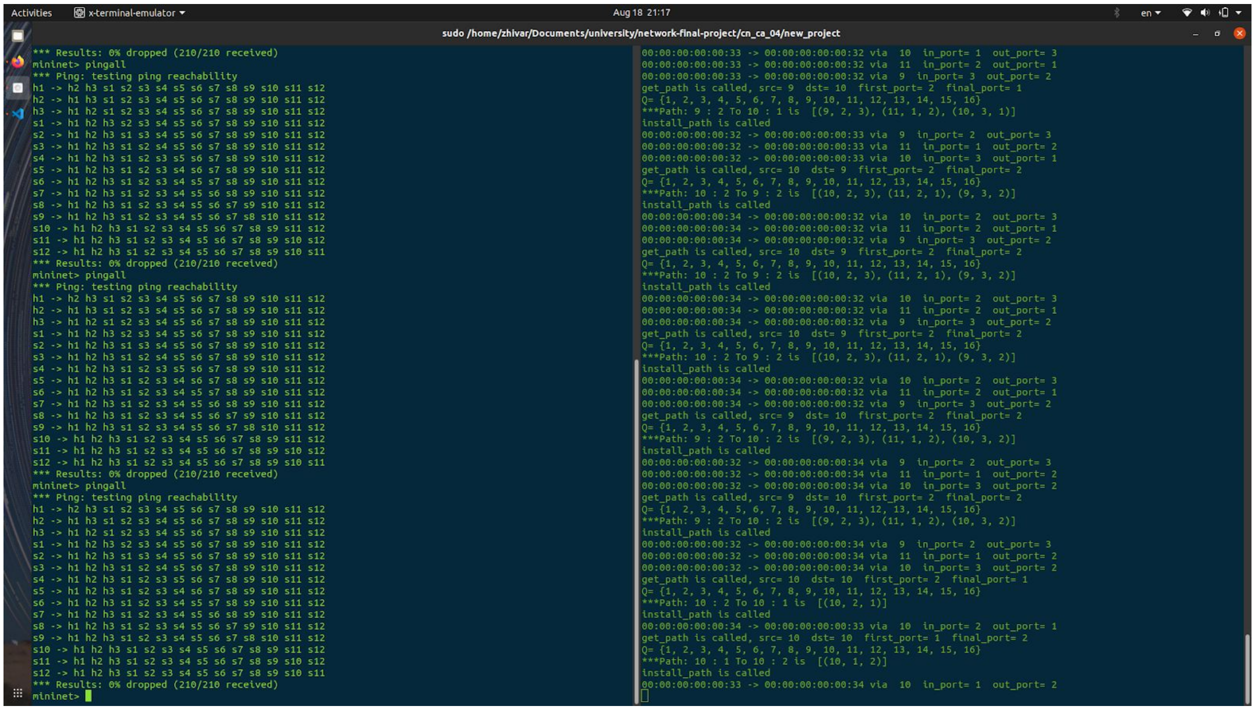
چون روش ارسال TCP است هم مسیر رفت ساخته می شود و هم برگشت.

بنابراین در این گزارش خط به خط کد فهمیده و توضیح داده شد و تغییرات لازم داده شد و شبکه درختی طراحی شد و تست های مناسب با خروجی های واضح گرفته شد و نشان می دهد که کد به درستی اجرا شده و تمام مراحل آن به درستی توسط ما درک شده است.

نتایج اجرا روی درخت پیش فرض مانند عکس های سایت داده شده


mohsen@ubuntu: ~/Desktop/cn_ca_04/new_project
mohsen@ubuntu:~/Desktop/cn_ca_04/new_project$ sudo mn --topo tree,3 --controller
remote
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(s1, s2) (s1, s5) (s2, s3) (s2, s4) (s3, h1) (s3, h2) (s4, h3) (s4, h4) (s5, s6)
(s5, s7) (s6, h5) (s6, h6) (s7, h7) (s7, h8)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X X h5 h6 h7 h8
h2 -> h1 X X h5 h6 h7 X
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 X X
h6 -> h1 h2 h3 h4 h5 X X
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 16% dropped (47/56 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
mininet>

```
mohsen@ubuntu: ~/Desktop/cn_ca_04/new_project
get_path is called, src= 7  dst= 6  first_port= 2  final_port= 2
Q= {1, 2, 3, 4, 5, 6, 7}
***Path: 7 : 2 To 6 : 2 is  [(7, 2, 3), (5, 2, 1), (6, 3, 2)]
install_path is called
3a:03:c5:72:fb:ec -> 0e:15:3e:ac:5f:e2 via  7  in_port= 2  out_port= 3
3a:03:c5:72:fb:ec -> 0e:15:3e:ac:5f:e2 via  5  in_port= 2  out_port= 1
3a:03:c5:72:fb:ec -> 0e:15:3e:ac:5f:e2 via  6  in_port= 3  out_port= 2
get_path is called, src= 7  dst= 6  first_port= 2  final_port= 2
Q= {1, 2, 3, 4, 5, 6, 7}
***Path: 7 : 2 To 6 : 2 is  [(7, 2, 3), (5, 2, 1), (6, 3, 2)]
install_path is called
3a:03:c5:72:fb:ec -> 0e:15:3e:ac:5f:e2 via  7  in_port= 2  out_port= 3
3a:03:c5:72:fb:ec -> 0e:15:3e:ac:5f:e2 via  5  in_port= 2  out_port= 1
3a:03:c5:72:fb:ec -> 0e:15:3e:ac:5f:e2 via  6  in_port= 3  out_port= 2
get_path is called, src= 6  dst= 7  first_port= 2  final_port= 2
Q= {1, 2, 3, 4, 5, 6, 7}
***Path: 6 : 2 To 7 : 2 is  [(6, 2, 3), (5, 1, 2), (7, 3, 2)]
install_path is called
0e:15:3e:ac:5f:e2 -> 3a:03:c5:72:fb:ec via  6  in_port= 2  out_port= 3
0e:15:3e:ac:5f:e2 -> 3a:03:c5:72:fb:ec via  5  in_port= 1  out_port= 2
0e:15:3e:ac:5f:e2 -> 3a:03:c5:72:fb:ec via  7  in_port= 3  out_port= 2
get_path is called, src= 6  dst= 7  first_port= 2  final_port= 2
Q= {1, 2, 3, 4, 5, 6, 7}
***Path: 6 : 2 To 7 : 2 is  [(6, 2, 3), (5, 1, 2), (7, 3, 2)]
install_path is called
0e:15:3e:ac:5f:e2 -> 3a:03:c5:72:fb:ec via  6  in_port= 2  out_port= 3
0e:15:3e:ac:5f:e2 -> 3a:03:c5:72:fb:ec via  5  in_port= 1  out_port= 2
0e:15:3e:ac:5f:e2 -> 3a:03:c5:72:fb:ec via  7  in_port= 3  out_port= 2
get_path is called, src= 6  dst= 7  first_port= 2  final_port= 2
Q= {1, 2, 3, 4, 5, 6, 7}
***Path: 6 : 2 To 7 : 2 is  [(6, 2, 3), (5, 1, 2), (7, 3, 2)]
install_path is called
0e:15:3e:ac:5f:e2 -> 3a:03:c5:72:fb:ec via  6  in_port= 2  out_port= 3
0e:15:3e:ac:5f:e2 -> 3a:03:c5:72:fb:ec via  5  in_port= 1  out_port= 2
0e:15:3e:ac:5f:e2 -> 3a:03:c5:72:fb:ec via  7  in_port= 3  out_port= 2
get_path is called, src= 7  dst= 7  first_port= 2  final_port= 1
Q= {1, 2, 3, 4, 5, 6, 7}
***Path: 7 : 2 To 7 : 1 is  [(7, 2, 1)]
install_path is called
3a:03:c5:72:fb:ec -> ca:80:31:9c:de:c7 via  7  in_port= 2  out_port= 1
get_path is called, src= 7  dst= 7  first_port= 1  final_port= 2
Q= {1, 2, 3, 4, 5, 6, 7}
***Path: 7 : 1 To 7 : 2 is  [(7, 1, 2)]
install_path is called
ca:80:31:9c:de:c7 -> 3a:03:c5:72:fb:ec via  7  in_port= 1  out_port= 2
```


Activities
x-terminal-emulator
Aug 18 21:17
en
sudo /home/zhivar/Documents/university/network-final-project/cn_ca_04/new_project
*** Results: 0% dropped (210/210 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12
h2 -> h1 h3 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12
h3 -> h1 h2 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12
s1 -> h1 h2 h3 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12
s2 -> h1 h2 h3 s1 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12
s3 -> h1 h2 h3 s1 s2 s4 s5 s6 s7 s8 s9 s10 s11 s12
s4 -> h1 h2 h3 s1 s2 s3 s5 s6 s7 s8 s9 s10 s11 s12
s5 -> h1 h2 h3 s1 s2 s3 s4 s6 s7 s8 s9 s10 s11 s12
s6 -> h1 h2 h3 s1 s2 s3 s4 s5 s7 s8 s9 s10 s11 s12
s7 -> h1 h2 h3 s1 s2 s3 s4 s5 s6 s8 s9 s10 s11 s12
s8 -> h1 h2 h3 s1 s2 s3 s4 s5 s6 s7 s9 s10 s11 s12
s9 -> h1 h2 h3 s1 s2 s3 s4 s5 s6 s7 s8 s10 s11 s12
s10 -> h1 h2 h3 s1 s2 s3 s4 s5 s6 s7 s8 s9 s11 s12
s11 -> h1 h2 h3 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s12
s12 -> h1 h2 h3 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11
*** Results: 0% dropped (210/210 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12
h2 -> h1 h3 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12
h3 -> h1 h2 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12
s1 -> h1 h2 h3 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12
s2 -> h1 h2 h3 s1 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12
s3 -> h1 h2 h3 s1 s2 s4 s5 s6 s7 s8 s9 s10 s11 s12
s4 -> h1 h2 h3 s1 s2 s3 s5 s6 s7 s8 s9 s10 s11 s12
s5 -> h1 h2 h3 s1 s2 s3 s4 s6 s7 s8 s9 s10 s11 s12
s6 -> h1 h2 h3 s1 s2 s3 s4 s5 s7 s8 s9 s10 s11 s12
s7 -> h1 h2 h3 s1 s2 s3 s4 s5 s6 s8 s9 s10 s11 s12
s8 -> h1 h2 h3 s1 s2 s3 s4 s5 s6 s7 s9 s10 s11 s12
s9 -> h1 h2 h3 s1 s2 s3 s4 s5 s6 s7 s8 s10 s11 s12
s10 -> h1 h2 h3 s1 s2 s3 s4 s5 s6 s7 s8 s9 s11 s12
s11 -> h1 h2 h3 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s12
s12 -> h1 h2 h3 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11
*** Results: 0% dropped (210/210 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12
h2 -> h1 h3 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12
h3 -> h1 h2 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12
s1 -> h1 h2 h3 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12
s2 -> h1 h2 h3 s1 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12
s3 -> h1 h2 h3 s1 s2 s4 s5 s6 s7 s8 s9 s10 s11 s12
s4 -> h1 h2 h3 s1 s2 s3 s5 s6 s7 s8 s9 s10 s11 s12
s5 -> h1 h2 h3 s1 s2 s3 s4 s6 s7 s8 s9 s10 s11 s12
s6 -> h1 h2 h3 s1 s2 s3 s4 s5 s7 s8 s9 s10 s11 s12
s7 -> h1 h2 h3 s1 s2 s3 s4 s5 s6 s8 s9 s10 s11 s12
s8 -> h1 h2 h3 s1 s2 s3 s4 s5 s6 s7 s9 s10 s11 s12
s9 -> h1 h2 h3 s1 s2 s3 s4 s5 s6 s7 s8 s10 s11 s12
s10 -> h1 h2 h3 s1 s2 s3 s4 s5 s6 s7 s8 s9 s11 s12
s11 -> h1 h2 h3 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s12
s12 -> h1 h2 h3 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11
*** Results: 0% dropped (210/210 received)
mininet>
00:00:00:00:00:33 -> 00:00:00:00:00:32 via 10 in_port= 1 out_port= 3
00:00:00:00:00:33 -> 00:00:00:00:00:32 via 11 in_port= 2 out_port= 1
00:00:00:00:00:33 -> 00:00:00:00:00:32 via 9 in_port= 3 out_port= 2
get_path is called, src= 9 dst= 10 first_port= 2 final_port= 1
Q= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
***Path: 9 : 2 To 10 : 1 is [(9, 2, 3), (11, 1, 2), (10, 3, 1)]
install_path is called
00:00:00:00:00:32 -> 00:00:00:00:00:33 via 9 in_port= 2 out_port= 3
00:00:00:00:00:32 -> 00:00:00:00:00:33 via 11 in_port= 1 out_port= 2
00:00:00:00:00:32 -> 00:00:00:00:00:33 via 10 in_port= 3 out_port= 1
get_path is called, src= 10 dst= 9 first_port= 2 final_port= 2
Q= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
***Path: 10 : 2 To 9 : 2 is [(10, 2, 3), (11, 2, 1), (9, 3, 2)]
install_path is called
00:00:00:00:00:34 -> 00:00:00:00:00:32 via 10 in_port= 2 out_port= 3
00:00:00:00:00:34 -> 00:00:00:00:00:32 via 11 in_port= 2 out_port= 1
00:00:00:00:00:34 -> 00:00:00:00:00:32 via 9 in_port= 3 out_port= 2
get_path is called, src= 10 dst= 9 first_port= 2 final_port= 2
Q= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
***Path: 10 : 2 To 9 : 2 is [(10, 2, 3), (11, 2, 1), (9, 3, 2)]
install_path is called
00:00:00:00:00:34 -> 00:00:00:00:00:32 via 10 in_port= 2 out_port= 3
00:00:00:00:00:34 -> 00:00:00:00:00:32 via 11 in_port= 2 out_port= 1
00:00:00:00:00:34 -> 00:00:00:00:00:32 via 9 in_port= 3 out_port= 2
get_path is called, src= 10 dst= 9 first_port= 2 final_port= 2
Q= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
***Path: 10 : 2 To 9 : 2 is [(10, 2, 3), (11, 2, 1), (9, 3, 2)]
install_path is called
00:00:00:00:00:34 -> 00:00:00:00:00:32 via 10 in_port= 2 out_port= 3
00:00:00:00:00:34 -> 00:00:00:00:00:32 via 11 in_port= 2 out_port= 1
00:00:00:00:00:34 -> 00:00:00:00:00:32 via 9 in_port= 3 out_port= 2
get_path is called, src= 9 dst= 10 first_port= 2 final_port= 2
Q= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
***Path: 9 : 2 To 10 : 2 is [(9, 2, 3), (11, 1, 2), (10, 3, 2)]
install_path is called
00:00:00:00:00:32 -> 00:00:00:00:00:34 via 9 in_port= 2 out_port= 3
00:00:00:00:00:32 -> 00:00:00:00:00:34 via 11 in_port= 1 out_port= 2
00:00:00:00:00:32 -> 00:00:00:00:00:34 via 10 in_port= 3 out_port= 2
get_path is called, src= 9 dst= 10 first_port= 2 final_port= 2
Q= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
***Path: 9 : 2 To 10 : 2 is [(9, 2, 3), (11, 1, 2), (10, 3, 2)]
install_path is called
00:00:00:00:00:32 -> 00:00:00:00:00:34 via 9 in_port= 2 out_port= 3
00:00:00:00:00:32 -> 00:00:00:00:00:34 via 11 in_port= 1 out_port= 2
00:00:00:00:00:32 -> 00:00:00:00:00:34 via 10 in_port= 3 out_port= 2
get_path is called, src= 10 dst= 10 first_port= 2 final_port= 1
Q= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
***Path: 10 : 2 To 10 : 1 is [(10, 2, 1)]
install_path is called
00:00:00:00:00:34 -> 00:00:00:00:00:33 via 10 in_port= 2 out_port= 1
get_path is called, src= 10 dst= 10 first_port= 1 final_port= 2
Q= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
***Path: 10 : 1 To 10 : 2 is [(10, 1, 2)]
install_path is called
00:00:00:00:00:33 -> 00:00:00:00:00:34 via 10 in_port= 1 out_port= 2

Tehran, Iran