به نام او

پروژه پایانی شبکه (sdn)

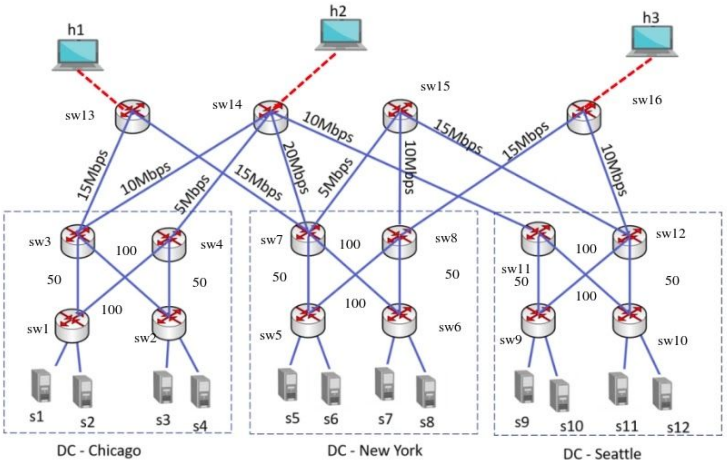
طاها شعبانی - ۸۱۰۱۹۶۴۹۱

هومان چمنی - ۸۱۰۱۹۶۴۴۳

مقدمه

هدف این پروژه، پیاده سازی یک کنترلر مبتنی بر کامپوننت به زبان پایتون و با استفاده از ابزار RYU است. RYU از پروتکل های مختلفی برای مدیریت دستگاه استفاده می کند که ما در این پروژه از پروتکل OpenFlow استفاده خواهیم کرد.
همچنین برای تست صحت عملکرد کنترلر از ابزار دیگری به نام mininet استفاده می کنیم که استفاده از این ابزار امکان ساخت سوییچ و هاست های مجازی را به ما می دهد.

در طول این پروژه چندین بار شبکه اصلی تغییر کرد. در نهایت تصمیم بر این شد که شبکه ای ساده سازی شده از شکل زیر استفاده شود که همه ی دور های آن حذف شده است (برای این کار همه ی یال ها با وزن ۱۰۰ و همچنین یال های (sw7 به sw13) و همچنین (sw12 به sw15) را حذف کردیم: در مجموع ۸ یال که کمترین میزان ممکن می باشد)



توضیحات اولیه

نحوه اجرای برنامه

برای اجرای کنترلر از دستور زیر استفاده می کنیم (*dijkstra_ryp.py اسم فایل کنترلر است*) :

```
ryu-manager dijkstra_ryp.py --observe-links
```

همچنین برای اجرای mininet نیز اگر بخواهیم از ساختار های آماده mininet برای مثال tree,3 استفاده کنیم، از دستور زیر بهره می بریم :

```
sudo mn --topo tree,3 --controller remote
```

و در صورتی که بخواهیم فایل توپولوژی[1] خودمان را به عنوان ورودی mininet استفاده کنیم، با استفاده از دستور زیر این کار امکان پذیر است (در صورتی که در فایل توپولوژی class تعریف کرده باشیم) :

```
sudo mn --custom topo.py --topo TopoExample
```

که topy.py همان فایل حاوی توپولوژی و TopoExample به این صورت در داخل فایل topo.py قابل تعریف است (در صورتی که در فایل توپولوژی class تعریف کرده باشیم) :

```
TOPOS = {'LinearTopo' : (lambda : TopoExample())}
```

همچنین کد فایل topo.py به صورت جدا نیز قابل اجراست (در صورتی که در فایل توپولوژی function صدا میزنیم: پایتون۲) :

```
sudo python topo.py
```

الگوریتم

الگوریتم استفاده شده در ساختار کنترلر و برای مسیریابی، الگوریتم دایجسترا[2] می باشد که کوتاه ترین مسیر بین دو نقطه را به ما می دهد.

[1] Topology

[2] Dijkstra

توضیح کد پروژه

خط به خط کد در داخل کامنت هایی که در داخل کد قرار داده ایم توضیح داده شده است اما برای مرور دوباره و توضیحات بیشتر دوباره این جا بیان می کنیم:

۱. در این خطوط که از ۱ تا ۳۱ می باشد، import های لازم از ryu و ... برای ساخت کنترلر را داریم.

```
from ryu.base import app_manager

from ryu.controller import mac_to_port

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.lib.mac import haddr_to_bin

from ryu.lib.packet import packet

from ryu.lib.packet import ethernet

from ryu.lib.packet import ether_types

from ryu.lib import mac


from ryu.topology.api import get_switch, get_link

from ryu.app.wsgi import ControllerBase

from ryu.topology import event, switches

from collections import defaultdict
```

۲. توضیحات به طور کامل در کامنت ها آمده است. ولی به اختصار :

Switches: لیست همه ی switch های توپولوژی فعلی که در تابع get_topology_data صدا زده می شود.

Mymac : یک dictionary که مقدار srcmac را (media access control address) را به ازای هر آدرس ethernet معتبر در شبکه، به مقدار switch که همان (data_port_id) آن است و port که port ارسالی است map می کند.

Adjacency : یک dictionary که تعیین میکند بین switch ها ارتباطی برقرار هست یا نه. اگر باشد شماره port و اگر نباشد مقدار none را ذخیره می کند. این مقادیر در تابع get_topology_data.

```
# switches
# list of all switches in our topology

switches = []



# mymac[srcmac]->(switch, port)
# MAC (media access control address) is a unique identifier assigned to a network
interface controller.
# MAC addresses are used in the medium access control protocol sublayer of the data
link layer.
# Here we use mac as a key for src of ethernet with values (switch, port) which switch
declares datapath_id and
# port declares in_port (in_port is declared inside message)

mymac={}



# adjacency map [sw1][sw2]->port from sw1 to sw2
# This means that if there is a port (portx) between switch1 and switch2, the value of
# adjacency[sw1][sw2] = portx, else it will be None.

adjacency=defaultdict(lambda:defaultdict(lambda:None))
```

۳. توضیح تابع minimum_distance :

در ابتدا مقدار min برابر ∞ قرار می دهیم. سپس در ادامه مقدار node که قرار است در نهایت node با کمترین distance باشد را برابر ۰ می گذاریم. بعد به ازای هر node موجود در مجموعه Q می آییم و آنی که کمترین مقدار distance را داراست پیدا کرده و در node ذخیره میکنیم. در نهایت node را return می کنیم.

```
def minimum_distance(distance, Q):
```

```
 min = float('Inf') # infinite number

 node = 0 # default value

 for v in Q: # checks in all nodes in Q(set) to find the minimum distance, and then
return it.

   if distance[v] < min:

     min = distance[v]

     node = v

 return node
```

۴. تابع get_path:

این تابع در داخل _packet_in_handler صدا زده می شود.

ورودی ها داخل کامنت ها توضیح داده شده اند

در ابتدا با استفاده از الگوریتم دایجسترا کوتاه ترین مسیر از src به همه node ها به دست می آوریم و در ادامه، مسیر از src به dst را در یک لیست به اسم path ذخیره کرده و در نهایت به ازای هر node در این مسیر in_port و out_port را نیز در r ذخیره کرده و r را return می کنیم.

```
def get_path (src,dst,first_port,final_port):
  # this function is called inside "_packet_in_handler()" which is called every
 # "EventOFPPacketIn" is triggered.
  # inputs {src: datapath id of current source mac | dst: datapath id of current
destination mac
 # first_port: port_in of current source mac | final_port: port_in of current source
mac}

 # Dijkstra's algorithm

 print ("get_path is called, src=",src," dst=",dst, " first_port=", first_port, "
final_port=", final_port)

 distance = {} # a dictionary to store distance from "src" to every node.

 previous = {} # just to handle dijkstra algorithm.
```

```
 for dpid in switches: # initialization for both "distance" = all "inf" and "previous"
= all "None"

   distance[dpid] = float('Inf')

   previous[dpid] = None


 distance[src]=0 # we want to measure distance from src node

 Q=set(switches) # set of all switches

 print ("Q=", Q)


 while len(Q)>0: # implementation of dijkstra: continue until you see all nodes in the
graph.

   u = minimum_distance(distance, Q) # u: node with smallest distance till now.

   Q.remove(u)



   for p in switches:

     if adjacency[u][p]!=None: #for all adjacent nodes with "u"

       w = 1 # default weight for edge: 1, because we assume that all edges (links)
have same time consumption ***bug*** : weights might not be all 1 and equal


       if distance[u] + w < distance[p]:

         # if it's possible to update "distance" for adjecents with "u"
         # with less values, then update it

         distance[p] = distance[u] + w

         previous[p] = u

 # after performing dijkstra :
```

```
r=[] # a list to store path from src to dst (in an inverse way: dst -> ... -> src)

p=dst # every node we want to add to path

r.append(p) # first node we add to r: dst

q=previous[p] # to store "previous node in path from dst to src which" mean "next in path from src to dst"

while q is not None:

  # while you haven't reached src

  if q == src: #src reached : end of while

    r.append(q)

    break

  p=q

  r.append(p) # appending to path

  q=previous[p]


r.reverse() # reverse "dst to src" to "src to dst"

if src==dst: # if path length is 1 (only src itself)

  path=[src]

else:

  path=r


# Now add the ports

r = []

in_port = first_port
```

```
 for s1,s2 in zip(path[:-1],path[1:]): #for example:(a,b,c,d) iterations:  (a,b),
(b,c), (c,d)

   # per every node s1 we store port from s1 to s2(next) and s1 to previous (it means
that for s2 previos is s1)

   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 # a list of (node, previous port path, next port in path)
```

۵. کلاس ProjectController : همان کنترلر که برای event های مختلف میخواهیم به وسیله آن، عملکرد درست را پیاده سازی کنیم.

۵.۱ تابع __init__ :

متغیر mac_to_port همانطور که در کامنت ها نیز مشخص شده است در جایی استفاده نشده است و حشو است و همینطور خط پایینی آن.

```
class ProjectController(app_manager.RyuApp):

   OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

   # Ryu supports openflow versions 1, 1.2, 1.3, 1.4, 1.5, so we store them in
"OFP_VERSIONS"

   def __init__(self, *args, **kwargs):

       super(ProjectController, self).__init__(*args, **kwargs)

       self.mac_to_port = {}
       # mac to port is a dictionary to map mac addresses to ports, but as we use
       # "mymac" dictionary instead, this dict:"mac_to_port" is useless.

       self.topology_api_app = self

       self.datapath_list=[] # list of all datapathes (every switch has a datapath)
```

```
    # Handy function that lists all attributes in the given object

    def ls(self,obj): # just a simple function to print info
        print("\n".join([x for x in dir(obj) if x[0] != "_"]))
```

۵.۲ تابع add_flow :

این تابع اضافی است و هیچ جایی در کد استفاده نشده است. اما توضیحات این که خط به خط چه کاری می کند به طور کامل در کامنت ها گفته شده است.

```
    def add_flow(self, datapath, in_port, dst, actions):
    # -> ***redundant*** : this function is never used

        ofproto = datapath.ofproto # declares openflow version

        parser = datapath.ofproto_parser # different versions have different apis, so
we use parser to use the correct one

        match = datapath.ofproto_parser.OFPMatch(in_port=in_port, eth_dst=dst)
        # there are different packets with different priorities, if current packet
doesn't match our priorities
        # by default it will be sent to the controller to be handled.


        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)]
        # inst consists of "an action and a mode" used as a prerequisite in sending
message

        # here we set priority to "OFP_DEFAULT_PRIORITY = 32768"
        mod = datapath.ofproto_parser.OFPFlowMod(

            datapath=datapath, match=match, cookie=0,

            command=ofproto.OFPFC_ADD, idle_timeout=0, hard_timeout=0,

            priority=ofproto.OFP_DEFAULT_PRIORITY, instructions=inst)

        # in order to modify "table flow" of a switch we build an "OFPFlowMod" object
and pass
        # match, inst, priority of packets, and datapath to it.
```

```
        datapath.send_msg(mod) #sending message with mod information
```

۵.۳ تابع install_path :

این تابع در داخل تابع _packet_in_handler صدا زده می شود وقتی که destination در داخل path فعلی مشخص باشد (در داخل mymac تعریف شده باشد).

هدف این تابع ارسال message از src تا dst است که مشابه تابع add_flow ابتدا match و inst را set میکند (توضیح این موارد داخل کد است) و سپس message را انتقال می دهد.

```
    def install_path(self, p, ev, src_mac, dst_mac):

        # is called inside "_packet_in_handler()" when destination is valid inside
"mymac"
        # this function sends message through the path from src to dst

        # inputs = {p: path from src to dst | ev: event that triggers
"_packet_in_handler()"
        # src_mac: source of path | dst_mac: destination of path}

        print ("install_path is called")


        msg = ev.msg # message of current event

        datapath = msg.datapath
        # datapath of current message which consists of attributes which declares
information about msg
        # such as id of sender, ofproto which is related to version and ...

        ofproto = datapath.ofproto # declares openflow version

        parser = datapath.ofproto_parser #different versions have different apis, so we
use parser to use the correct one

        for sw, in_port, out_port in p: # sends msg through the path

         match=parser.OFPMatch(in_port=in_port, eth_src=src_mac, eth_dst=dst_mac)
         # creates its match (described in "add_flow()")

         actions=[parser.OFPActionOutput(out_port)]
```

```
        # declares openflow action (says what to do after matching)

        for each in self.datapath_list:
          if each.id == sw:
            datapath = each

        # datapath=self.datapath_list[int(sw)-1]
        # -> ***bug:*** this part of code had a bug so we replaced it : sw value may
not be equal to datapath_list index

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS , actions)]
        # creates its inst ( described in "add_flow()")


        #here we set priority = 1
        mod = datapath.ofproto_parser.OFPFlowMod(

        datapath=datapath, match=match, idle_timeout=0, hard_timeout=0,

        priority=1, instructions=inst)
        # creates its inst ( described in "add_flow()")

        datapath.send_msg(mod)
```

۵.۴ تابع switch_features_handler که با event:EventOFPSwitchFeatures صدا زده می شود.

این event هر موقعی که یک message بین switch و controller منتقل شود صدا زده می شود. این message که در قالب req/res کار می کند بر روی پروتکل OFP کار می کند و حاوی یک message است.

در واقع این event هر موقعی که میخواهیم یک switch به شبکه اضافه کنیم صدا زده می شود.

سایر توضیحات در داخل کامنت ها موجود است، تنها نکته ای که به علت اهمیت دوباره این جا نیز توضیح می دهم این است که در ارسال، priority آن را برابر ۰ یعنی کمترین مقدار قرار می دهیم که متعلق به مواردی است که ارجاع به controller داریم مثل زمانی که اگر در ارسال message در تابع install path یا add_flow خود match صورت نگیرد از این اولویت استفاده می شود تا توسط کنترلر هندل شود. (توضیح در کامنت های تابع add_flow)

```
  @set_ev_cls(ofp_event.EventOFPSwitchFeatures , CONFIG_DISPATCHER)

  def switch_features_handler(self , ev):

      # every message transfer between switch and controller triggers an action which
calls this function
```

```
        # these requests and response works over openflowProtocol and contains a
message
        # every time a switch enters the network, this event triggers.

        print ("switch_features_handler is called")

        datapath = ev.msg.datapath # datapath of current message (described in
"install_path")

        # all the beneath declarations described in "install_path"
        ofproto = datapath.ofproto

        parser = datapath.ofproto_parser

        match = parser.OFPMatch()

        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
ofproto.OFPCML_NO_BUFFER)]

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS ,actions)]

        # here we set priority = 0 to set a minimum priority for the times packet
priority could not be found
        # and we should sent it to controller

        mod = datapath.ofproto_parser.OFPFlowMod(

        datapath=datapath, match=match, cookie=0,

        command=ofproto.OFPFC_ADD, idle_timeout=0, hard_timeout=0,

        priority=0, instructions=inst)

        datapath.send_msg(mod)
```

۵.۵ تابع packet_in_handler_:

این تابع با event:EventOFPPacketIn فراخوانی می شود که هر موقعی packet ای دریافت شود صدا زده می شود.

توضیحات declaration ها در این تابع و توابع قبلی به طور کامل در کامنت ها توضیح داده شده است.

اما موارد مهم تر:

مقدار in_port همان port ای است که پیام را به switch فعلی رسانده است.

همچنین مقادیر pkt و eth نیز در داخل کامنت ها توضیح داده شده اند.
در ادامه از broadcast توسط پروتکل lldp که مخفف Link Layer Discovery Protocol است جلوگیری شده است زیرا امکان گیر کردن در حلقه ها تا بی نهایت را ممکن می سازند. (اگر شبکه حلقه داشته باشد)
سایر ارسال ها نظیر ارسال از طریق ipv4, ipv6, icmp, arp مشکلی ندارد و به طور عمده به جز lldp از طریق ipv4 ارسال می شود که در بخش اسکرین شات ها در ادامه قابل مشاهده است.
در ادامه با استفاده از ethernet مقادیر src و dst و datapath_id ست می شود و چک می شود اگر src جدید بود در mymac اضافه شود و در نهایت اگر dst معتبر بود با استفاده از تابع install_path ارسال ها در داخل مسیر صورت بگیرد و در غیر این صورت مقدار flood را برایش ست میکنیم که به این معنا است که پیام باید broadcast شود (به همه ی switch ها به جز switch ارسال کننده ارسال شود)
در ادامه message چک میشود تا اگر به صورت یک packet قابل ارسال بود از طریق همان یک packet و اگر نمیشد به صورت بافر شده ارسال شود.
در نهایت packet خروجی ساخته می شود (با استفاده از کلاس OFPPacketOut در parser) و از طریق datapath فعلی ارسال می شود.
(توضیحات دقیق تر در کامنت ها)

```
    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)

    def _packet_in_handler(self, ev):

        # this function is called every time a packer is received (using
"EventOFPPacketIn" to check)


        #these declarations described previously
        msg = ev.msg

        datapath = msg.datapath

        ofproto = datapath.ofproto

        parser = datapath.ofproto_parser


        in_port = msg.match['in_port']
        # sets in_port: port which sent current packet to this switch (receiver switch)


        pkt = packet.Packet(msg.data)
```

```
        # fetching packet from msg

        eth = pkt.get_protocol(ethernet.ethernet)
        # fetching ethernet from packet protocol



        # avoid broadcast from LLDP

        # The Link Layer Discovery Protocol is a vendor-neutral link layer protocol
used by network devices for advertising
        # their identity, capabilities, and neighbors on a local area network based on
IEEE 802 technology,
        # principally wired Ethernet.

        # In order to avoid cyclic and infinite sends, we avoid broadcasting LLDP
        if eth.ethertype==35020:
            return


        dst = eth.dst
        # destination of ethernet

        src = eth.src
        # source of ethernet

        dpid = datapath.id
        # setting datapath id to use as switch inside mymac

        self.mac_to_port.setdefault(dpid, {})
        # if dpid was not valid, return an empty dict instead
        # -> ***redundant*** : this function is useless and does nothing

        if src not in mymac.keys(): # if src is new, add it in mymac (key: src, dpid:
switch, in_port: port)
            mymac[src]=( dpid,  in_port)


        if dst in mymac.keys():
            # if destination is valid, then call "get_path()" to get best path using
dijkstra
            # then call "install_path()" to send message through path
```

```
            p = get_path(mymac[src][0], mymac[dst][0], mymac[src][1], mymac[dst][1])

            print (p)

            self.install_path(p, ev, src, dst)

            out_port = p[0][2]

        else:
            #if destination is not valid, broadcast it using flooding (send to all
except for sender)
            out_port = ofproto.OFPP_FLOOD

        actions = [parser.OFPActionOutput(out_port)]
        # set action based on out_port



        # install a flow to avoid packet_in next time
        if out_port != ofproto.OFPP_FLOOD:
            # if out_port was not set for flooding, set match (match described in
"install_path()")
            match = parser.OFPMatch(in_port=in_port, eth_src=src, eth_dst=dst)
            # -> ***redundant*** : "match" is never used



        data=None

        #if message is not buffered and it could be send via 1 packet, then set data,
otherwise data would be None
        # and the message would be sent through another way.
        if msg.buffer_id==ofproto.OFP_NO_BUFFER:
           data=msg.data

        # OFPPacketOut class is used to build a packet_out message.
        out = parser.OFPPacketOut(

            datapath=datapath, buffer_id=msg.buffer_id, in_port=in_port,

            actions=actions, data=data)

        #sends message
        datapath.send_msg(out)
```

۵.۶ تابع get_topology_data:

ابتدا مقادیر switch_list, switches, datapath_list, links_list, mylins ست می شود که همگی در داخل کامنت ها توضیح داده شده اند. پس از آن به ازای هر لینک معتبر در شبکه بین سوئیچ ها، adjacency map ست می شود که قبلا توضیح داده شده است به چه کار میآید (هم در کامنت ها و هم در توابع قبلی: در الگوریتم دایجسترا به کار می آید.)

```
    @set_ev_cls(event.EventSwitchEnter)
    # The event EventSwitchEnter will trigger the activation of get_topology_data().
    # there are EventSwitchEnter and EventSwitchLeave events. here we used
EventSwitchEnter
    # to get all links. Links on a switch will be discovered after Ryu found that
switch.

    def get_topology_data(self, ev):

        global switches

        switch_list = get_switch(self.topology_api_app, None)
        # fetching all switches from current topology

        switches=[switch.dp.id for switch in switch_list]
        # fetching all datapath ids from switch_list

        self.datapath_list=[switch.dp for switch in switch_list]
        # fetching all datapaths from switch_list


        print ("switches=", switches)


        links_list = get_link(self.topology_api_app, None)
        #fetch all links from topology

        mylinks=[(link.src.dpid,link.dst.dpid,link.src.port_no,link.dst.port_no) for
link in links_list]
        # per every link in links_list, store datapath_id of src and dst and
        # port number of src and dst in mylinks (important values)

        for s1,s2,port1,port2 in mylinks: #fill the adjacency map with valid values
(used in dijkstra later, and ...)
```

```
        adjacency[s1][s2]=port1


        adjacency[s2][s1]=port2
```

اشکالات[3] کد پروژه

باگ ها

(**نکته**: همه ی باگ ها با ***bug*** در کامنت ها مشخص شده اند)

۱. در تابع get_path و برای محاسبه الگوریتم دایجسترا همه ی وزن ها برابر ۱ در نظر گرفته شده اند که لزوما این مورد درست نیست. البته به علت اینکه ما دور ها را حذف کردیم و برای حذف دور ها یک سری یال با وزن های متفاوت حذف شدند، شبکه ای که ماند همه وزن ها برابرند و بنابراین برای توپولوژی ما w=1 هم کفایت می کند.

۲. در تابع packet_in_handler_ که توسط event:EventOFPPacketIn صدا زده می شود (این موارد در بخش توضیح کد و کامنت ها توضیح داده شده است.)

علت: در خط ۲۸۶ که در ابتدا اجرا می شد با این پیشفرض عمل می کرد که مقدار عددی switch با شماره خانه در لیست datapath_list برابر است، در حالی که لزومی ندارد این مسئله درست باشد بنابراین خط های ۲۸۲ تا ۲۸۴ اضافه شد که در datapath_list بگردد و مقدار خواسته شده را پیدا کند.

```
282         for dpi in self.datapath_list:
283           if dpi.id == sw:
284             datapath = dpi
285 
286         # datapath=self.datapath_list[int(sw)-1]
287         # -> ***bug:*** this part of code had a bug so we replaced it : sw value may not be equal to datapath_list index
288 
289         inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS , actions)]
290         # creates its inst ( described in "add_flow()")
```

حشو ها[4]

(**نکته**: همه ی حشو ها با ***redundant*** در کامنت ها مشخص شده اند)

۱. تابع add_flow در داخل کلاس ProjectController هیچگاه استفاده نشده است.

۲. متغیر mac_to_port که هم در تابع packet_in_handler_ و هم تابع __init__ از کلاس ProjectController تعریف شده است هیچگاه استفاده نشده است و به جای آن از متغیر mymac استفاده می کنیم.

۳. متغیر match در تابع packet_in_handler_ از کلاس ProjectController بعد از تعریف هیچگاه استفاده نشده است.

[3] Bugs and redundancies

[4] Redundancies

اجرای کد و اسکرین شات ها

پس از ساخت فایل توپولوژی و همچنین کنترلر. باید آن ها را اجرا کنیم :

اجرای کنترلر : ryu-manager dijkstra_ryp.py --observe-link

اجرای توپولوژی : sudo python topo.py

نکته: برای درست اجرا شدن ابتدا باید فایل topology و سپس فایل کنترلر اجرا شود.

۱. در ابتدای اجرای کنترلر :

```
loading app dijkstra_ryp.py
loading app ryu.controller.ofp_handler
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app dijkstra_ryp.py of ProjectController
... controller created ...

instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu.topology.switches of Switches
```

۱. و در ابتدای اجرای mininet (فایل توپولوژی) :

```
*** Adding controller
*** Add switches
*** Add hosts
*** Add links
(15.00Mbit) (15.00Mbit) (5.00Mbit) (5.00Mbit) (15.00Mbit) (15.00Mbit) (20.00Mbit) (20.00Mbit) (5.00Mbit) (
5.00Mbit) (10.00Mbit) (10.00Mbit) (15.00Mbit) (15.00Mbit) (10.00Mbit) (10.00Mbit) (10.00Mbit) (10.00Mbit)
*** Starting network
*** Configuring hosts
h1 h2 h3 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12
*** Starting controllers
*** Starting switches
(15.00Mbit) (5.00Mbit) (15.00Mbit) (20.00Mbit) (5.00Mbit) (10.00Mbit) (15.00Mbit) (10.00Mbit) (10.00Mbit)
(15.00Mbit) (15.00Mbit) (5.00Mbit) (20.00Mbit) (10.00Mbit) (5.00Mbit) (10.00Mbit) (15.00Mbit) (10.00Mbit)
*** Post configure switches and hosts
*** Starting CLI:
```

۲. سپس بعد از اتصال این دو در بخش کنترلر خواهیم داشت :

```
... controller created ...

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
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
switch_features_handler is called
switch_features_handler is called
get_topology_data() is called
switches= [7, 8, 16, 14, 3, 6, 15, 4, 2, 12, 13, 11]
get_topology_data() is called
switches= [7, 8, 16, 14, 3, 6, 15, 4, 2, 12, 13, 11, 10, 1, 9, 5]
get_topology_data() is called
switches= [7, 8, 16, 14, 3, 6, 15, 4, 2, 12, 13, 11, 10, 1, 9, 5]
get_topology_data() is called
switches= [7, 8, 16, 14, 3, 6, 15, 4, 2, 12, 13, 11, 10, 1, 9, 5]
get_topology_data() is called
switches= [7, 8, 16, 14, 3, 6, 15, 4, 2, 12, 13, 11, 10, 1, 9, 5]
get_topology_data() is called
switches= [7, 8, 16, 14, 3, 6, 15, 4, 2, 12, 13, 11, 10, 1, 9, 5]
get_topology_data() is called
switches= [7, 8, 16, 14, 3, 6, 15, 4, 2, 12, 13, 11, 10, 1, 9, 5]
get_topology_data() is called
switches= [7, 8, 16, 14, 3, 6, 15, 4, 2, 12, 13, 11, 10, 1, 9, 5]
get_topology_data() is called
switches= [7, 8, 16, 14, 3, 6, 15, 4, 2, 12, 13, 11, 10, 1, 9, 5]
get_topology_data() is called
switches= [7, 8, 16, 14, 3, 6, 15, 4, 2, 12, 13, 11, 10, 1, 9, 5]
get_topology_data() is called
switches= [7, 8, 16, 14, 3, 6, 15, 4, 2, 12, 13, 11, 10, 1, 9, 5]
get_topology_data() is called
switches= [7, 8, 16, 14, 3, 6, 15, 4, 2, 12, 13, 11, 10, 1, 9, 5]
get_topology_data() is called
switches= [7, 8, 16, 14, 3, 6, 15, 4, 2, 12, 13, 11, 10, 1, 9, 5]
get_topology_data() is called
switches= [7, 8, 16, 14, 3, 6, 15, 4, 2, 12, 13, 11, 10, 1, 9, 5]
get_topology_data() is called
switches= [7, 8, 16, 14, 3, 6, 15, 4, 2, 12, 13, 11, 10, 1, 9, 5]
get_topology_data() is called
switches= [7, 8, 16, 14, 3, 6, 15, 4, 2, 12, 13, 11, 10, 1, 9, 5]
```

که در این مرحله switch دونه دونه به شبکه اضافه شده و به ازای هر دفعه صدا شدن تابع get_topology_data اطلاعات آن را نشان می دهیم.

۳. برای گرفتن همه ی ping ها و بررسی ارسال ها از دستور pingall استفاده می کنیم که در نهایت همچین خروجی خواهد داشت:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 s1 s2 s3 s4 s5 X X X s9 s10 X s12
h2 -> h1 X s1 s2 X X X X X X X X X
h3 -> h1 h2 s1 s2 s3 X s5 s6 X X s9 s10 s11 X
s1 -> h1 h2 h3 s2 s3 s4 s5 s6 X X s9 s10 X s12
s2 -> h1 h2 h3 s1 X s4 X X s7 X s9 s10 X X
s3 -> h1 h2 h3 s1 s2 s4 X X X X X X X X
s4 -> h1 h2 h3 s1 s2 s3 X s6 X X X X X X
s5 -> h1 h2 h3 s1 s2 s3 s4 s6 X X s9 s10 s11 s12
s6 -> h1 h2 h3 s1 s2 s3 s4 s5 X X s9 s10 X s12
s7 -> h1 h2 h3 s1 s2 s3 s4 s5 s6 s8 X 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 X X
s10 -> h1 h2 h3 s1 s2 s3 s4 s5 s6 s7 s8 s9 X 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: 24% dropped (158/210 received)
```

که reachability برابر با ۷۶ درصد بوده و دسترسی ها نیز به طور کامل نشان داده شده اند.

۳.۱ همچنین در حین ارسال پکت ها در طول مسیر ما اطلاعات آن را نیز چاپ میکنیم که این یک نمونه آن است :

```
install_path is called
start sending message through path:
14 11 9 ... end of path ...
get_path() is called, src= 14  dst= 9  first_port= 1  final_port= 1
Q= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
... minimum distance function is called ...
... minimum distance function is called ...
... minimum distance function is called ...
... minimum distance function is called ...
... minimum distance function is called ...
... minimum distance function is called ...
... minimum distance function is called ...
... minimum distance function is called ...
... minimum distance function is called ...
... minimum distance function is called ...
... minimum distance function is called ...
... minimum distance function is called ...
... minimum distance function is called ...
... minimum distance function is called ...
... minimum distance function is called ...
... minimum distance function is called ...
... path:  [(14, 1, 4), (11, 2, 1), (9, 3, 1)]

[(14, 1, 4), (11, 2, 1), (9, 3, 1)]
```

که برای مثال در این مرحله مسیر از sw14 به sw9 است و مجموعه switch ها از ۱ تا ۱۶ هستند که در Q نگهداری می شود و به ازای هر کدام از این ۱۶ تا تابع minimum distance در الگوریتم dijkstra صدا زده می شود و در نهایت نیز مسیر به صورت : 14-11-9 خواهد بود. (مقادیر سه تایی در path در کد و در بخش get_path هم در کامنت و هم در این داکیومنت توضیح داده شده است -رجوع به بخش توضیح کد-)

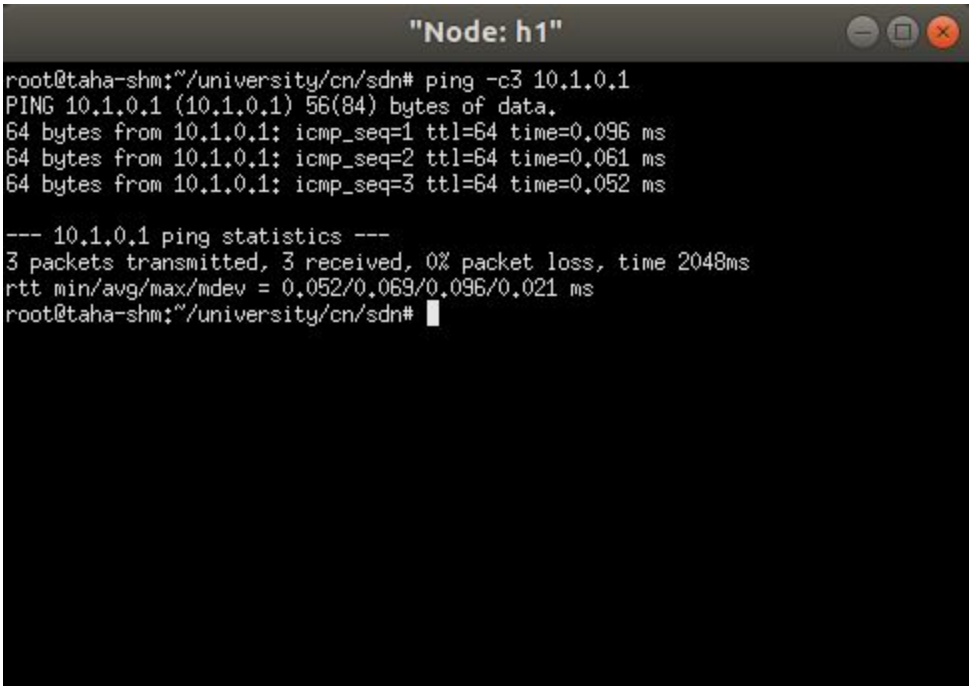
۴. همچنین برای مشاهده ping (مدت زمان ارسال پکت و دریافت جواب) بین دو host می شود از دستور زیر استفاده کرد :

```
mininet> h1 ping h2
PING 10.1.0.2 (10.1.0.2) 56(84) bytes of data.
64 bytes from 10.1.0.2: icmp_seq=1 ttl=64 time=1.22 ms
64 bytes from 10.1.0.2: icmp_seq=2 ttl=64 time=0.099 ms
64 bytes from 10.1.0.2: icmp_seq=3 ttl=64 time=0.098 ms
64 bytes from 10.1.0.2: icmp_seq=4 ttl=64 time=0.103 ms
64 bytes from 10.1.0.2: icmp_seq=5 ttl=64 time=0.100 ms
64 bytes from 10.1.0.2: icmp_seq=6 ttl=64 time=0.129 ms
64 bytes from 10.1.0.2: icmp_seq=7 ttl=64 time=0.110 ms
64 bytes from 10.1.0.2: icmp_seq=8 ttl=64 time=0.097 ms
```

۴.۱ همین جزییات رو با استفاده از xterm نیز می شود مشاهده کرد:

(که 10.1.0.1 همان آدرس ip مخصوص h1 است که در فایل topo.py تعریف کرده ایم.

```
mininet> xterm h1
```



۴.۲ همچنین اگر در زمانی که داریم ping بین دو هاست را بررسی میکنیم در xterm عبارت tcpdump -n را بزنیم همان جزییات را به طور دقیق تری نشان می دهد:

```
root@taha-shm:~/university/cn/sdn# tcpdump -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h1-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
19:40:17.343181 IP 10.1.0.1 > 10.1.0.2: ICMP echo request, id 11482, seq 9, leng
th 64
19:40:17.343266 IP 10.1.0.2 > 10.1.0.1: ICMP echo reply, id 11482, seq 9, length
 64
19:40:17.710489 LLDP, length 46
19:40:18.367184 IP 10.1.0.1 > 10.1.0.2: ICMP echo request, id 11482, seq 10, len
gth 64
19:40:18.367269 IP 10.1.0.2 > 10.1.0.1: ICMP echo reply, id 11482, seq 10, lengt
h 64
19:40:19.391178 IP 10.1.0.1 > 10.1.0.2: ICMP echo request, id 11482, seq 11, len
gth 64
19:40:19.391266 IP 10.1.0.2 > 10.1.0.1: ICMP echo reply, id 11482, seq 11, lengt
h 64
19:40:20.006947 LLDP, length 46
19:40:20.415069 IP 10.1.0.1 > 10.1.0.2: ICMP echo request, id 11482, seq 12, len
gth 64
19:40:20.415169 IP 10.1.0.2 > 10.1.0.1: ICMP echo reply, id 11482, seq 12, lengt
h 64
```

۵. برای مشاهده جزئیات شبکه می توان از کامند هایی نظیر net, dump, nodes استفاده کرد که این کامند ها برای شبکه ما به خوبی کار کرده و نشان میدهد به درستی ساخته شده است.

```
mininet> dump
<Host h1: h1-eth0:10.1.0.1 pid=8414>
<Host h2: h2-eth0:10.1.0.2 pid=8416>
<Host h3: h3-eth0:10.1.0.3 pid=8418>
<Host s1: s1-eth0:10.0.0.1 pid=8420>
<Host s2: s2-eth0:10.0.0.2 pid=8422>
<Host s3: s3-eth0:10.0.0.3 pid=8424>
<Host s4: s4-eth0:10.0.0.4 pid=8426>
<Host s5: s5-eth0:10.0.1.1 pid=8428>
<Host s6: s6-eth0:10.0.1.2 pid=8430>
<Host s7: s7-eth0:10.0.1.3 pid=8432>
<Host s8: s8-eth0:10.0.1.4 pid=8436>
<Host s9: s9-eth0:10.0.2.1 pid=8438>
<Host s10: s10-eth0:10.0.2.2 pid=8442>
<Host s11: s11-eth0:10.0.2.3 pid=8444>
<Host s12: s12-eth0:10.0.2.4 pid=8446>
<OVSSwitch sw1: lo:127.0.0.1,sw1-eth1:None,sw1-eth2:None,sw1-eth3:None pid=8364>
<OVSSwitch sw2: lo:127.0.0.1,sw2-eth1:None,sw2-eth2:None,sw2-eth3:None pid=8367>
<OVSSwitch sw3: lo:127.0.0.1,sw3-eth1:None,sw3-eth2:None pid=8370>
<OVSSwitch sw4: lo:127.0.0.1,sw4-eth1:None,sw4-eth2:None pid=8373>
<OVSSwitch sw5: lo:127.0.0.1,sw5-eth1:None,sw5-eth2:None,sw5-eth3:None pid=8376>
<OVSSwitch sw6: lo:127.0.0.1,sw6-eth1:None,sw6-eth2:None,sw6-eth3:None pid=8379>
<OVSSwitch sw7: lo:127.0.0.1,sw7-eth1:None,sw7-eth2:None,sw7-eth3:None,sw7-eth4:None pid=8382>
<OVSSwitch sw8: lo:127.0.0.1,sw8-eth1:None,sw8-eth2:None,sw8-eth3:None pid=8385>
<OVSSwitch sw9: lo:127.0.0.1,sw9-eth1:None,sw9-eth2:None,sw9-eth3:None pid=8388>
<OVSSwitch sw10: lo:127.0.0.1,sw10-eth1:None,sw10-eth2:None,sw10-eth3:None pid=8391>
<OVSSwitch sw11: lo:127.0.0.1,sw11-eth1:None,sw11-eth2:None pid=8394>
<OVSSwitch sw12: lo:127.0.0.1,sw12-eth1:None,sw12-eth2:None pid=8397>
<OVSSwitch sw13: lo:127.0.0.1,sw13-eth1:None,sw13-eth2:None,sw13-eth3:None pid=8400>
<OVSSwitch sw14: lo:127.0.0.1,sw14-eth1:None,sw14-eth2:None,sw14-eth3:None,sw14-eth4:None pid=8403>
<OVSSwitch sw15: lo:127.0.0.1,sw15-eth1:None,sw15-eth2:None pid=8406>
<OVSSwitch sw16: lo:127.0.0.1,sw16-eth1:None,sw16-eth2:None,sw16-eth3:None pid=8409>
<RemoteController c0: 127.0.0.1:6633 pid=8357>
```

```
mininet> net
h1 h1-eth0:sw13-eth1
h2 h2-eth0:sw14-eth1
h3 h3-eth0:sw16-eth1
s1 s1-eth0:sw1-eth1
s2 s2-eth0:sw1-eth2
s3 s3-eth0:sw2-eth1
s4 s4-eth0:sw2-eth2
s5 s5-eth0:sw5-eth1
s6 s6-eth0:sw5-eth2
s7 s7-eth0:sw6-eth1
s8 s8-eth0:sw6-eth2
s9 s9-eth0:sw9-eth1
s10 s10-eth0:sw9-eth2
s11 s11-eth0:sw10-eth1
s12 s12-eth0:sw10-eth2
sw1 lo:  sw1-eth1:s1-eth0 sw1-eth2:s2-eth0 sw1-eth3:sw3-eth1
sw2 lo:  sw2-eth1:s3-eth0 sw2-eth2:s4-eth0 sw2-eth3:sw4-eth1
sw3 lo:  sw3-eth1:sw1-eth3 sw3-eth2:sw13-eth2
sw4 lo:  sw4-eth1:sw2-eth3 sw4-eth2:sw14-eth2
sw5 lo:  sw5-eth1:s5-eth0 sw5-eth2:s6-eth0 sw5-eth3:sw7-eth1
sw6 lo:  sw6-eth1:s7-eth0 sw6-eth2:s8-eth0 sw6-eth3:sw8-eth1
sw7 lo:  sw7-eth1:sw5-eth3 sw7-eth2:sw13-eth3 sw7-eth3:sw14-eth3 sw7-eth4:sw15-eth1
sw8 lo:  sw8-eth1:sw6-eth3 sw8-eth2:sw15-eth2 sw8-eth3:sw16-eth2
sw9 lo:  sw9-eth1:s9-eth0 sw9-eth2:s10-eth0 sw9-eth3:sw11-eth1
sw10 lo:  sw10-eth1:s11-eth0 sw10-eth2:s12-eth0 sw10-eth3:sw12-eth1
sw11 lo:  sw11-eth1:sw9-eth3 sw11-eth2:sw14-eth4
sw12 lo:  sw12-eth1:sw10-eth3 sw12-eth2:sw16-eth3
sw13 lo:  sw13-eth1:h1-eth0 sw13-eth2:sw3-eth2 sw13-eth3:sw7-eth2
sw14 lo:  sw14-eth1:h2-eth0 sw14-eth2:sw4-eth2 sw14-eth3:sw7-eth3 sw14-eth4:sw11-eth2
sw15 lo:  sw15-eth1:sw7-eth4 sw15-eth2:sw8-eth2
sw16 lo:  sw16-eth1:h3-eth0 sw16-eth2:sw8-eth3 sw16-eth3:sw12-eth2
```

```
mininet> nodes
available nodes are:
c0 h1 h2 h3 s1 s10 s11 s12 s2 s3 s4 s5 s6 s7 s8 s9 sw1 sw10 sw11 sw12 sw13 sw14 sw15 sw16 sw2 sw3 sw4 sw5 sw6 sw7 sw8 sw9
```

۶. برای مشاهده پهنای باند[5] ها نیز از دستور iperf استفاده می کنیم :

```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and s12
.*** Results: ['4.78 Mbits/sec', '5.31 Mbits/sec']
```

که همانطور که در شکل بخش مقدمه مشخص شده است بین h1 و s12 کمینه مقدار پهنای باند برابر 5Mbps است (از مسیر sw7 به sw15) بنابراین همان حدود نیز محاسبه می شود.

پایان...

[5] Bandwidth