



دانشگاه تهران

دانشکده برق و کامپیوتر

شبکه های کامپیوتری مسیریابی در شبکه با کنترلر RYU

اعضای گروه:

امید بداقی
مبینا شاه بند
عرفان وهابی

دکتر خونساری

تابستان ۹۹

فهرست مطالب

2	مقدمه
3	توضیح نحوه کار کد الگوریتم کوتاه ترین مسیر
14	توضیح نحوه کار کد کنترل RYU
37	اصلاحات کد
42	نتایج
42	توپولوژی درختی با عمق 3
49	توپولوژی صورت پروژه فعلی با حذف دور
58	توپولوژی صورت پروژه قبلی با حذف دور (امتیازی)
65	پهنای باند تصادفی (امتیازی)

مقدمه

هدف از این پروژه پیاده سازی مسیریابی توسط الگوریتم دایکسترا روی کنترلر RYU و با استفاده از mininet است. در ادامه شرح مفصل بخش های کد این پیاده سازی آمده است.

توضیح نحوه کار کد الگوریتم کوتاه ترین مسیر

```
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
```

در ابتدای کد توابع مورد نیاز از این کنترلر import شده اند.

```
#switches  
  
switches = []  
  
#mymac[srcmac]->(switch, port)  
mymac={}  
  
#adjacency map [sw1][sw2]->port from sw1 to sw2  
adjacency=defaultdict(lambda:defaultdict(lambda:None))
```

لیستی که در آن بعدا سوئیچ ها ذخیره می شوند، ساخته شده است. پس از آن دیکشنری ای که در آن MAC آدرس ها ذخیره می شوند، ساخته شده و در آخر هم یک دیکشنری با مقدار اولیه `None` ساخته شده که در واقع نشان دهنده ی گراف متناظر با شبکه می باشد. یعنی در صورتی که خانه `i` `adjacency[a][b]` خالی نباشد، نشان می دهد که از `a` به `b` لینک وجود دارد و مقدار این خانه از دیکشنری هم برابر با پورت متناظر می باشد.



```
def minimum_distance(distance, Q):  
    min = float('Inf')  
    node = 0  
  
    for v in Q:  
        if distance[v] < min:  
            min = distance[v]  
            node = v  
  
    return node
```

این تابع لیستی از گره ها را گرفته و بین آنها گره ای را پیدا می کند که فاصله آن که در دیکشنری `distance` ذخیره شده، کمترین باشد. در ادامه توضیحات نحوه فراخوانی و کاربرد این تابع آمده است.

●

●

●

```
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  
  
    distance = {}  
  
    previous = {}
```

از این تابع برای پیدا کردن کوتاه ترین مسیر بین دو گره مبدأ و مقصد استفاده می شود. ورودی های این تابع گره مبدأ، گره مقصد، پورت ورودی سمت مقصد و پورت خروجی سمت مبدأ هستند. در ابتدای بدنه این تابع، یک دیکشنری با نام distance ساخته شده است. این دیکشنری کوتاه ترین فاصله بین هر دو گره را در خود ذخیره می کند. سپس یک دیکشنری دیگری با نام previous ساخته شده است. این دیکشنری برای ذخیره سازی مسیری که کوتاه ترین مسیر بدست آمده بین دو گره مبدأ و مقصد است، استفاده می شود؛ به این صورت که برای هر گره، گرهی قبلی اش در مسیر را ذخیره می کنیم.

```
for dpid in switches:  
    distance[dpid] = float('Inf')  
    previous[dpid] = None
```

حال برای تمام گره ها (که در این مدل مسیریابی همان سوئیچ ها هستند) مقدار فاصله‌ی بی نهایت را نسبت می دهیم؛ زیرا در اجرای الگوریتم هر بار فاصله‌ی جدید با فاصله‌ی قبلی مقایسه می شود و در صورت کمتر بودن، جایگزین می شود. بنابراین در ابتدا باید بزرگ‌ترین عدد ممکن را به آن نسبت دهیم تا در اولین بار مقایسه جایگزین شود.

سپس گره قبلی هر گره را `None` نسبت می دهیم؛ زیرا در ابتدا مسیری پیدا نشده است و هیچ گره ای در مسیر مورد نظر پدر (گره پیشین) ندارد.

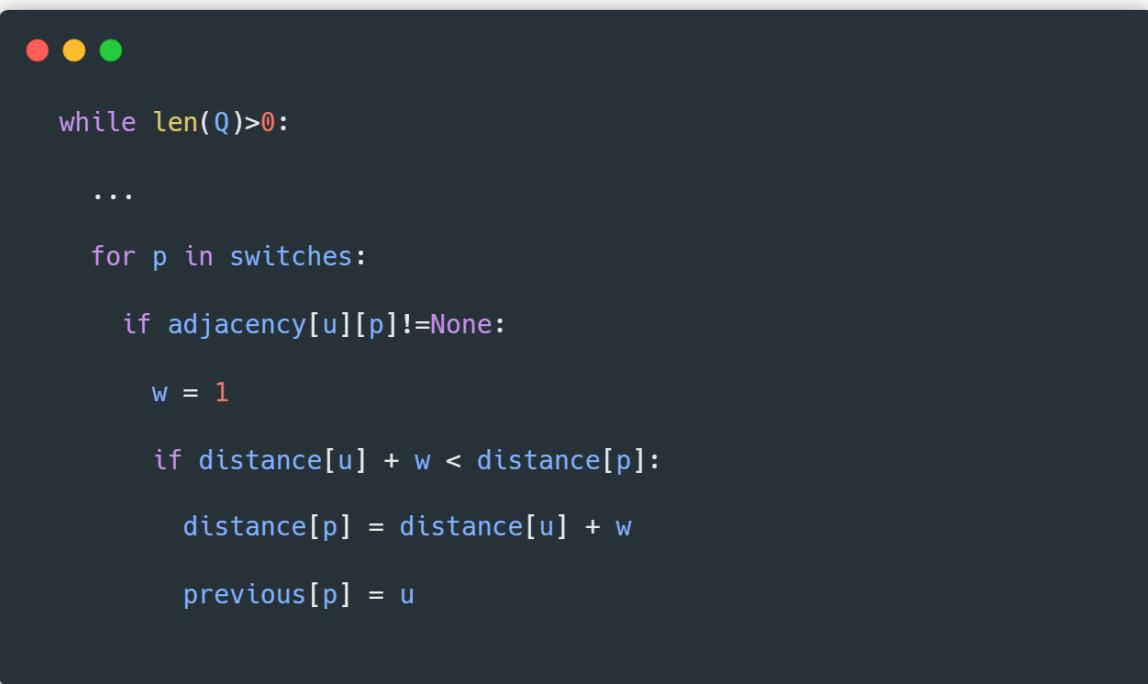
```
distance[src]=0  
Q=set(switches)  
print "Q=", Q
```

حال فاصله مربوط به گره مبدأ را 0 قرار می دهیم؛ زیرا گره مبدأ گره آغازین است و فاصله آن باید 0 نسبت داده شود. پس از آن یک کپی از لیست سوئیچ ها می سازیم. علت آنست که در ادامه اجرای الگوریتم کوتاه ترین مسیر، این لیست تغییر می کند؛ اما ما نمی خواهیم که لیست اصلی گره ها تغییر کند. پس از لیست اصلی یک کپی می سازیم.

```
while len(Q)>0:  
    u = minimum_distance(distance, Q)  
    Q.remove(u)
```

حال به ازای هر گره (سوئیچ ها) گره ای که کمترین فاصله برای آن ثبت شده است را در متغیر `u` ذخیره می کنیم و این گره را از لیست گره ها حذف می کنیم. حال با

توجه به بدنه تابع `minimum_distance` که بالاتر آمده است، و از طرفی اینکه دیکشنری `distance` تماماً با مقادیر بی نهایت پر شده (بجز گره مبدأ که 0 بود)، در مقایسه‌ی موجود در بدنه آن تابع هیچگاه شرط مقایسه برقرار نمی‌شود. پس اولین گره ای که انتخاب می‌شود همان گره 0 است که در بدنه تابع مذکور در ابتدا مقداردهی شده است.



```
while len(Q)>0:  
    ...  
    for p in switches:  
        if adjacency[u][p]!=None:  
            w = 1  
            if distance[u] + w < distance[p]:  
                distance[p] = distance[u] + w  
                previous[p] = u
```

در ادامه، برای تمام سوئیچ‌های دیگر که همسایه سوئیچ `u` هستند (یعنی سوئیچ منتخب `u` به آنها لینک دارد) فاصله آنها را به روز رسانی می‌کنیم. به این صورت که مقایسه‌ی می‌کنیم اگر مجموع وزن یال بین `u` و گره همسایه با فاصله ذخیره شده برای گره `u` از فاصله ذخیره شده برای گره همسایه کمتر بود، فاصله گره همسایه برابر با مجموع وزن یال بین `u` و گره همسایه با فاصله ذخیره شده برای گره `u` قرار می‌گیرد. همچنین در

مسیر ذخیره شده به عنوان کوتاه ترین مسیر، پدر (گره پیشین در مسیر) گره همسایه را برابر با گره u قرار می دهیم؛ زیرا در واقع با به روز رسانی فاصله مذکور، مسیر رسیدن به گره همسایه را از طریق گره u ثبت کرده ایم.

وزن یال ها در اینجا 1 فرض شده است.

پس از اتمام حلقه for فاصله های تمام همسایه های u به روز رسانی می شوند. پس از اتمام حلقه while مقدار کوتاه ترین مسیر برای تمام سوئیچ ها بدست می آید.

```
r=[]
p=dst
r.append(p)
q=previous[p]
```

حال می خواهیم خود کوتاه ترین مسیر را داشته باشیم. می دانیم که در قسمت های قبل گره i پدر (گره پیشین در مسیر) هر کدام از گره ها را ذخیره کرده ایم. حال به این منظور از گره مقصد شروع می کنیم تا به گره مبدأ برسیم (زیرا برای هر گره پدرس را ذخیره کرده ایم، پس باید مسیر را بر عکس طی کنیم، یعنی از فرزند ها به پدرها). در ابتدا گره شروع برای پیمایش مسیر برابر با گره مقصد قرار داده می شود و آن را به کوتاه ترین مسیر اضافه می کنیم. سپس گره پدر را هم در متغیر دیگری ذخیره می کنیم که برای پیمایش مسیر استفاده می شود.

```
while q is not None:  
    if q == src:  
        r.append(q)  
        break  
  
    p=q  
    r.append(p)  
    q=previous[p]
```

در پیمایش مسیر، تا زمانی که به گره مبدأ نرسیده ایم، یکی یکی به عقب می رویم و گره های روی مسیر را به کوتاه ترین مسیر اضافه می کنیم. هنگامی که به گره مبدأ رسیدیم، آن را هم به مسیر اضافه می کنیم و از حلقه بیرون می آییم.

```
r.reverse()  
if src==dst:  
    path=[src]  
else:  
    path=r
```

حال با توجه به قسمت قبل، می دانیم که از گره های فرزند به سمت پدر رسیدیم (از مقصد به مبدأ). پس برای بدست آوردن مسیر از مبدأ به مقصد باید مسیر بدست آمده را برعکس کنیم که با reverse این کار انجام شده است. سپس بررسی می کنیم که اگر مبدأ و مقصد یکسان داده شده بودند، مسیر ما همان خود گره مبدأ می شود و در غیر این صورت همان مسیری که بدست آورده ایم.

```
# Now add the ports  
r = []  
in_port = first_port  
for s1,s2 in zip(path[:-1],path[1:]):
```

حال می خواهیم پورت های مربوط به مسیر پیدا شده را هم بدست آوریم. به این منظور ابتدا پورت ورودی را برابر با پورت ورودی تابع اصلی (تابع get_path) قرار می دهیم. این پورت در هر مرحله پیمایش مسیر به پورت ورودی از گره اولی به دومی تغییر می یابد. پورت خروجی هم به همین ترتیب است. برای پیمایش مسیر از تابع zip استفاده شده تا دو تابی هایی از گره های متوالی روی مسیر داشته باشیم.

```
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
```

حال همانطور که در بالا گفته شد، هر بار پورت خروجی را برابر با خانه s_1, s_2 دیکشنری متناظر با گراف توپولوژی قرار می دهیم. این پورت، پورت خروجی گره اولی است. پورت ورودی هم پورت ورودی گره دومی است. حال هر بار یک سه تابی از گره کنونی در پیمایش، پورت ورودی و خروجی متناظر را به خروجی کلی تابع اضافه می کنیم. نهایتاً گره مقصد و پورت های متناظر را هم به خروجی کلی اضافه می کنیم و خروجی بازگردانده می شود.

توضیح نحوه کار کد کنترلر RYU

```
class ProjectController(app_manager.RyuApp):  
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
```

در ابتدا برای اجرای برنامه مبتنی بر RYU، از کلاس OpenFlow ryu.base.app_manager.RyuApp به 1.3 تنظیم شده است.

```
def __init__(self, *args, **kwargs):  
    super(ProjectController, self).__init__(*args, **kwargs)  
    self.mac_to_port = {}  
    self.topology_api_app = self  
    self.datapath_list=[ ]
```

سپس مقدار دهی های اولیه صورت گرفته است. جدول MAC آدرس ها هم ساخته شده و تنظیم می کنیم که برای خود این کنترلر می خواهیم عملیات ها را انجام دهیم. نهایتاً لیست datapath ها هم ساخته می شود. Datapath ها در واقع عملیات های اصلی مانند ایجاد رخداد (event) های متناظر با پیام های دریافتی و ارتباط با سوئیچ OpenFlow را دارند.



```
# Handy function that lists all attributes in the given object
def ls(self,obj):
    print("\n".join([x for x in dir(obj) if x[0] != "_"]))
```

این تابع به این منظور نوشته شده تا تمام اعضای کلاس (خصیصه ها) مربوط به یک شیء را چاپ کند. علت اینکه موارد دارای "- " ابتدایی حذف شده اند این است که این موارد تابع هستند و ما فقط خصیصه ها را خواسته ایم.

```
def add_flow(self, datapath, in_port, dst, actions):  
    ofproto = datapath.ofproto  
    parser = datapath.ofproto_parser  
  
    match = datapath.ofproto_parser.OFPMatch(in_port=in_port,  
                                             eth_dst=dst)
```

در این تابع قصد داریم یک entry به جدول flow اضافه کنیم. در ابتدا از datapath خصیصه ofproto و ofproto_parser را می گیریم. این دو شیء نشان دهنده پروتکل OpenFlow استفاده شده میان کنترلر و سوئیچ هستند. سپس یک قاعده برای match بسته ها تعیین می کنیم و آن هم این است که پورت ورودی برابر با پورت ورودی این تابع باشد و مقصد هم برابر با مقصد ورودی تابع باشد.

```
inst = [parser.OFPIstructionActions(ofproto.OFPI_APPLY_ACTIONS,  
                                     actions)]
```

این قسمت از کد یک دستور برای ارسال پیغام flow_mod است. برای این استفاده شده action که مورد نظر فوراً استفاده و انجام شود. توجه داشته باشید که actions ورودی تابع است.



```
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)  
  
datapath.send_msg(mod)
```

نهایتاً در این تابع entry مورد نظر ساخته می شود و خصوصیات مورد نظر که ورودی تابع و... بودند نیز به این entry داده شده و به سوئیچ OpenFlow ارسال می گردد. علت استفاده از ofproto.OFPFC_ADD این است که می خواهیم یک entry جدید به جدول اضافه شود.

همچنین datapath تنظیم شده هم همان ای است که ورودی این تابع بود که آن هم از یک رخداد می آید. در ادامه این روند شرح داده می شود.

```
def install_path(self, p, ev, src_mac, dst_mac):  
    print "install_path is called"  
  
    #print "p=", p, " src_mac=", src_mac, " dst_mac=", dst_mac  
  
    msg = ev.msg  
  
    datapath = msg.datapath  
  
    ofproto = datapath.ofproto  
  
    parser = datapath.ofproto_parser
```

در این تابع مسیر بدست آمده را به جدول flow ها می خواهیم اضافه کنیم. ورودی های این تابع خود مسیر (p)، رخداد (ev)، آدرس MAC مبدأ (src_mac) و آدرس مقصد (dst_mac) هستند. در ادامه ابتدا پیغام متناظر با رخداد را در msg ذخیر می کنیم. سپس datapath متناظر را در parser ذخیره می کنیم. توضیحات مربوط به datapath بالاتر شرح داده شد. parser و ofproto هم در قسمت های قبل شرح داده شد.

```
def install_path(self, p, ev, src_mac, dst_mac):  
    ...  
    for sw, in_port, out_port in p:  
        #print src_mac,"->", dst_mac, "via ", sw, " in_port=",  
        in_port, " out_port=", out_port  
        match=parser.OFPMatch(in_port=in_port, eth_src=src_mac,  
        eth_dst=dst_mac)  
        actions=[parser.OFPActionOutput(out_port)]  
        datapath=self.datapath_list[int(sw)-1]
```

حال با توجه به اینکه می خواهیم کل مسیر را به جدول اضافه کنیم، پس باید روی مسیر پیمایش کنیم. این مسیر از الگوریتم کوتاه ترین مسیر بدست آمده و در بخش مربوط به این الگوریتم، شرح داده شد که این مسیر به صورت لیستی از سه تابی هایی از سوئیچ مقصد، پورت ورودی و پورت خروجی ذخیره می شود. پس برای پیمایش روی این مسیر باید در هر مرحله یک سه تابی را پیمایش کنیم. در بدنه حلقه for، یک match می سازیم که قاعده اش همان پورت ورودی، آدرس MAC ورودی و خروجی باشد. سپس یک شیء از کلاس OFPActionOutput ساخته می شود و مقصدش هم همان پورت خروجی تنظیم می شود. پس از آن datapath متناظر با سوئیچ مقصد نیز از لیست datapath ها گرفته می شود.

```
def install_path(self, p, ev, src_mac, dst_mac):  
    ...  
    for sw, in_port, out_port in p:  
        ...  
        inst =  
[parser.OFPInstructionActions(ofproto.OFPI_APPLY_ACTIONS , actions)]  
  
        mod = datapath.ofproto_parser.OFPPFlowMod(datapath=datapath,  
match=match, idle_timeout=0, hard_timeout=0, priority=1,  
instructions=inst)  
  
        datapath.send_msg(mod)
```

در نهایت در انتهای بدن حلقه، یک instruction ساخته می شود که آن هم در قسمت های قبل شرح داده شد. سپس یک پیغام Flow Mod ساخته شده و خصیصه های آن هم به تناسب تنظیم می شوند و این پیغام نهایتاً ارسال می شود. آن به 1 تنظیم شده و علت آنست که می خواهیم این عملیات بر حالتی که Table-miss داریم اولویت داشته باشد. با پایان حلقه for، کل مسیر به جدول اضافه می شود.

```
● ● ●  
@set_ev_cls(ofp_event.EventOFPSwitchFeatures , CONFIG_DISPATCHER)  
  
def switch_features_handler(self , ev):  
    ...
```

این تابع برای مدیریت دریافت یک پیغام از نوع switch features از طرف یک datapath توسط کنترلر است. خط اول این تابع را به مدیریت رخداد EventOFPSwitchFeatures منصوب می کند. ورودی این تابع هم یک رخداد از همین نوع است. چون این رخداد مربوط به config هاست و برای رد و بدل کردن بسته های عادی نیست، نوع dispatcher آن را CONFIG_DISPATCHER تعیین می کنیم.

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

def switch_features_handler(self , ev):
    ...
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    match = parser.OFPMatch()
```

حال datapath را از پیغام متناظر با رخداد دریافت می کنیم و پس از آن اطلاعات مربوط به پروتکل OpenFlow میان کنترلر و سوئیچ را دریافت می کنیم و یک match می سازیم که هیچ قاعده ای ندارد و هر بسته ای را match می کند.

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

def switch_features_handler(self , ev):
    ...

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

    inst = [parser.OFPIstructionActions(ofproto.OFPI_APPLY_ACTIONS
, actions)]
```

سپس یک شیء از کلاس OFPActionOutput ساخته می شود و مقصدش هم کنترلر تنظیم می شود و OFPCML_NO_BUFFER به این منظور استفاده شده که تمام پیغام ها یک جا به کنترلر فرستاده شوند. پس از آن instruction ساخته شده که توضیحات آن قبل اشاره شده است.

```
● ● ● @set_ev_cls(ofp_event.EventOFPSwitchFeatures , CONFIG_DISPATCHER) def switch_features_handler(self , ev): ... 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)
```

در نهایت یک پیغام Flow Mod ساخته شده و خصیصه های آن به تناسب تنظیم می شوند و این پیغام فرستاده می شود.

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

این تابع برای پذیرفتن بسته های PacketIn است. در خط اول نوع رخداد آن را تنظیم می کنیم و ورودی این تابع هم یک رخداد از همین نوع EventOFPPacketIn

است. چون این رخداد مربوط به بسته های داده است، نوع dispatcher آن را `MAIN_DISPATCHER` تعریف می کنیم.

```
● ● ●

@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']
```

در ابتدای بدنه این تابع، همان روند که در قسمت های قبل شرح داده شد تکرار می شود. پس از آن پورت ورودی را از بدنه ای پیغام می گیریم.

```

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    ...
    pkt = packet.Packet(msg.data)
    eth = pkt.get_protocol(ethernet.ethernet)

```

حال یک بسته با استفاده از کتابخانه ryu.lib.packet می سازیم و داده‌ی آن را برابر با پیغام دریافتی قرار می‌دهیم. سپس فریم ethernet را از این بسته دریافت می‌کنیم.

این فریم به صورت زیر می‌تواند باشد:

Preamble	Destination MAC	Source MAC	EtherType/ Size	PayLoad	CRC
1 2 3 4 5 6 7 8	1 2 3 4 5 6	1 2 3 4 5 6	1 2		1 2 3 4

```
● ● ●  
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)  
  
def _packet_in_handler(self, ev):  
    ...  
  
    #avoid broadcast from LLDP  
  
    if eth.ethertype==35020:  
  
        return
```

حال اگر مشخصه etherType آن از نوع Link Layer Discovery Protocol بود، آن را مدیریت نمی کنیم و از تابع خارج می شویم (پیغام های از این نوع مرتبأ بدون داشتن مقصد به صورت broadcast برای بررسی صحت کار کرد شبکه فرستاده می شوند).

```
● ● ● @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER) def _packet_in_handler(self, ev): ... dst = eth.dst src = eth.src dpid = datapath.id self.mac_to_port.setdefault(dpid, {})
```

مبدأ و مقصد و شناسه datapath را از آن فریم می گیریم. سپس شناسه datapath را به جدول MAC آدرس ها اضافه می کنیم.

```
● ● ● @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER) def _packet_in_handler(self, ev): ... if src not in mymac.keys(): mymac[src]=( dpid, in_port)
```

در ابتدای کد یک دیکشنری داشتیم که هر سوئیچ را به شناسه datapath و پورت ورودی متناظر map می کرد. حال اگر سوئیچ مبدأ در این دیکشنری نبود، آن را به دیکشنری اضافه می کنیم.

```
● ● ●  
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)  
  
def _packet_in_handler(self, ev):  
  
    ...  
  
    if dst in mymac.keys():  
  
        p = get_path(mymac[src][0], mymac[dst][0], mymac[src][1],  
mymac[dst][1])
```

حال می خواهیم بینیم که آیا مقصد در آن دیکشنری بوده یا نه؛ در صورتی که باشد، کوتاه ترین مسیر از مبدأ تا مقصد را با استفاده از تابعی که الگوریتم کوتاه ترین مسیر را اجرا می کرد، بدست می آوریم. همان طور که در توضیحات این تابع گفته شد، ورودی های آن شناسه مبدأ، مقصد، و پورت های آنها هستند.

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    ...
    if dst in mymac.keys():
        ...
        print p
        self.install_path(p, ev, src, dst)
        out_port = p[0][2]
```

حال پس از بدست آمدن کوتاه ترین مسیر مبدأ تا مقصد، این مسیر را با استفاده از تابع `install_path` که بالاتر مفصلاً شرح داده شد، به جدول flow اضافه می کنیم. همانطور که قبلاً گفته شد، مسیر متشكل از سه تابی های مقصد، پورت ورودی و خروجی است. پس برای بدست آوردن پورت خروجی مبدأ، کافیست سومین عنصر در سه تابی اول در این مسیر را بدست آوریم (چون اندیس ها از 0 شروع می شوند، اندیس 2 متناظر با سومین عنصر است).

```
██████████  
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)  
  
def _packet_in_handler(self, ev):  
    ...  
    if dst in mymac.keys():  
        ...  
    else:  
        out_port = ofproto.OFPP_FLOOD
```

حال در صورتی که مقصد از قبل در دیکشنری نبود، باید flood رخ بدهد و بسته به همه ارسال شود.

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    ...
    data=None
    if msg.buffer_id==ofproto.OFP_NO_BUFFER:
        data=msg.data
```

حال در صورتی که پیغام بافر نداشت، بدنه داده های آن را گرفته و به عنوان داده در خروجی قرار می دهیم. در غیر این صورت تنها شناسه بافر در خروجی قرار گرفته و داده های آن خالی می ماند.



```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    ...
    out = parser.OFPPacketOut(
        datapath=datapath, buffer_id=msg.buffer_id,
        in_port=in_port,
        actions=actions, data=data)
    datapath.send_msg(out)
```

در انتهای بدنه این تابع، خروجی که از نوع PacketOut است را می سازیم و خصیصه های آن را هم که هر کدام در بالا توضیح داده شد را تنظیم می کنیم و خروجی ارسال می شود.



```
@set_ev_cls(event.EventSwitchEnter)
def get_topology_data(self, ev):
    ...
```

این تابع وظیفه آن را دارد که توپولوژی را گرفته و از روی آن گراف همسایگی متناظر را بسازد. از این گراف برای پیدا کردن کوتاه ترین مسیرها استفاده می شود. این تابع در ابتدای کار صدا زده می شود.

```
@set_ev_cls(event.EventSwitchEnter)
def get_topology_data(self, ev):
    global switches
    switch_list = get_switch(self.topology_api_app, None)
    switches=[switch.dp.id for switch in switch_list]
    self.datapath_list=[switch.dp for switch in switch_list]
    #print "self.datapath_list=", self.datapath_list
    print "switches=", switches
```

در ابتدای بدنه تابع، لیست سوئیچ ها را از توپولوژی می گیریم و از آن لیست شناسه datapath ها و لیست خود متناظر را می سازیم.

```
@set_ev_cls(event.EventSwitchEnter)

def get_topology_data(self, ev):
    ...
    links_list = get_link(self.topology_api_app, None)

    mylinks=
[(link.src.dpid,link.dst.dpid,link.src.port_no,link.dst.port_no) for
 link in links_list]
```

حال لیست لینک ها را از توپولوژی می گیریم. سپس از این لیست، لیستی متشکل از 4 تابی هایی را می سازیم که شامل شناسه datapath مبدأ، شناسه datapath مقصد، پورت مبدأ و پورت مقصد است.

```
@set_ev_cls(event.EventSwitchEnter)

def get_topology_data(self, ev):
    ...
    for s1,s2,port1,port2 in mylinks:
        adjacency[s1][s2]=port1
        adjacency[s2][s1]=port2
```

در نهایت به ازای هر عنصر لیستی که در قسمت قبل گفته شد، خانه‌ی متناظر با آن را در گراف متناظر با توپولوژی شبکه تکمیل می‌کنیم.

اصلاحات کد

ایراداتی در کد ارائه شده وجود داشت که یک به یک به شرح زیر بر طرف گشته اند:

1. در indentation کد ارائه شده ایرادات بسیاری وجود داشت که همه آنها اصلاح شد.

2. در تابع install_path از تابع get_switch استفاده شده است و فرض شده است که این تابع لیست datapath را مرتب شده بر اساس شناسه شان بر می گرداند اما این فرض اشتباه است و این تابع لزوماً لیست مرتب شده را بر نمی گرداند. پس آن را به گونه ای اصلاح می کنیم که عنصری از این لیست را باز گرداند که شناسه آن برابر با sw باشد.

حالت اشتباه:

```
def install_path(self, p, ev, src_mac, dst_mac):
    ...
    for sw, in_port, out_port in p:
        ...
        datapath=self.datapath_list[int(sw)-1]
```

حالت اصلاح شده:

```
def install_path(self, p, ev, src_mac, dst_mac):  
    ...  
    for sw, in_port, out_port in p:  
        ...  
        datapath = [dp for dp in self.datapath_list if dp.id  
                   == sw][0]
```

3. در تابع `minimum_distance` در ابتدا مقدار اولیه گره منتخب برابر با 0 قرار داده شده اما در لیست سوئیچ ها 0 وجود ندارد. پس آن را طوری تغییر می دهیم که مقدار اولیه اش اولین عنصر لیست باشد.

حالت اشتباه:

```
def minimum_distance(distance, Q):  
    ...  
    node = 0
```

حالت اصلاح شده:

```
def minimum_distance(distance, Q):
    ...
    node = next(iterator(Q))
```

4. در تابع packet_in_handler در حالتی که مقصد از قبل شناخته شده باشد، پورت خروجی از اولین عنصر در کوتاه ترین مسیر پیدا شده بدست می‌آید، اما این پورت در موقعي که از قبل FLOOD رخ داده لزوماً پورت درست را نمی‌دهد و سبب می‌شود که تعدادی از بسته‌ها به مقصد درست نرسند. برای حل این مشکل بجای اینکه پورت مقصد را از اولین عنصر مسیر بدست آوریم، عنصری از مسیر را پیدا می‌کنیم که شناسه datapath آن با ای که از رخداد کنونی (PacketIn) بدست آمده یکسان باشد و پورت خروجی آن را به عنوان پورت خروجی قرار می‌دهیم.

حالت اشتباه:

```
● ● ●  
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)  
def _packet_in_handler(self, ev):  
    ...  
    if dst in mymac.keys():  
        ...  
        out_port = p[0][2]
```

حالت اصلاح شده:

```
● ● ●  
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)  
def _packet_in_handler(self, ev):  
    ...  
    if dst in mymac.keys():  
        ...  
        out_port = p[0][2]  
        foundDp = [x for x in p if x[0] == dpid]  
        if (len(foundDp) > 0):  
            out_port = foundDp[0][2]
```

علت استفاده از شرط if این است که در صورتی که توپولوژی اشتباه باشد، exception رخ ندهد.

5. تابع add_flow هیچ جا استفاده نشده و اضافی است و می‌تواند حذف شود؛ زیرا در تابع install_path عمل اضافه کردن entry به Flow Table را انجام داده ایم.

6. تابعی با نام ls نوشته شده که هیچ جا استفاده نشده و قابل حذف است؛ این تابع ممکن است در هنگام debug کد به کار آید، زیرا توسط آن می‌توان خصیصه‌های اشیاء را بدست آورد.

7. در تابع packet_in_handler یک شرط بررسی شده که در بدنه آن عملاً هیچ کاری صورت نمی‌گیرد و اضافی و قابل حذف است:

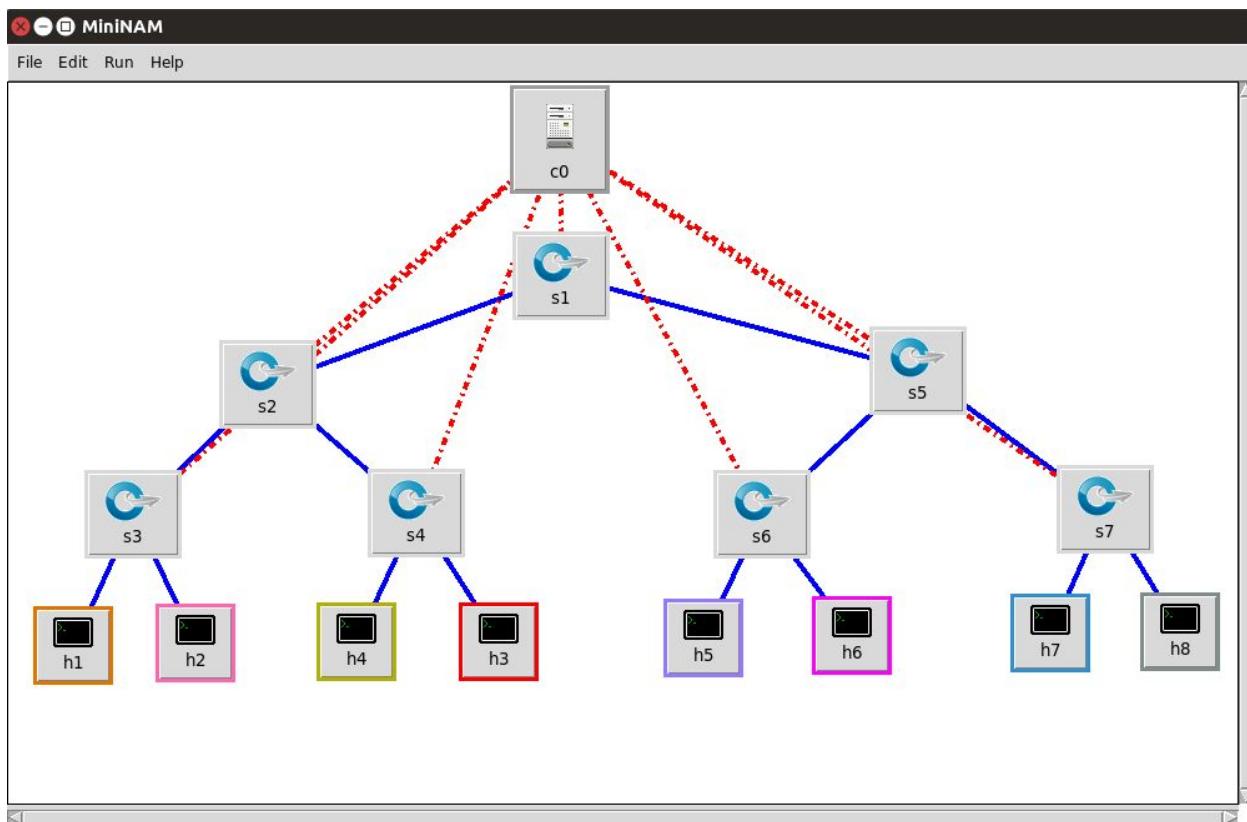
```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    ...
    if out_port != ofproto.OFPP_FLOOD:
        match = parser.OFPMatch(in_port=in_port, eth_src=src,
                               eth_dst=dst)
```

نتایج

در تمامی قسمت های بعد از کد اصلاح شده استفاده شده است.

توپولوژی درختی با عمق 3

شکل توپولوژی:



این توپولوژی همان توپولوژی tree,3 است.

پنجره کنترلر در ابتدای کار:

```
mobina@mobina:~/Desktop$ ryu-manager ryu.py --observe-links
loading app ryu.py
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app ryu.py of ProjectController
instantiating app ryu.topology.switches of Switches
instantiating app ryu.controller.ofp_handler of OFPHandler
switch_features_handler is called
switches= [1, 3, 6]
s3:3 connected to s2:1
s6:3 connected to s5:1
s1:1 connected to s2:3
s7:3 connected to s5:2
s1:2 connected to s5:3
switches= [1, 2, 3, 4, 5, 6, 7]
s2:3 connected to s1:1
s5:1 connected to s6:3
s4:3 connected to s2:2
```

همانطور که مشاهده می شود، در ابتدای کار تابع switch_features_handler صدای زده می شود و سوئیچ ها توسط کنترلر شناسایی می شوند.

سپس entry های جدول MAC آدرس یک به یک به آن اضافه می شوند:

```
mobina@mobina: ~/Desktop
s6:3 connected to s5:1
s5:3 connected to s1:2
s2:2 connected to s4:3
adding new entry to mac table: dpid = 5 in_port = 3
adding new entry to mac table: dpid = 2 in_port = 3
adding new entry to mac table: dpid = 6 in_port = 3
adding new entry to mac table: dpid = 3 in_port = 3
adding new entry to mac table: dpid = 7 in_port = 2
adding new entry to mac table: dpid = 5 in_port = 1
adding new entry to mac table: dpid = 1 in_port = 1
adding new entry to mac table: dpid = 3 in_port = 1
adding new entry to mac table: dpid = 7 in_port = 1
adding new entry to mac table: dpid = 7 in_port = 3
adding new entry to mac table: dpid = 5 in_port = 2
adding new entry to mac table: dpid = 4 in_port = 1
adding new entry to mac table: dpid = 2 in_port = 1
adding new entry to mac table: dpid = 2 in_port = 2
adding new entry to mac table: dpid = 4 in_port = 3
adding new entry to mac table: dpid = 4 in_port = 2
adding new entry to mac table: dpid = 6 in_port = 1
adding new entry to mac table: dpid = 3 in_port = 2
adding new entry to mac table: dpid = 6 in_port = 2
adding new entry to mac table: dpid = 1 in_port = 2
```

همچنین جهت بررسی صحت دریافت توپولوژی توسط کنترلر اتصال سوییچ ها و پورت هایشان در خروجی چاپ و بررسی شد:

```
switches= [1, 2, 3, 4, 5, 6, 7]
s2:3 connected to s1:1
s5:1 connected to s6:3
s4:3 connected to s2:2
s2:1 connected to s3:3
s1:1 connected to s2:3
s3:3 connected to s2:1
s5:2 connected to s7:3
s1:2 connected to s5:3
s5:3 connected to s1:2
s2:2 connected to s4:3
```

نتیجه اجرای دستور net در مینی نت:

```

xmobina@mobina: ~/Desktop
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> 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
mininet> 

```

نتیجه اجرای دستور pingall در مینی نت:

پنجره مینی نت:

```

xmobina@mobina: ~/Desktop
*** 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 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> 

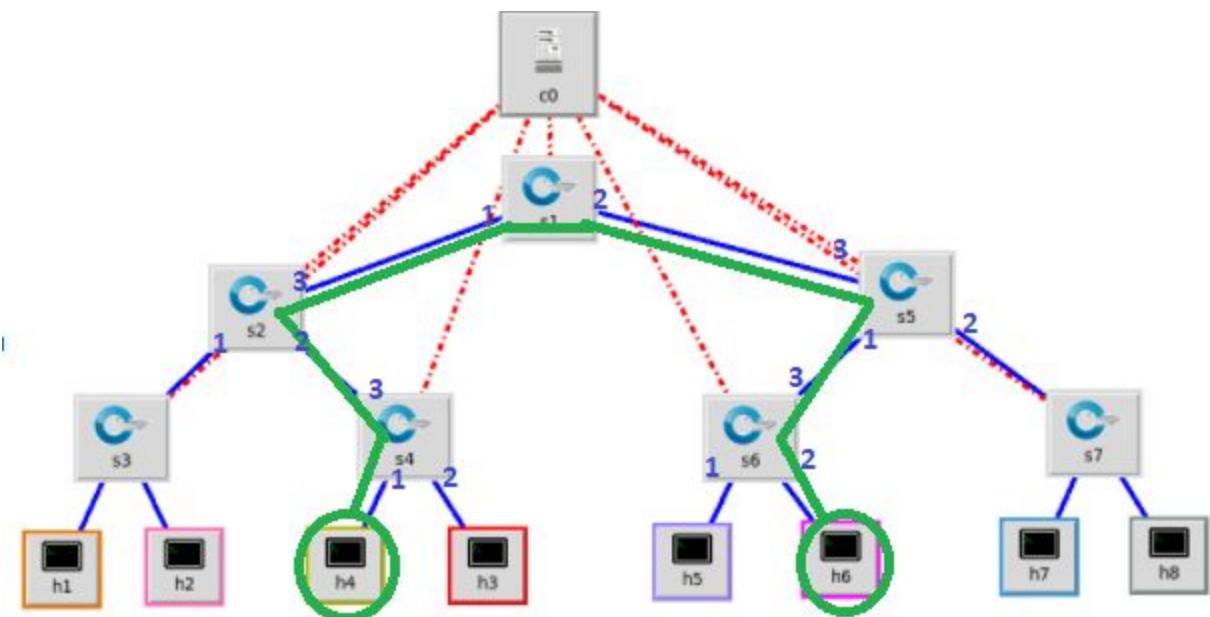
```

همانطور که مشاهده می شود تمام بسته ها رسیده اند و هیچ بسته ای drop نشده است.

پنجه کنترلر:

```
> From l->s4 to s6->2
get_path src(switch: 4, port: 1) dst(switch: 6, port: 2)
install_path: 1->s4->3 2->s2->3 1->s1->2 3->s5->1 3->s6->2
> From l->s7 to s4->1
get_path src(switch: 7, port: 1) dst(switch: 4, port: 1)
install_path: 1->s7->3 2->s5->3 2->s1->1 3->s2->2 3->s4->1
> From l->s7 to s4->1
get_path src(switch: 7, port: 1) dst(switch: 4, port: 1)
install_path: 1->s7->3 2->s5->3 2->s1->1 3->s2->2 3->s4->1
> From l->s4 to s7->1
get_path src(switch: 4, port: 1) dst(switch: 7, port: 1)
install_path: 1->s4->3 2->s2->3 1->s1->2 3->s5->2 3->s7->1
> From l->s4 to s7->1
get_path src(switch: 4, port: 1) dst(switch: 7, port: 1)
install_path: 1->s4->3 2->s2->3 1->s1->2 3->s5->2 3->s7->1
```

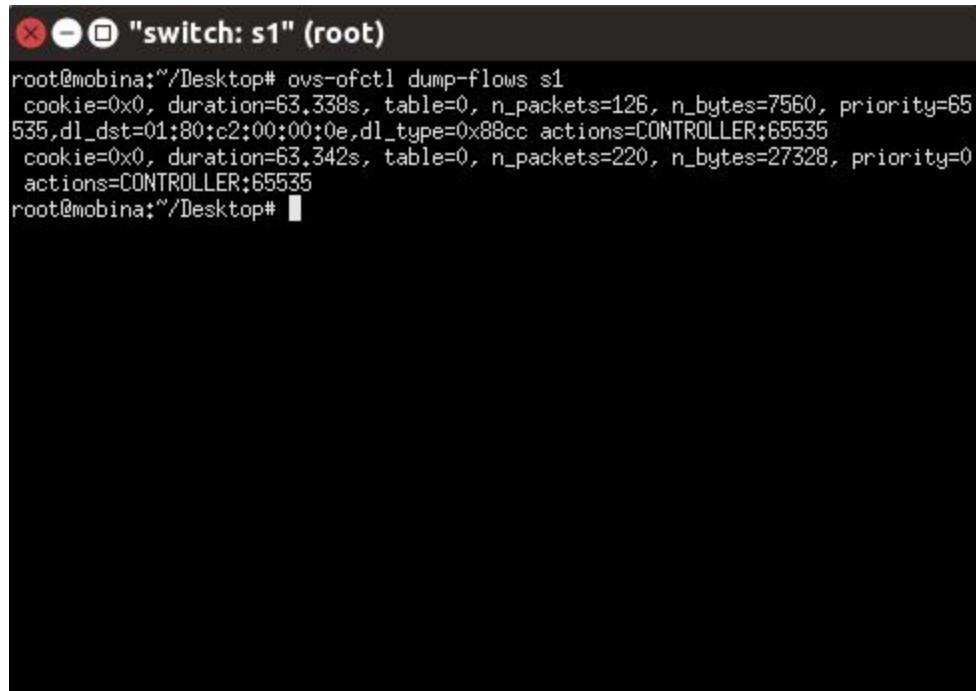
در خط اول هر بخش از این خروجی که با ">" شروع میشود، سوییچ مبدا و پورت ورودی آن و سوییچ مقصد و پورت خروجی آن نشان داده میشود. خط دوم نشان دهنده اجرای تابع get_path و ورودی های آن است. خط سوم نیز نشان دهنده اجرا شدن تابع install_path و مسیری که در جدول روتر ها ثبت می شود است. به عنوان مثال برای رفتن از هاست 3 (پورت 1 سوییچ 4) به هاست 6 (پورت 2 سوییچ 6) باید مسیر زیر را طی کرد:



که همان مسیری است که در خروجی کنترلر مشاهده کردیم.

پنجره سویچ ها:

به عنوان مثال، نتیجه اجرای دستور `ovs-ofctl dump-flows` در پنجره سویچ ۱:



```
root@mobina:~/Desktop# ovs-ofctl dump-flows s1
cookie=0x0, duration=63.338s, table=0, n_packets=126, n_bytes=7560, priority=65
535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x0, duration=63.342s, table=0, n_packets=220, n_bytes=27328, priority=0
actions=CONTROLLER:65535
root@mobina:~/Desktop#
```

که نشان می دهد سویچ درست عمل می کند.

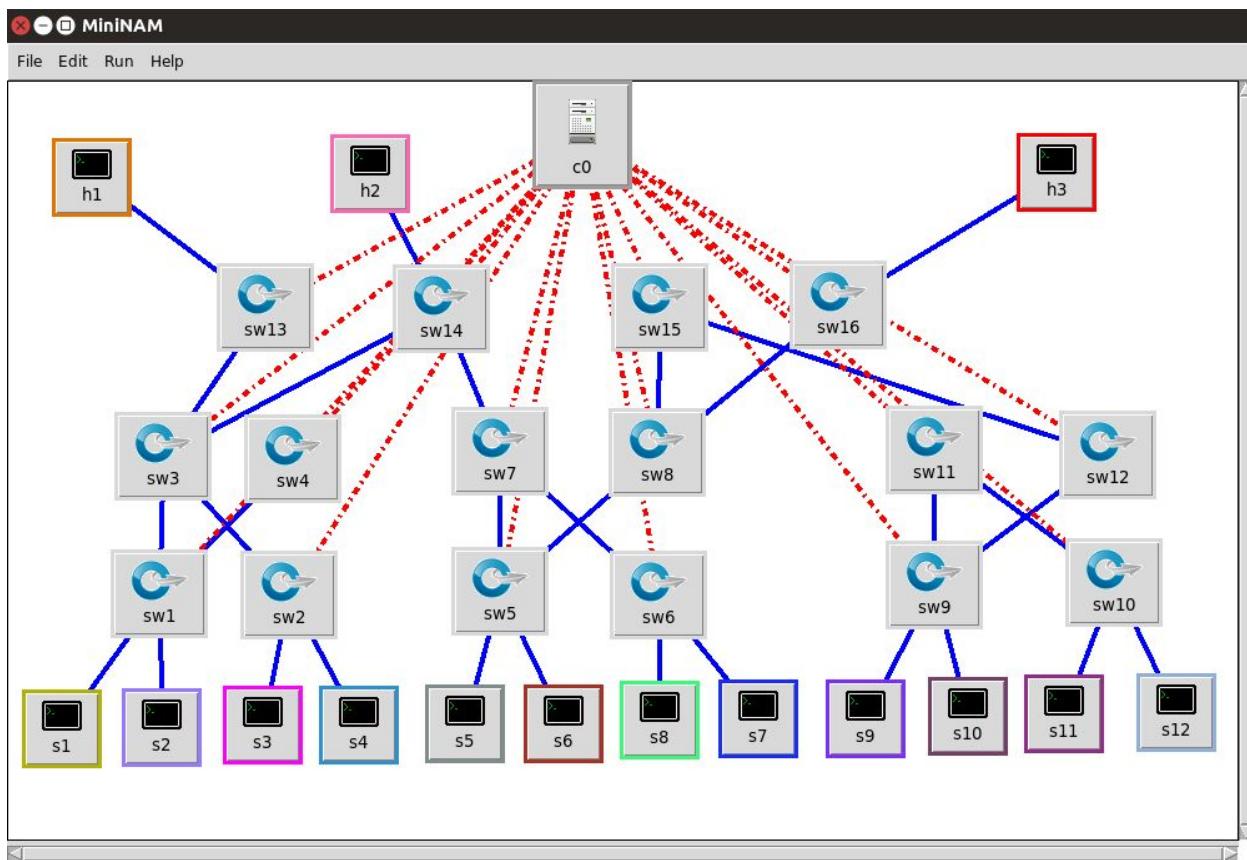
پنجره هاست ها:

```
root@mobina:~/Desktop# tcpdump -en -i h6-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h6-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
21:14:16.461316 d6:5f:d8:b8:27:b8 > 01:80:c2:00:00:0e, ethertype LLDP (0x88cc),
length 60: LLDP, length 46
21:14:17.093910 1e:31:e3:36:49:27 > 33:33:00:00:00:02, ethertype IPv6 (0x86dd),
length 70: fe80::1c31:e3ff:fe36:4927 > ff02::2: ICMP6, router solicitation, leng
th 16
21:14:17.099313 fa:2c:15:e8:99:a6 > 33:33:00:00:00:02, ethertype IPv6 (0x86dd),
length 70: fe80::f82c:15ff:fee8:99a6 > ff02::2: ICMP6, router solicitation, leng
th 16
21:14:17.468994 d6:5f:d8:b8:27:b8 > 01:80:c2:00:00:0e, ethertype LLDP (0x88cc),
length 60: LLDP, length 46
21:14:18.477320 d6:5f:d8:b8:27:b8 > 01:80:c2:00:00:0e, ethertype LLDP (0x88cc),
length 60: LLDP, length 46
21:14:19.485590 d6:5f:d8:b8:27:b8 > 01:80:c2:00:00:0e, ethertype LLDP (0x88cc),
length 60: LLDP, length 46
^C
6 packets captured
6 packets received by filter
0 packets dropped by kernel
root@mobina:~/Desktop#
```

که نشان می دهد هاست درست عمل می کند.

توپولوژی صورت پروژه فعلی با حذف دور

شکل توپولوژی:



توجه داشته باشید که مشخصات لینک ها در این شکل نیامده است. از توپولوژی اولیه که دارای حلقه بود، کمترین تعداد یال حذف شده تا حلقه ها به طور کامل حذف شوند. این توپولوژی در فایل topo2.py قرار دارد.

پنجره کنترلر در ابتدای کار:

```
mobina@mobina: ~/Desktop
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switches= [8, 2, 15, 14, 7]
switches= [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
switches= [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
switches= [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
switches= [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
adding new entry to mac table: dpid = 1 in_port = 3
adding new entry to mac table: dpid = 3 in_port = 2
switches= [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
switches= [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
switches= [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
switches= [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
switches= [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
switches= [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
switches= [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
switches= [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
switches= [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
switches= [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
adding new entry to mac table: dpid = 7 in_port = 2
adding new entry to mac table: dpid = 8 in_port = 1
```

لاگ کنترلر جهت بررسی صحت اتصال سویچ ها به یکدیگر:

```
s8:3 connected to s16:2
s8:2 connected to s15:1
s12:2 connected to s15:2
s12:1 connected to s9:4
s14:3 connected to s7:3
s14:2 connected to s3:4
s3:3 connected to s13:2
s16:2 connected to s8:3
s8:1 connected to s5:4
s5:4 connected to s8:1
s1:3 connected to s3:1
s7:2 connected to s6:3
s4:1 connected to s1:4
s10:3 connected to s11:2
s3:1 connected to s1:3
s3:2 connected to s2:3
s9:4 connected to s12:1
s7:1 connected to s5:3
s3:4 connected to s14:2
s15:1 connected to s8:2
s11:2 connected to s10:3
s1:4 connected to s4:1
s7:3 connected to s14:3
s5:3 connected to s7:1
s13:2 connected to s3:3
s6:3 connected to s7:2
s15:2 connected to s12:2
s11:1 connected to s9:3
s2:3 connected to s3:2
s9:3 connected to s11:1
```

پنجره مینی نت در ابتدای کار که مشخصات لینک ها نیز در آن نشان داده می شود:

```
mobina@mobina: ~/Desktop
*** Add switches
*** Add hosts
*** Add links
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12
*** Adding switches:
sw1 sw2 sw3 sw4 sw5 sw6 sw7 sw8 sw9 sw10 sw11 sw12 sw13 sw14 sw15 sw16
*** Adding links:
(h1, sw13) (h2, sw14) (h3, sw16) (s1, sw1) (s2, sw2) (s3, sw2) (s4, sw2) (s5, sw5) (s6, sw5) (s7, sw6) (s8, sw6) (s9, sw9) (s10, sw9) (s11, sw10) (s12, sw10) (50.00Mbit) (50.00Mbit) (sw1, sw3) (100.00Mbit) (100.00Mbit) (sw1, sw4) (100.00Mbit) (100.00Mbit) (sw2, sw3) (15.00Mbit) (15.00Mbit) (sw3, sw13) (10.00Mbit) (10.00Mbit) (sw3, sw14) (50.00Mbit) (sw5, sw7) (100.00Mbit) (100.00Mbit) (sw5, sw8) (100.00Mbit) (100.00Mbit) (sw6, sw7) (20.00Mbit) (20.00Mbit) (sw7, sw14) (10.00Mbit) (sw8, sw15) (15.00Mbit) (15.00Mbit) (sw8, sw16) (50.00Mbit) (50.00Mbit) (sw9, sw11) (100.00Mbit) (100.00Mbit) (sw9, sw12) (100.00Mbit) (100.00Mbit) (sw10, sw11) (15.00Mbit) (15.00Mbit) (sw12, sw15)
*** Configuring hosts
h1 h2 h3 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12
*** Starting controller
c0
*** Starting 16 switches
sw1 sw2 sw3 sw4 sw5 sw6 sw7 sw8 sw9 sw10 sw11 sw12 sw13 sw14 sw15 sw16 ... (50.00Mbit) (100.00Mbit) (100.00Mbit) (50.00Mbit) (100.00Mbit) (15.00Mbit) (10.00Mbit) (100.00Mbit) (50.00Mbit) (100.00Mbit) (100.00Mbit) (50.00Mbit) (100.00Mbit) (20.00Mbit) (100.00Mbit) (10.00Mbit) (15.00Mbit) (50.00Mbit) (100.00Mbit) (100.00Mbit) (15.00Mbit) (15.00Mbit) (10.00Mbit) (20.00Mbit) (10.00Mbit) (15.00Mbit) (15.00Mbit)
*** Starting CLI:
mininet> 
```

نتیجه اجرای دستور net در مینی نت:

```
mobina@mobina: ~/Desktop
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 sw1-eth4:sw4-eth1
sw2 lo: sw2-eth1:s3-eth0 sw2-eth2:s4-eth0 sw2-eth3:sw3-eth2
sw3 lo: sw3-eth1:sw1-eth3 sw3-eth2:sw2-eth3 sw3-eth3:sw13-eth2 sw3-eth4:sw14-eth2
sw4 lo: sw4-eth1:sw1-eth4
sw5 lo: sw5-eth1:s5-eth0 sw5-eth2:s6-eth0 sw5-eth3:sw7-eth1 sw5-eth4:sw8-eth1
sw6 lo: sw6-eth1:s7-eth0 sw6-eth2:s8-eth0 sw6-eth3:sw7-eth2
sw7 lo: sw7-eth1:sw5-eth3 sw7-eth2:sw6-eth3 sw7-eth3:sw14-eth3
sw8 lo: sw8-eth1:sw5-eth4 sw8-eth2:sw15-eth1 sw8-eth3:sw16-eth2
sw9 lo: sw9-eth1:s9-eth0 sw9-eth2:s10-eth0 sw9-eth3:sw11-eth1 sw9-eth4:sw12-eth1
sw10 lo: sw10-eth1:s11-eth0 sw10-eth2:s12-eth0 sw10-eth3:sw11-eth2
sw11 lo: sw11-eth1:sw9-eth3 sw11-eth2:sw10-eth3
sw12 lo: sw12-eth1:sw9-eth4 sw12-eth2:sw15-eth2
sw13 lo: sw13-eth1:h1-eth0 sw13-eth2:sw3-eth3
sw14 lo: sw14-eth1:h2-eth0 sw14-eth2:sw3-eth4 sw14-eth3:sw7-eth3
sw15 lo: sw15-eth1:sw8-eth2 sw15-eth2:sw12-eth2
sw16 lo: sw16-eth1:h3-eth0 sw16-eth2:sw8-eth3
c0
mininet> 
```

نتیجه اجرای دستور pingall در مینی نت:

پنجره مینی نت:

```

xmobina@mobina: ~/Desktop
c0
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.57 Mbits/sec', '12.5 Mbits/sec']
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>

```

همانطور که مشاهده می شود تمام بسته ها رسیده اند و هیچ بسته ای drop نشده است.

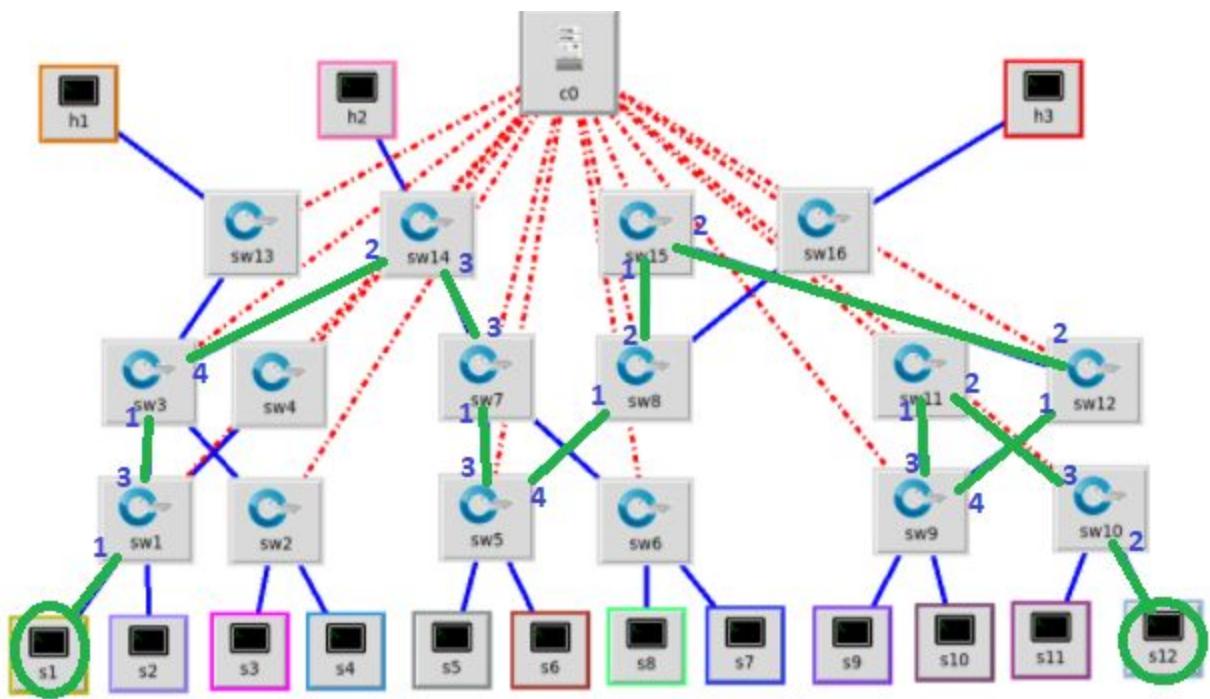
پنجره کنترلر:

```

> From l->s1 to s10->2
get_path src(switch: 1, port: 1) dst(switch: 10, port: 2)
install_path: 1->s1->3 1->s3->4 2->s14->3 3->s7->1 3->s5->4 1->s8->2 1->s15->2 2->s12->1 4->s9->3 1->s11->2 3->s10->2
> From l->s2 to s1->2
get_path src(switch: 2, port: 1) dst(switch: 1, port: 2)
install_path: 1->s2->3 2->s3->1 3->s1->2
> From l->s2 to s1->2
get_path src(switch: 2, port: 1) dst(switch: 1, port: 2)
install_path: 1->s2->3 2->s3->1 3->s1->2
> From 2->s1 to s2->1
get_path src(switch: 1, port: 2) dst(switch: 2, port: 1)
install_path: 2->s1->3 1->s3->2 3->s2->1

```

تمامی فرآیند ها مانند توپولوژی درخت با عمق 3 رخ می دهد که شرح آن در همان بخش موجود است. به عنوان مثال برای رفتن از سرور 1 (پورت 1 سویچ 1) به سرور 12 (پورت 2 سویچ 10) که طولانی ترین اتصال است باید مسیر زیر را طی کرد:



که همان مسیری است که در خط اول خروجی پنجره کنترلر پیشتر نشان داده شده بود:

```
> From l->s1 to s10->2
get_path src(switch: 1, port: 1) dst(switch: 10, port: 2)
install_path: 1->s1->3 1->s3->4 2->s14->3 3->s7->1 3->s5->4 1->s8->2 1->
s15->2 2->s12->1 4->s9->3 1->s11->2 3->s10->2
```

پنجره سوییچ ها:



```
x -o "switch: sw8" (root)
root@mobina:~/Desktop# ovs-ofctl dump-flows sw8
cookie=0x0, duration=26.572s, table=0, n_packets=38, n_bytes=2280, priority=655
35,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x0, duration=26.593s, table=0, n_packets=409, n_bytes=54639, priority=0
actions=CONTROLLER:65535
root@mobina:~/Desktop#
```

که نشان می دهد سویچ به درستی عمل می کند.

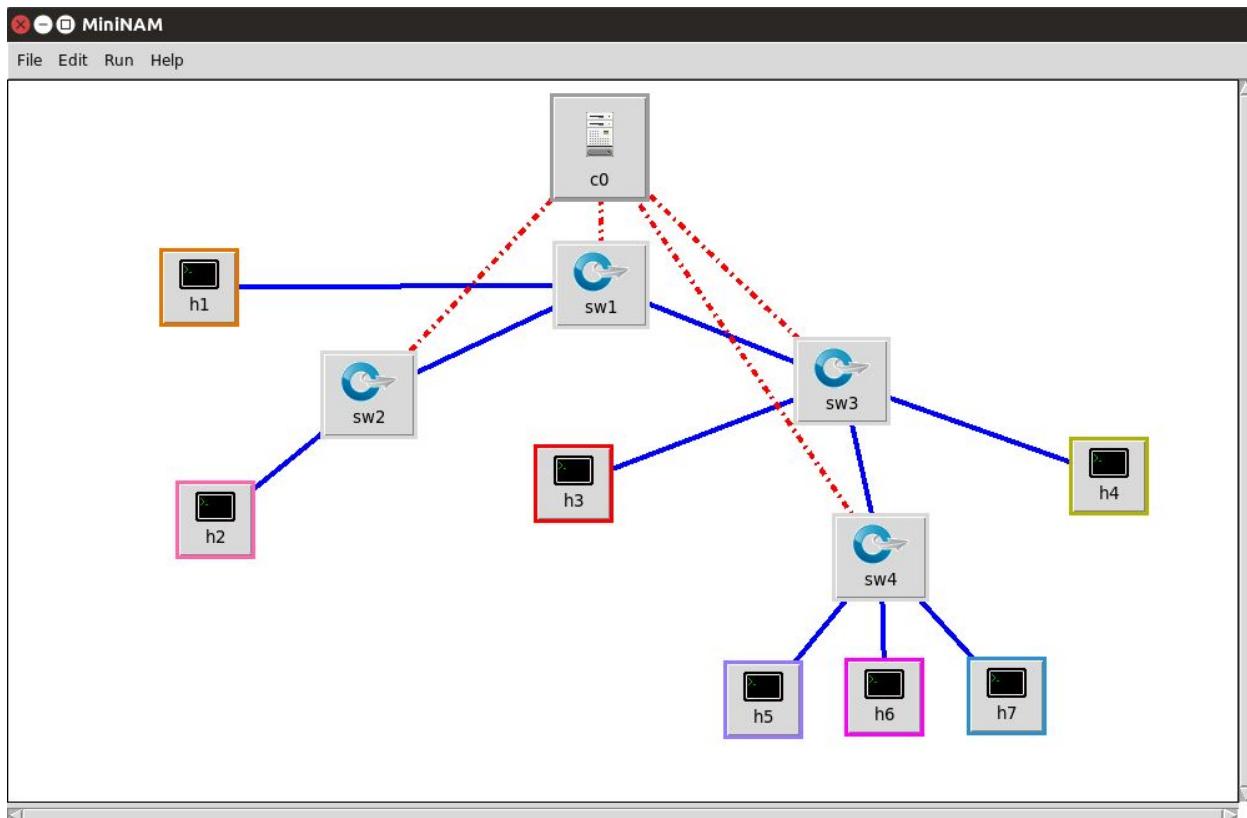
پنجره هاست ها:

```
✖ - □ "host: s4"
22:12:21,181331 00:00:00:00:00:32 > 33:33:00:00:00:02, ethertype IPv6 (0x86dd),
length 70: fe80::200:ff:fe00:32 > ff02::2: ICMP6, router solicitation, length 16
22:12:22,299780 06:9b:14:9a:84:1c > 01:80:c2:00:00:0e, ethertype LLDP (0x88cc),
length 60: LLDP, length 46
22:12:23,197142 be:5f:68:13:49:d0 > 33:33:00:00:00:02, ethertype IPv6 (0x86dd),
length 70: fe80::bc5f:68ff:fe13:49d0 > ff02::2: ICMP6, router solicitation, leng
th 16
22:12:23,216440 00:00:00:00:00:24 > 33:33:00:00:00:02, ethertype IPv6 (0x86dd),
length 70: fe80::200:ff:fe00:24 > ff02::2: ICMP6, router solicitation, length 16
22:12:23,216462 46:26:91:84:19:0d > 33:33:00:00:00:02, ethertype IPv6 (0x86dd),
length 70: fe80::4426:91ff:fe84:190d > ff02::2: ICMP6, router solicitation, leng
th 16
22:12:23,216481 00:00:00:00:00:21 > 33:33:00:00:00:02, ethertype IPv6 (0x86dd),
length 70: fe80::200:ff:fe00:21 > ff02::2: ICMP6, router solicitation, length 16
22:12:23,219421 62:46:1f:4a:ee:db > 33:33:00:00:00:02, ethertype IPv6 (0x86dd),
length 70: fe80::6046:1fff:fe4a:eedb > ff02::2: ICMP6, router solicitation, leng
th 16
22:12:23,225495 00:00:00:00:00:03 > 33:33:00:00:00:02, ethertype IPv6 (0x86dd),
length 70: fe80::200:ff:fe00:3 > ff02::2: ICMP6, router solicitation, length 16
^C
9 packets captured
9 packets received by filter
0 packets dropped by kernel
root@mobinat:"/Desktop#
```

که نشان می دهد هاست به درستی عمل می کند.

توپولوژی صورت پروژه قبلی با حذف دور (امتیازی)

شکل توپولوژی:



توجه داشته باشید که مشخصات لینک ها در این شکل نیامده است. این توپولوژی در فایل topo1.py قرار دارد.

پنجه کنترلر در ابتدای کار:

```
xmobina@mobina: ~/Desktop
instantiating app ryu.topology.switches of Switches
instantiating app ryu.controller.ofp_handler of OFPHandler
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switch_features_handler is called
switches= [1, 2, 3, 4]
adding new entry to mac table: dpid = 2 in_port = 1
adding new entry to mac table: dpid = 3 in_port = 1
adding new entry to mac table: dpid = 1 in_port = 3
adding new entry to mac table: dpid = 1 in_port = 2
adding new entry to mac table: dpid = 2 in_port = 2
adding new entry to mac table: dpid = 3 in_port = 3
adding new entry to mac table: dpid = 4 in_port = 2
adding new entry to mac table: dpid = 4 in_port = 3
adding new entry to mac table: dpid = 4 in_port = 1
adding new entry to mac table: dpid = 1 in_port = 1
adding new entry to mac table: dpid = 4 in_port = 4
adding new entry to mac table: dpid = 3 in_port = 2
adding new entry to mac table: dpid = 3 in_port = 4
```

لایگ کنترلر جهت بررسی صحت اتصال سویچ ها به یکدیگر:

```
switches= [1, 2, 3, 4]
s3:4 connected to s4:4
s4:4 connected to s3:4
s2:2 connected to s1:2
s3:3 connected to s1:3
s1:2 connected to s2:2
s1:3 connected to s3:3
```

پنجره مینی نت در ابتدای کار که مشخصات لینک ها نیز در آن نشان داده می شود:

```

mobina@mobina: ~/Desktop
*** Add links
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7
*** Adding switches:
sw1 sw2 sw3 sw4
*** Adding links:
(1.00Mbit) (1.00Mbit) (h1, sw1) (2.00Mbit) (2.00Mbit) (h2, sw2) (3.00Mbit) (3.00Mbit)
(h3, sw3) (4.00Mbit) (4.00Mbit) (h4, sw3) (5.00Mbit) (5.00Mbit) (h5, sw4)
(1.00Mbit) (1.00Mbit) (h6, sw4) (2.00Mbit) (2.00Mbit) (h7, sw4) (3.00Mbit) (3.00Mbit)
(sw1, sw2) (4.00Mbit) (4.00Mbit) (sw1, sw3) (1.00Mbit) (1.00Mbit) (sw3, sw4)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7
*** Starting controller
c0
*** Starting 4 switches
sw1 sw2 sw3 sw4 ... (1.00Mbit) (3.00Mbit) (4.00Mbit) (2.00Mbit) (3.00Mbit) (3.00Mbit)
(4.00Mbit) (4.00Mbit) (1.00Mbit) (5.00Mbit) (1.00Mbit) (2.00Mbit) (1.00Mbit)
)
*** Starting CLI:
mininet> 

```

نتیجه اجرای دستور net در مینی نت:

```

mobina@mobina: ~/Desktop
*** Starting controller
c0
*** Starting 4 switches
sw1 sw2 sw3 sw4 ... (1.00Mbit) (3.00Mbit) (4.00Mbit) (2.00Mbit) (3.00Mbit) (3.00Mbit)
(4.00Mbit) (4.00Mbit) (1.00Mbit) (5.00Mbit) (1.00Mbit) (2.00Mbit) (1.00Mbit)
)
*** Starting CLI:
mininet> net
h1 h1-eth0:sw1-eth1
h2 h2-eth0:sw2-eth1
h3 h3-eth0:sw3-eth1
h4 h4-eth0:sw3-eth2
h5 h5-eth0:sw4-eth1
h6 h6-eth0:sw4-eth2
h7 h7-eth0:sw4-eth3
sw1 lo: sw1-eth1:h1-eth0 sw1-eth2:sw2-eth2 sw1-eth3:sw3-eth3
sw2 lo: sw2-eth1:h2-eth0 sw2-eth2:sw1-eth2
sw3 lo: sw3-eth1:h3-eth0 sw3-eth2:h4-eth0 sw3-eth3:sw1-eth3 sw3-eth4:sw4-eth4
sw4 lo: sw4-eth1:h5-eth0 sw4-eth2:h6-eth0 sw4-eth3:h7-eth0 sw4-eth4:sw3-eth4
c0
mininet> nodes
available nodes are:
c0 h1 h2 h3 h4 h5 h6 h7 sw1 sw2 sw3 sw4
mininet> 

```

نتیجه اجرای دستور pingall در مینی نت:

پنجره مینی نت:

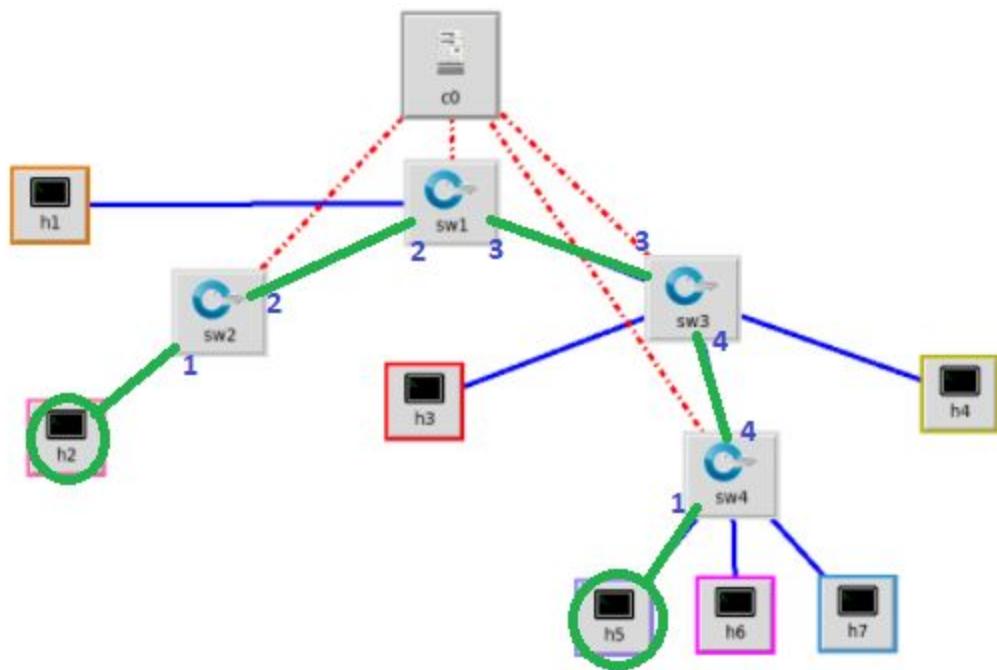
```
mobina@mobina: ~/Desktop
h3 h3-eth0:sw3-eth1
h4 h4-eth0:sw3-eth2
h5 h5-eth0:sw4-eth1
h6 h6-eth0:sw4-eth2
h7 h7-eth0:sw4-eth3
sw1 lo: sw1-eth1:h1-eth0 sw1-eth2:sw2-eth2 sw1-eth3:sw3-eth3
sw2 lo: sw2-eth1:h2-eth0 sw2-eth2:sw1-eth2
sw3 lo: sw3-eth1:h3-eth0 sw3-eth2:h4-eth0 sw3-eth3:sw1-eth3 sw3-eth4:sw4-eth4
sw4 lo: sw4-eth1:h5-eth0 sw4-eth2:h6-eth0 sw4-eth3:h7-eth0 sw4-eth4:sw3-eth4
c0
mininet> nodes
available nodes are:
c0 h1 h2 h3 h4 h5 h6 h7 sw1 sw2 sw3 sw4
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7
h2 -> h1 h3 h4 h5 h6 h7
h3 -> h1 h2 h4 h5 h6 h7
h4 -> h1 h2 h3 h5 h6 h7
h5 -> h1 h2 h3 h4 h6 h7
h6 -> h1 h2 h3 h4 h5 h7
h7 -> h1 h2 h3 h4 h5 h6
*** Results: 0% dropped (42/42 received)
mininet> 
```

همانطور که مشاهده می شود تمام بسته ها رسیده اند و هیچ بسته ای drop نشده است.

پنجره کنترلر:

```
> From l->s2 to s4->1
get_path src(switch: 2, port: 1) dst(switch: 4, port: 1)
install_path: 1->s2->2 2->s1->3 3->s3->4 4->s4->1
> From l->s2 to s4->1
get_path src(switch: 2, port: 1) dst(switch: 4, port: 1)
install_path: 1->s2->2 2->s1->3 3->s3->4 4->s4->1
> From 2->s4 to s2->1
get_path src(switch: 4, port: 2) dst(switch: 2, port: 1)
install_path: 2->s4->4 4->s3->3 3->s1->2 2->s2->1
> From 2->s4 to s2->1
get_path src(switch: 4, port: 2) dst(switch: 2, port: 1)
install_path: 2->s4->4 4->s3->3 3->s1->2 2->s2->1
> From l->s2 to s4->2
get_path src(switch: 2, port: 1) dst(switch: 4, port: 2)
install_path: 1->s2->2 2->s1->3 3->s3->4 4->s4->2
```

تمامی فرآیند ها مانند توپولوژی درخت با عمق 3 رخ می دهد که شرح آن در همان بخش موجود است. به عنوان مثال برای رفتن از هاست 2 (پورت 1 سوییچ 2) به هاست 5 (پورت 1 سوییچ 4) باید مسیر زیر را طی کرد:



که همان مسیریست که در خط اول خروجی پنجره کنترلر پیشتر نشان داده شده بود:

```
> From l->s2 to s4->l
get_path src(switch: 2, port: 1) dst(switch: 4, port: 1)
install_path:   l->s2->2  2->s1->3  3->s3->4  4->s4->l
```

پنجره سوییچ ها:

```
✖ - ○ "switch: sw2" (root)
root@mobina:~/Desktop# ovs-ofctl dump-flows sw2
  cookie=0x0, duration=106.968s, table=0, n_packets=119, n_bytes=7140, priority=6
  5535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:65535
  cookie=0x0, duration=106.974s, table=0, n_packets=75, n_bytes=7945, priority=0
  actions=CONTROLLER:65535
root@mobina:~/Desktop#
```

که نشان می دهد سویچ به درستی عمل می کند.

پنجره هاست ها:

```
✖ - ○ "host: h3"
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
22:09:34.313911 7a:3b:b1:78:48:db > 01:80:c2:00:00:0e, ethertype LLDP (0x88cc),
length 60: LLDP, length 46
22:09:35.214445 7a:3b:b1:78:48:db > 01:80:c2:00:00:0e, ethertype LLDP (0x88cc),
length 60: LLDP, length 46
22:09:35.247759 7a:1b:f2:c3:6f:a4 > 33:33:00:00:00:02, ethertype IPv6 (0x86dd),
length 70: fe80::781b:f2ff:fecc:6fa4 > ff02::2: ICMP6, router solicitation, leng
th 16
22:09:36.114757 7a:3b:b1:78:48:db > 01:80:c2:00:00:0e, ethertype LLDP (0x88cc),
length 60: LLDP, length 46
22:09:37.015018 7a:3b:b1:78:48:db > 01:80:c2:00:00:0e, ethertype LLDP (0x88cc),
length 60: LLDP, length 46
22:09:37.915297 7a:3b:b1:78:48:db > 01:80:c2:00:00:0e, ethertype LLDP (0x88cc),
length 60: LLDP, length 46
22:09:38.815554 7a:3b:b1:78:48:db > 01:80:c2:00:00:0e, ethertype LLDP (0x88cc),
length 60: LLDP, length 46
^C
7 packets captured
7 packets received by filter
0 packets dropped by kernel
root@mobina:~/Desktop#
```

که نشان می دهد هاست به درستی عمل می کند.

پهنهای باند تصادفی (امتیازی)

با توجه به نکات مطرح شده در صورت پروژه قبلی، پهنهای باند تمام لینک‌ها هر ۱۰ ثانیه باید تغییر کند و یک عدد تصادفی بین ۱ تا ۵ به هر کدام از آنها اختصاص یابد.

در این کد (فایل runner1.py)، از یک کلاس برای این منظور استفاده شده است. این کلاس، یک تابع (changeBW()) دارد که بر روی یک ترد مستقل، هر ۱۰ ثانیه اجرا می‌شود. شیوه‌ی کار به این صورت است که در تمام لینک‌ها، پهنهای باند را به یک مقدار رندوم بین ۱ تا ۵ تغییر می‌دهیم (پهنهای باند هر لینک در

```
net.links[i].intf1.params['bw']
```

ذخیره می‌شود).

همچنین بر روی یک ترد مستقل دیگر، هر ۵ ثانیه پهنهای باند هر لینک را چاپ می‌کنیم تا تغییر آن نمایش داده شود.

شبکه‌ی مورد استفاده دارای همان توپولوژی ای است که در قسمت قبل آمده است.

نتیجه اجرای کد به مدت ۴۰ ثانیه به صورت زیر است:

```
mobina@mobina:~/Desktop$ sudo python2 runner1.py
4 4 4 4 1 3 1 4 3 5
4 4 4 4 1 3 1 4 3 5
5 1 5 2 1 2 3 2 1 2
5 1 5 2 1 2 3 2 1 2
5 1 5 3 2 2 1 3 1 4
5 1 5 3 2 2 1 3 1 4
5 3 1 1 5 3 2 3 4 1
5 3 1 1 5 3 2 3 4 1
5 3 5 3 3 4 3 2 3 1
```

همان طور که مشاهده می شود پهنانی باند لینک ها هر ده ثانیه به صورت تصادفی مقدار دهی می شود.

...