CS339 Lab6

邬心远

519021910604

Q2: switch path

solve arp storm

To make the network work, we need firstly solve the ARP-storm, or the arp package will use up all the bandwidth.

The method I used to solve this contains two step. First, assure that only the ARP package from one specific port of a switch will broadcast, thus to avoid loop. A dictionary is used to realize this:

```
if (datapath.id, arp_src_ip, arp_dst_ip) in self.arp_in:
    # packet come back at different port.
    if self.arp_in[(datapath.id, arp_src_ip, arp_dst_ip)] != in_port:
        datapath.send_packet_out(in_port=in_port, actions=[])
        return True
    else:
        self.arp_in[(datapath.id, arp_src_ip, arp_dst_ip)] = in_port
```

Also, we need to answer to the ARP package:

```
opcode = arp_pkt.opcode
    if opcode == arp.ARP_REQUEST:
        arp_src_ip = arp_pkt.src_ip
        arp_dst_ip = arp_pkt.dst_ip

        if arp_dst_ip in self.arp_table:  # arp reply
            actions = [parser.OFPActionOutput(in_port)]
            ARP_Reply = packet.Packet()
            ARP_Reply.add_protocol(ethernet.ethernet(ethertype=eth.ethertype,
                                                     dst=eth.src,

src=self.arp_table[arp_dst_ip]))
            ARP_Reply.add_protocol(arp.arp(opcode=arp.ARP_REPLY,
                                           src_mac=self.arp_table[arp_dst_ip],
                                           src_ip=arp_dst_ip,
                                           dst_mac=eth_src, dst_ip=arp_src_ip))

            ARP_Reply.serialize()
            out = datapath.ofproto_parser.OFPPacketOut(datapath=datapath,

buffer_id=datapath.ofproto.OFP_NO_BUFFER,

in_port=datapath.ofproto.OFPP_CONTROLLER,
                                                       actions=actions,
data=ARP_Reply.data)
            datapath.send_msg(out)
```

Find topology and switch path:

We can use `networkx` and `topology_api` in ryu to build up the topology of the network. Then `nx.all_simple_path()` can be used to find all the path between h1 and h2. Then to tell the switch its next switch on path to find the outport. Part of the code to build up topology and fing outport is like following:

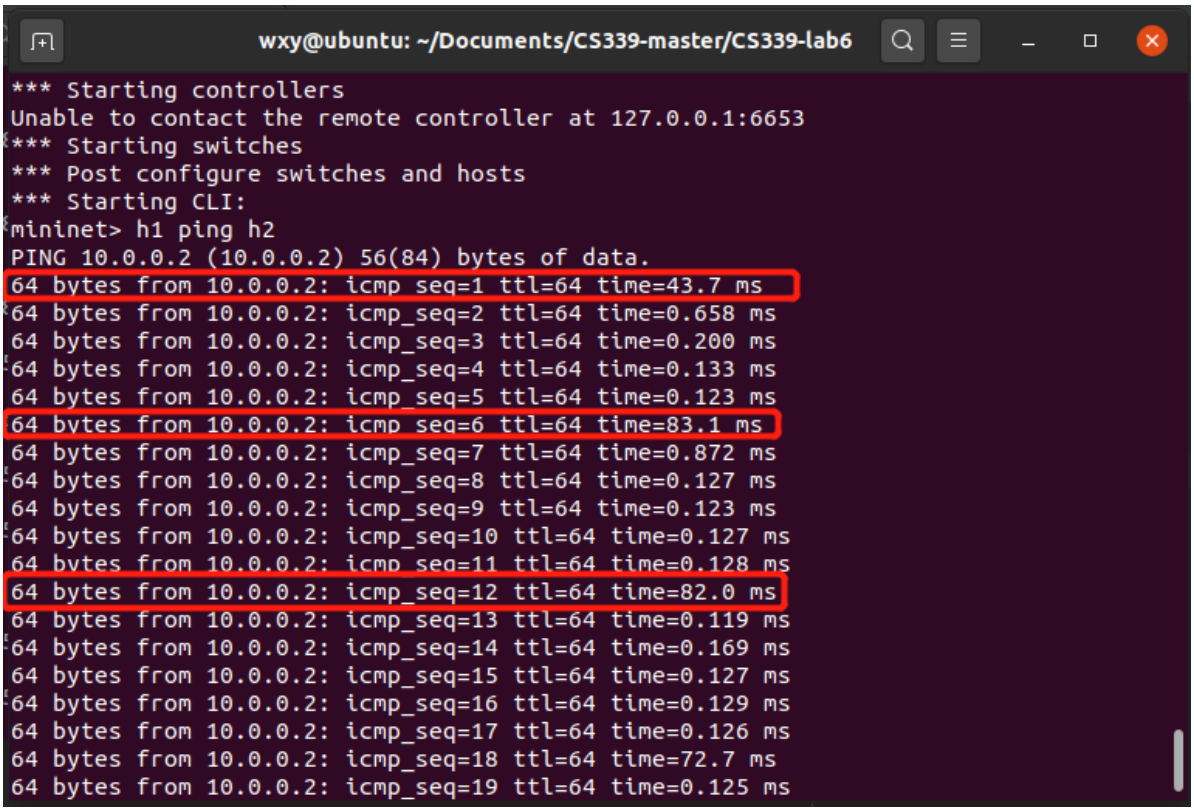
```
@set_ev_cls(event.EventSwitchEnter, [CONFIG_DISPATCHER, MAIN_DISPATCHER])
def get_topo(self, ev):
    switch_list = get_switch(self.topology_api_app)
    switches = [switch.dp.id for switch in switch_list]
    self.G.add_nodes_from(switches)

    link_list = get_link(self.topology_api_app)
    links = [(link.src.dpid, link.dst.dpid, {'attr_dict': {'port':
link.src.port_no}}) for link in link_list]
    self.G.add_edges_from(links)
    links = [(link.dst.dpid, link.src.dpid, {'attr_dict': {'port':
link.dst.port_no}}) for link in link_list]
    self.G.add_edges_from(links)

def get_out_port(self, datapath, src, dst, in_port):
    dpid = datapath.id
    if src not in self.G:
        self.G.add_node(src)
        self.G.add_edge(dpid, src, attr_dict={'port': in_port})
        self.G.add_edge(src, dpid)

    if dst in self.G:
        current_time = time.time()
        paths = list(nx.all_simple_paths(self.G, src, dst))
        if (current_time-self.last_switch_time) > 3:
            for p in paths:
                if p[2] != self.last_switch:
                    path = p
                    self.last_switch = path[2]
                    self.last_switch_time = current_time
                    break
        else:
            for p in paths:
                if p[2] == self.last_switch:
                    path = p
                    break
            next_hop = path[path.index(dpid) + 1]
            out_port = self.G[dpid][next_hop]['attr_dict']['port']
    else:
        out_port = datapath.ofproto.OFPP_FLOOD
        return out_port
```

Also, when adding flow to switches, I set hard_timeout to 5s, using

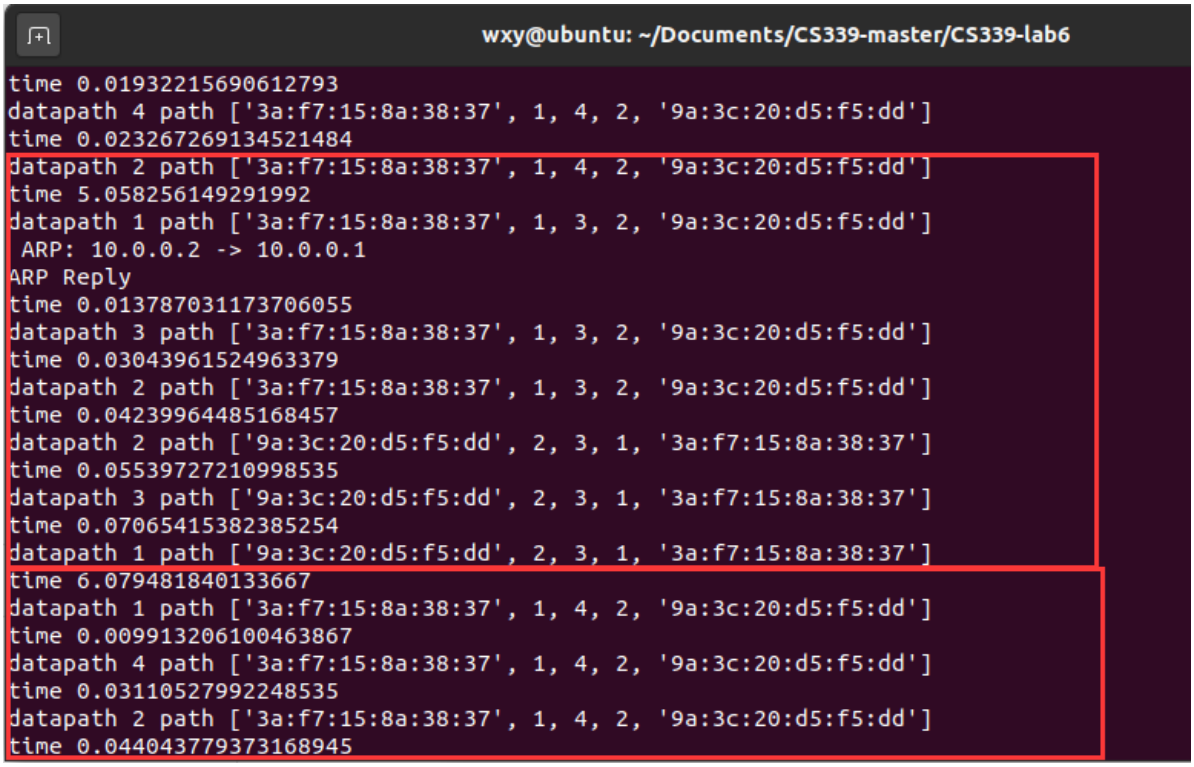
```
mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                        match=match, instructions=inst, hard_timeout=timeout)
```

so that after 5 seconds, the flow rule will expire and a PacketIn will happen and `get_out_port()` will give another path.

Here's the result of h1 ping h2, there is a obvious increasement of transimission time when path change:



Also, we can see how controller dispatch the path to switchers in controller's end:



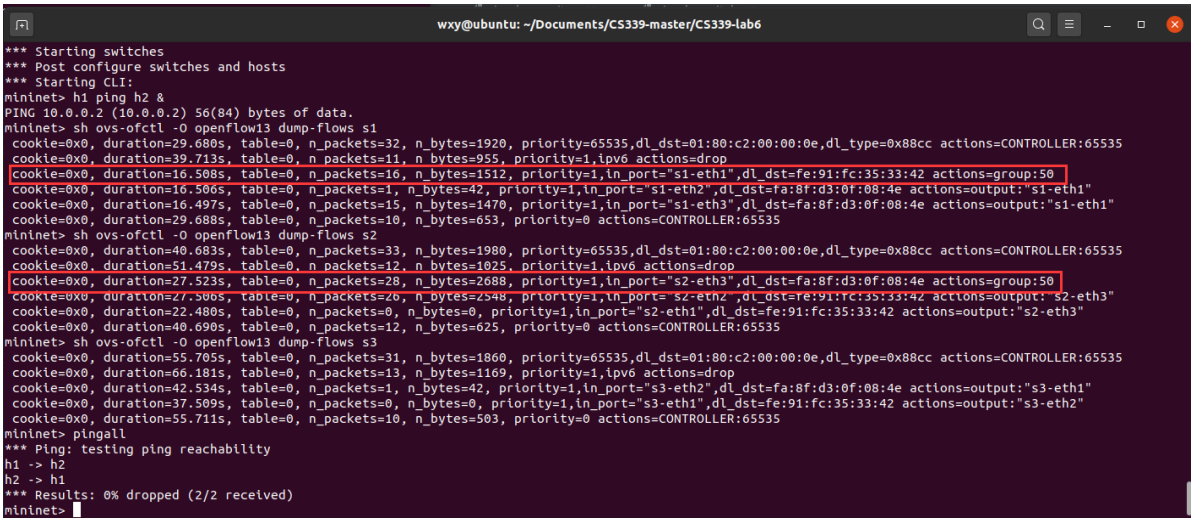
Q3: Using both paths

To using both paths, we can first change the `get_out_port()` function to return both ports if one more path is avaliable.

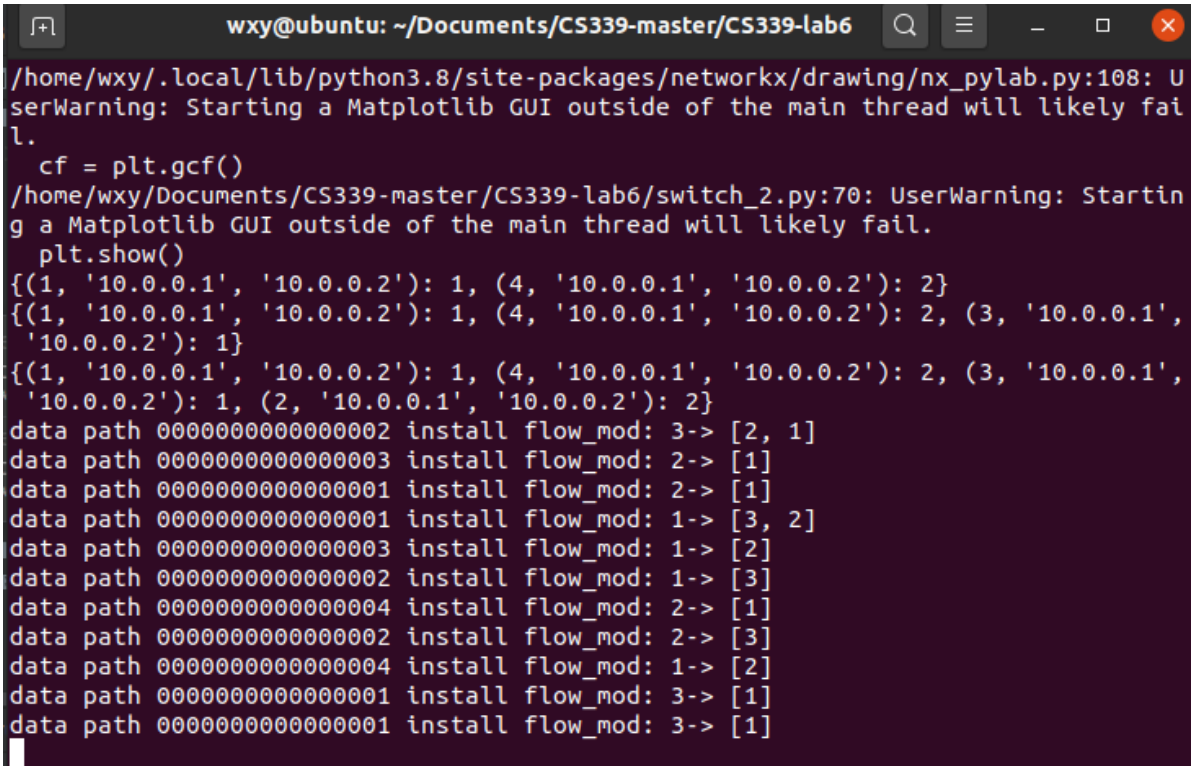
Then we can add group to flow table:

```
elif len(out_ports) == 2:
    actions1 = [ofp_parser.OFPActionOutput(out_ports[0])]
    actions2 = [ofp_parser.OFPActionOutput(out_ports[1])]
    weight1 = 50
    weight2 = 50
    watch_port = ofp.OFPP_ANY
    watch_group = ofp.OFPQ_ALL
    buckets = [ofp_parser.OFPBucket(weight1, watch_port, watch_group, actions1),
               ofp_parser.OFPBucket(weight2, watch_port, watch_group, actions2)]
    group_id = 50
    req = ofp_parser.OFPGroupMod(datapath, ofp.OFPGC_ADD, ofp.OFPGT_SELECT,
group_id, buckets)
    datapath.send_msg(req)
    actions = [ofp_parser.OFPActionGroup(group_id=group_id)]
```

After this, the flow table is like:



and we can see how controller add flow rule from the output information from controller:



Q4: fast failover

To send package from h1 to h2 in one path and return from another, just small modification on `get_out_port()` function is needed, it returns two ports in the order related to src and dst:

```
if dst in self.G:
    paths = list(nx.all_simple_paths(self.G, src, dst))
    if len(self.switch) == 0:
        for p in paths:
            self.switch.append(p[2])  # store s3 and s4
    for p in paths:
        if p[2] == self.switch[self.hosts.index(src)]:
            path = p
    next_hop = path[path.index(dpid) + 1]
    out_ports = [self.G[dpid][next_hop]['attr_dict']['port']]
    if next_hop in self.switch:
        next_hop_sub = self.switch[1-self.switch.index(next_hop)]
        out_ports.append(self.G[dpid][next_hop_sub]['attr_dict']['port'])
```

Then, we can use `OFPGT_FF` rule for group so package will change path when ine path is invalid:

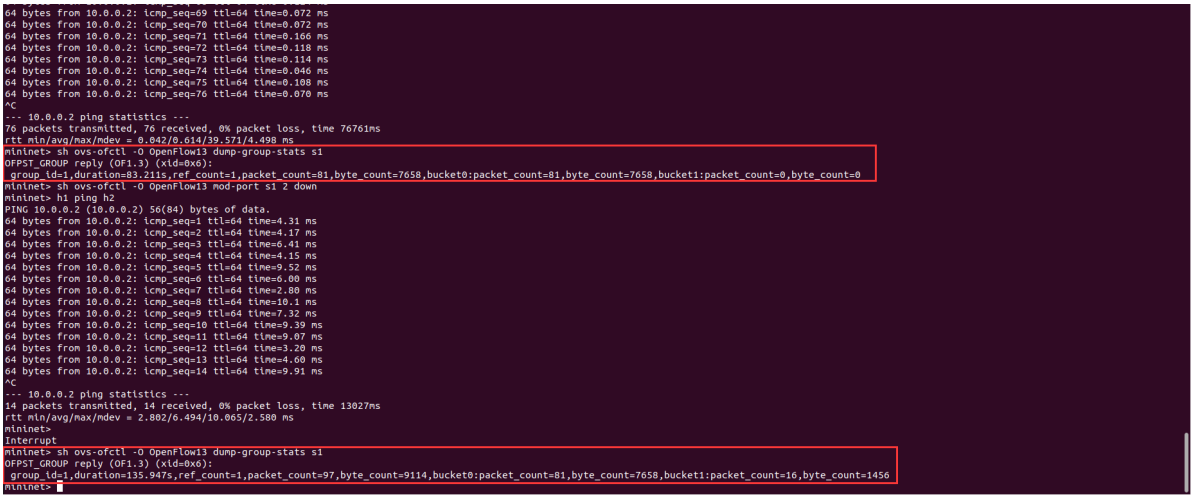
```
elif len(out_ports) == 2:
    action1 = [ofp_parser.OFPActionOutput(out_ports[0])]
    action2 = [ofp_parser.OFPActionOutput(out_ports[1])]
    buckets = [ofp_parser.OFPBucket(watch_port=out_ports[0], actions=action1),
               ofp_parser.OFPBucket(watch_port=out_ports[1], actions=action2)]
    group_id = int(dpid)
    req = ofp_parser.OFPGroupMod(datapath, ofp.OFPGC_ADD, ofp.OFPGT_FF, group_id,
buckets)
    datapath.send_msg(req)
    actions = [ofp_parser.OFPActionGroup(group_id=group_id)]
```

The final flow table is:

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
 cookie=0x0, duration=220.157s, table=0, n_packets=430, n_bytes=25800, priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x0, duration=293.073s, table=0, n_packets=21, n_bytes=1766, priority=1,ipv6 actions=drop
 cookie=0x0, duration=287.320s, table=0, n_packets=97, n_bytes=9114, priority=1,in_port="s1-eth3",dl_dst=da:c4:3b:f1:90:b2 actions=output:"s1-eth1"
 cookie=0x0, duration=287.317s, table=0, n_packets=96, n_bytes=9016, priority=1,in_port="s1-eth1",dl_dst=f6:f5:c5:87:64:f4 actions=group:1
 cookie=0x0, duration=220.162s, table=0, n_packets=6, n_bytes=456, priority=0 actions=CONTROLLER:65535
*** s2 ------------------------------------------------------------------------
 cookie=0x0, duration=220.162s, table=0, n_packets=634, n_bytes=38040, priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x0, duration=293.078s, table=0, n_packets=24, n_bytes=2050, priority=1,ipv6 actions=drop
 cookie=0x0, duration=287.336s, table=0, n_packets=97, n_bytes=9114, priority=1,in_port="s2-eth3",dl_dst=da:c4:3b:f1:90:b2 actions=group:2
 cookie=0x0, duration=287.315s, table=0, n_packets=80, n_bytes=7560, priority=1,in_port="s2-eth1",dl_dst=f6:f5:c5:87:64:f4 actions=output:"s2-eth3"
 cookie=0x0, duration=220.167s, table=0, n_packets=24, n_bytes=2014, priority=0 actions=CONTROLLER:65535
*** s3 ------------------------------------------------------------------------
 cookie=0x0, duration=220.165s, table=0, n_packets=427, n_bytes=25620, priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x0, duration=293.081s, table=0, n_packets=16, n_bytes=1416, priority=1,ipv6 actions=drop
 cookie=0x0, duration=287.322s, table=0, n_packets=80, n_bytes=7560, priority=1,in_port="s3-eth1",dl_dst=f6:f5:c5:87:64:f4 actions=output:"s3-eth2"
 cookie=0x0, duration=220.170s, table=0, n_packets=22, n_bytes=1912, priority=0 actions=CONTROLLER:65535
*** s4 ------------------------------------------------------------------------
 cookie=0x0, duration=220.172s, table=0, n_packets=633, n_bytes=37980, priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x0, duration=293.085s, table=0, n_packets=20, n_bytes=1770, priority=1,ipv6 actions=drop
 cookie=0x0, duration=287.337s, table=0, n_packets=97, n_bytes=9114, priority=1,in_port="s4-eth1",dl_dst=da:c4:3b:f1:90:b2 actions=output:"s4-eth2"
 cookie=0x0, duration=220.176s, table=0, n_packets=20, n_bytes=1754, priority=0 actions=CONTROLLER:65535
mininet>
```

We can see that s3 and s4 only has flow rules in one direction.

We can do a simple test in mininet:



when the network work properly, packets from h1 to h2 go through s3, and after we shut down a port, it will ago through s4.