

# 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 output. Part of the code to build up topology and find output 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:

```
wxy@ubuntu: ~/Documents/CS339-master/CS339-lab6
*** Starting controllers
Unable to contact the remote controller at 127.0.0.1:6653
*** Starting switches
*** Post configure switches and hosts
*** Starting CLI:
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=43.7 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.658 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.200 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.133 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.123 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=83.1 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.872 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.127 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.123 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.127 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.128 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=82.0 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=0.119 ms
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=0.169 ms
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=0.127 ms
64 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=0.129 ms
64 bytes from 10.0.0.2: icmp_seq=17 ttl=64 time=0.126 ms
64 bytes from 10.0.0.2: icmp_seq=18 ttl=64 time=72.7 ms
64 bytes from 10.0.0.2: icmp_seq=19 ttl=64 time=0.125 ms
```

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

```
wxy@ubuntu: ~/Documents/CS339-master/CS339-lab6
time 0.01932215690612793
datapath 4 path ['3a:f7:15:8a:38:37', 1, 4, 2, '9a:3c:20:d5:f5:dd']
time 0.023267269134521484
datapath 2 path ['3a:f7:15:8a:38:37', 1, 4, 2, '9a:3c:20:d5:f5:dd']
time 5.058256149291992
datapath 1 path ['3a:f7:15:8a:38:37', 1, 3, 2, '9a:3c:20:d5:f5:dd']
ARP: 10.0.0.2 -> 10.0.0.1
ARP Reply
time 0.013787031173706055
datapath 3 path ['3a:f7:15:8a:38:37', 1, 3, 2, '9a:3c:20:d5:f5:dd']
time 0.03043961524963379
datapath 2 path ['3a:f7:15:8a:38:37', 1, 3, 2, '9a:3c:20:d5:f5:dd']
time 0.04239964485168457
datapath 2 path ['9a:3c:20:d5:f5:dd', 2, 3, 1, '3a:f7:15:8a:38:37']
time 0.05539727210998535
datapath 3 path ['9a:3c:20:d5:f5:dd', 2, 3, 1, '3a:f7:15:8a:38:37']
time 0.07065415382385254
datapath 1 path ['9a:3c:20:d5:f5:dd', 2, 3, 1, '3a:f7:15:8a:38:37']
time 6.079481840133667
datapath 1 path ['3a:f7:15:8a:38:37', 1, 4, 2, '9a:3c:20:d5:f5:dd']
time 0.009913206100463867
datapath 4 path ['3a:f7:15:8a:38:37', 1, 4, 2, '9a:3c:20:d5:f5:dd']
time 0.03110527992248535
datapath 2 path ['3a:f7:15:8a:38:37', 1, 4, 2, '9a:3c:20:d5:f5:dd']
time 0.044043779373168945
```

## 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 available.

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:

```
wxy@ubuntu: ~/Documents/CS339-master/CS339-lab6
*** Starting switches
*** Post configure switches and hosts
*** Starting CLI:
mininet> h1 ping h2 8
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
mininet> sh ovs-ofctl -O openflow13 dump-flows s1
cookie=0x0, duration=29.680s, table=0, n_packets=32, n_bytes=1920, priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x0, duration=39.713s, table=0, n_packets=11, n_bytes=955, priority=1,ipv6 actions=drop
cookie=0x0, duration=16.508s, table=0, n_packets=16, n_bytes=1512, priority=1,in_port="s1-eth1",dl_dst=fe:91:fc:35:33:42 actions=group:50
cookie=0x0, duration=16.506s, table=0, n_packets=1, n_bytes=42, priority=1,in_port="s1-eth2",dl_dst=fa:8f:d3:0f:08:4e actions=output:"s1-eth1"
cookie=0x0, duration=16.497s, table=0, n_packets=15, n_bytes=1470, priority=1,in_port="s1-eth3",dl_dst=fa:8f:d3:0f:08:4e actions=output:"s1-eth1"
cookie=0x0, duration=29.688s, table=0, n_packets=10, n_bytes=653, priority=0 actions=CONTROLLER:65535
mininet> sh ovs-ofctl -O openflow13 dump-flows s2
cookie=0x0, duration=40.683s, table=0, n_packets=33, n_bytes=1980, priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x0, duration=51.479s, table=0, n_packets=12, n_bytes=1025, priority=1,ipv6 actions=drop
cookie=0x0, duration=27.523s, table=0, n_packets=28, n_bytes=2688, priority=1,in_port="s2-eth3",dl_dst=fa:8f:d3:0f:08:4e actions=group:50
cookie=0x0, duration=27.500s, table=0, n_packets=20, n_bytes=2548, priority=1,in_port="s2-eth2",dl_dst=fe:91:fc:35:33:42 actions=output:"s2-eth3"
cookie=0x0, duration=22.480s, table=0, n_packets=0, n_bytes=0, priority=1,in_port="s2-eth1",dl_dst=fe:91:fc:35:33:42 actions=output:"s2-eth3"
cookie=0x0, duration=40.690s, table=0, n_packets=12, n_bytes=625, priority=0 actions=CONTROLLER:65535
mininet> sh ovs-ofctl -O openflow13 dump-flows s3
cookie=0x0, duration=55.705s, table=0, n_packets=31, n_bytes=1860, priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x0, duration=66.181s, table=0, n_packets=13, n_bytes=1169, priority=1,ipv6 actions=drop
cookie=0x0, duration=42.534s, table=0, n_packets=1, n_bytes=42, priority=1,in_port="s3-eth2",dl_dst=fa:8f:d3:0f:08:4e actions=output:"s3-eth1"
cookie=0x0, duration=37.509s, table=0, n_packets=0, n_bytes=0, priority=1,in_port="s3-eth1",dl_dst=fe:91:fc:35:33:42 actions=output:"s3-eth2"
cookie=0x0, duration=55.711s, table=0, n_packets=10, n_bytes=503, priority=0 actions=CONTROLLER:65535
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

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

```
wxy@ubuntu: ~/Documents/CS339-master/CS339-lab6
/home/wxy/.local/lib/python3.8/site-packages/networkx/drawing/nx_pylab.py:108: UserWarning: Starting a Matplotlib GUI outside of the main thread will likely fail.
  cf = plt.gcf()
/home/wxy/Documents/CS339-master/CS339-lab6/switch_2.py:70: UserWarning: Starting a Matplotlib GUI outside of the main thread will likely fail.
  plt.show()
{(1, '10.0.0.1', '10.0.0.2'): 1, (4, '10.0.0.1', '10.0.0.2'): 2}
{(1, '10.0.0.1', '10.0.0.2'): 1, (4, '10.0.0.1', '10.0.0.2'): 2, (3, '10.0.0.1', '10.0.0.2'): 1}
{(1, '10.0.0.1', '10.0.0.2'): 1, (4, '10.0.0.1', '10.0.0.2'): 2, (3, '10.0.0.1', '10.0.0.2'): 1, (2, '10.0.0.1', '10.0.0.2'): 2}
data path 0000000000000002 install flow_mod: 3-> [2, 1]
data path 0000000000000003 install flow_mod: 2-> [1]
data path 0000000000000001 install flow_mod: 2-> [1]
data path 0000000000000001 install flow_mod: 1-> [3, 2]
data path 0000000000000003 install flow_mod: 1-> [2]
data path 0000000000000002 install flow_mod: 1-> [3]
data path 0000000000000004 install flow_mod: 2-> [1]
data path 0000000000000002 install flow_mod: 2-> [3]
data path 0000000000000004 install flow_mod: 1-> [2]
data path 0000000000000001 install flow_mod: 3-> [1]
data path 0000000000000001 install flow_mod: 3-> [1]
```

## 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(dp_id) + 1]
    out_ports = [self.G[dp_id][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[dp_id][next_hop_sub]['attr_dict']['port'])
```

Then, we can use `OFPGT_FF` rule for group so package will change path when one 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(dp_id)
    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> dptctl dump-flows
*** s1 ***
cookie=0x0, duration=220.157s, table=0, n_packets=430, n_bytes=25800, priority=65535,d_l_dst=01:80:c2:00:00:0e,d_l_type=0x80cc 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",d_l_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",d_l_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,d_l_dst=01:80:c2:00:00:0e,d_l_type=0x80cc 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",d_l_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",d_l_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,d_l_dst=01:80:c2:00:00:0e,d_l_type=0x80cc 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",d_l_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,d_l_dst=01:80:c2:00:00:0e,d_l_type=0x80cc 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",d_l_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:

```

64 bytes from 10.0.0.2: icmp_seq=69 ttl=64 time=0.072 ms
64 bytes from 10.0.0.2: icmp_seq=70 ttl=64 time=0.072 ms
64 bytes from 10.0.0.2: icmp_seq=71 ttl=64 time=0.166 ms
64 bytes from 10.0.0.2: icmp_seq=72 ttl=64 time=0.118 ms
64 bytes from 10.0.0.2: icmp_seq=73 ttl=64 time=0.114 ms
64 bytes from 10.0.0.2: icmp_seq=74 ttl=64 time=0.046 ms
64 bytes from 10.0.0.2: icmp_seq=75 ttl=64 time=0.108 ms
64 bytes from 10.0.0.2: icmp_seq=76 ttl=64 time=0.070 ms
^C
--- 10.0.0.2 ping statistics ---
76 packets transmitted, 76 received, 0% packet loss, time 7676ms
rtt min/avg/max/mdev = 0.042/0.614/39.571/4.498 ms
mininet> sh ovs-ofctl -O OpenFlow13 dump-group-stats s1
OFPST_GROUP reply (OF1.3) (xid=0x0):
  group_id=1,duration=83.211s,ref_count=1,packet_count=81,byte_count=7658,bucket0:packet_count=81,byte_count=7658,bucket1:packet_count=0,byte_count=0
mininet> sh ovs-ofctl -O OpenFlow13 mod-port s1 2 down
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=4.31 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=4.17 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=6.41 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=4.15 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=9.52 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=9.60 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=2.80 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=10.1 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=7.32 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=9.39 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=9.07 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=3.20 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=4.68 ms
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=9.91 ms
^C
--- 10.0.0.2 ping statistics ---
14 packets transmitted, 14 received, 0% packet loss, time 13027ms
rtt min/avg/max/mdev = 2.802/6.494/10.065/2.580 ms
mininet>
Interrupt
mininet> sh ovs-ofctl -O OpenFlow13 dump-group-stats s1
OFPST_GROUP reply (OF1.3) (xid=0x0):
  group_id=1,duration=135.947s,ref_count=1,packet_count=97,byte_count=9114,bucket0:packet_count=81,byte_count=7658,bucket1:packet_count=16,byte_count=1456
mininet>

```

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