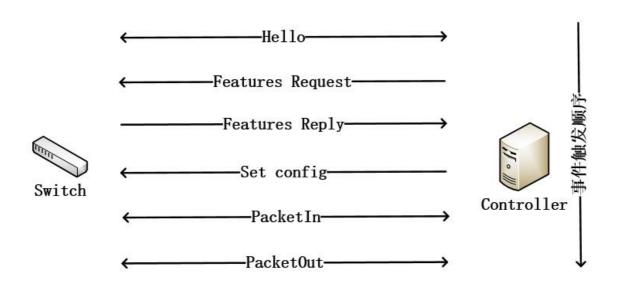
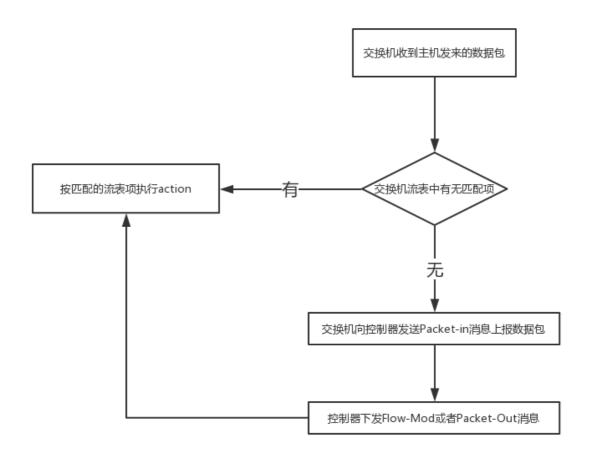
Lab: RYU Programming

1. The Step of Openflow switch interacting with controller





Assignments

Topo

Requirements

Set up the following network first:

```
s3
/ \
h1 - s1 s2 - h2
\ /
```

```
#!/usr/bin/python
    from mininet.topo import Topo
    from mininet.net import Mininet
 3
    from mininet.log import setLogLevel
    from mininet.cli import CLI
 5
    from mininet.node import OVSSwitch, Controller, RemoteController
 6
    from time import sleep
 7
 8
 9
    class SingleSwitchTopo(Topo):
10
11
        "Single switch connected to n hosts."
        def build(self):
12
13
            s1 = self.addSwitch('s1')
            s2 = self.addSwitch('s2')
14
            s3 = self.addSwitch('s3')
15
16
            s4 = self.addSwitch('s4')
17
18
            h1 = self.addHost('h1')
            h2 = self.addHost('h2')
19
            h1 = self.addHost('h1')
20
21
            h2 = self.addHost('h2')
            self.addLink(s1,s4,1,1)
22
            self.addLink(s1,s3,2,1)
23
24
            self.addLink(s4,s2,2,2)
            self.addLink(s3,s2,2,1)
25
26
            self.addLink(s1,h1,3,1)
27
            self.addLink(s2,h2,3,1)
28
    if __name__ == '__main__':
29
        setLogLevel('info')
30
31
        topo = SingleSwitchTopo()
        c1 = RemoteController('c1', ip='127.0.0.1')
32
33
        net = Mininet(topo=topo, controller=c1)
        net.start()
34
35
        CLI(net)
        net.stop()
36
```

auto switch per 5secs

Requirements

Write a RYU controller that switches paths (h1-s1-s3-s2-h2 or h1-s1-s4-s2-h2) between h1 and h2 every 5 seconds.

Ideas

- 1. set a hard timeout flow.
- 2. after 5 seconds the timer is out, it will trigger the event ofp_event.EventOFPFlowRemoved
- 3. set a _flow_removed_handler to add new flow with another 5secs hard_timeout to the data path.

Code

```
1 #http://docs.openvswitch.org/en/latest/faq/openflow/
   #https://mail.openvswitch.org/pipermail/ovs-discuss/2016-August/042394.html
 3 #https://stackoverflow.com/questions/36949861/group-table-issue-openflow-
    mininet
 5 from ryu.base import app_manager
 6 from ryu.controller import ofp_event
   from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
 7
 8
   from ryu.controller.handler import set_ev_cls
    from ryu.ofproto import ofproto_v1_3
 9
   from ryu.lib.packet import packet
   from ryu.lib.packet import ethernet
   from ryu.lib.packet import ether_types
12
    s1op = 1
13
14
    s2op = 1
15
16
   class SimpleSwitch13(app_manager.RyuApp):
17
        OFP_VERSIONS = [ofproto_v1_3.0FP_VERSION]
18
        def __init__(self, *args, **kwargs):
19
20
            super(SimpleSwitch13, self).__init__(*args, **kwargs)
21
            self.mac_to_port = {}
22
        def add_flow_tmp(self, datapath, priority, match, actions,
    buffer_id=None):
23
            ofproto = datapath.ofproto
24
            parser = datapath.ofproto_parser
25
26
            inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
27
                                                  actions)]
            if buffer_id:
28
                mod = parser.OFPFlowMod(datapath=datapath,
29
    buffer_id=buffer_id, hard_timeout=5,
30
     flags=ofproto.OFPFF_SEND_FLOW_REM, priority=priority, match=match,
31
                                        instructions=inst)
32
            else:
                mod = parser.OFPFlowMod(datapath=datapath,
33
    priority=priority, hard_timeout=5,
34
     flags=ofproto.OFPFF_SEND_FLOW_REM, match=match, instructions=inst)
```

```
35
            datapath.send_msg(mod)
36
        @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
37
38
        def switch_features_handler(self, ev):
            global slop
39
40
            global s2op
            datapath = ev.msg.datapath
41
            ofproto = datapath.ofproto
42
            parser = datapath.ofproto_parser
43
44
45
            # install table-miss flow entry
            1.1.1
46
            match = parser.OFPMatch()
47
            actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
48
                                                ofproto.OFPCML_NO_BUFFER)]
49
            self.add_flow(datapath, 0, match, actions)
50
            1.1.1
51
            # switch s1
52
            if datapath.id == 1:
53
54
                #add the return flow for h1 in s1.
55
56
                # h1 is connected to port 3.
57
                actions = [parser.OFPActionOutput(s1op)]
                match = parser.OFPMatch(in_port=3)
58
                self.add_flow_tmp(datapath, 10, match, actions)
59
60
                op = 2
                actions = [parser.OFPActionOutput(3)]
61
62
                match = parser.OFPMatch(in_port=1)
                self.add_flow(datapath, 10, match, actions)
63
64
65
                actions = [parser.OFPActionOutput(3)]
                match = parser.OFPMatch(in_port=2)
67
                self.add_flow(datapath, 10, match, actions)
68
69
            # switch s2
70
            if datapath.id == 2:
71
                actions = [parser.OFPActionOutput(s2op)]
72
73
                match = parser.OFPMatch(in_port=3)
                self.add_flow_tmp(datapath, 10, match, actions)
74
75
                op = 2
                #add the return flow for h2 in s4.
76
                # h2 is connected to port 3.
77
78
                actions = [parser.OFPActionOutput(3)]
                match = parser.OFPMatch(in_port=1)
79
80
                self.add_flow(datapath, 10, match, actions)
81
82
                actions = [parser.OFPActionOutput(3)]
                match = parser.OFPMatch(in_port=2)
83
                self.add_flow(datapath, 10, match, actions)
84
85
86
87
            # switch s4
            if datapath.id == 4:
89
90
                actions = [parser.OFPActionOutput(2)]
                match = parser.OFPMatch(in_port=1)
91
92
                self.add_flow(datapath, 10, match, actions)
```

```
93
 94
                 actions = [parser.OFPActionOutput(1)]
 95
                 match = parser.OFPMatch(in_port=2)
 96
                  self.add_flow(datapath, 10, match, actions)
 97
 98
             # switch s3
 99
100
             if datapath.id == 3:
                 # h1 is connected to port 3.
101
102
                 actions = [parser.OFPActionOutput(2)]
                 match = parser.OFPMatch(in_port=1)
103
104
                  self.add_flow(datapath, 10, match, actions)
105
106
                 actions = [parser.OFPActionOutput(1)]
107
                 match = parser.OFPMatch(in_port=2)
                 self.add_flow(datapath, 10, match, actions)
108
109
         def add_flow(self, datapath, priority, match, actions, buffer_id=None):
110
             ofproto = datapath.ofproto
111
             parser = datapath.ofproto_parser
112
113
114
             inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
115
                                                    actions)]
             if buffer_id:
116
                 mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
117
118
                                          priority=priority, match=match,
119
                                           instructions=inst)
120
             else:
                 mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
121
                                          match=match, instructions=inst)
122
123
             datapath.send_msg(mod)
124
125
         @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
126
         def _packet_in_handler(self, ev):
127
128
             # If you hit this you might want to increase
             # the "miss_send_length" of your switch
129
             if ev.msg.msg_len < ev.msg.total_len:</pre>
130
131
                  self.logger.debug("packet truncated: only %s of %s bytes",
                                    ev.msg.msg_len, ev.msg.total_len)
132
133
             msg = ev.msg
             datapath = msg.datapath
134
             ofproto = datapath.ofproto
135
136
             parser = datapath.ofproto_parser
             in_port = msg .match['in_port']
137
138
             pkt = packet.Packet(msg.data)
139
140
             eth = pkt.get_protocols(ethernet.ethernet)[0]
141
             if eth.ethertype == ether_types.ETH_TYPE_LLDP:
142
143
                 # ignore lldp packet
144
                 return
145
             dst = eth.dst
             src = eth.src
146
147
             dpid = datapath.id
148
             if dst[:5] == "33:3":
149
150
                  self.logger.info("drop ipv6 multicast packet %s", dst)
```

```
151
                  return
152
153
             self.mac_to_port.setdefault(dpid, {})
154
             self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)
155
156
             if(dpid==1 \text{ or dpid} == 4):
157
158
                  if(op==1):
                      out_port=2
159
160
                      op = 2
                  else:
161
162
                      out_port = 1
163
                      op = 1
164
             actions = [parser.OFPActionOutput(out_port)]
165
             # install a flow to avoid packet_in next time
166
167
             if out_port != ofproto.OFPP_FLOOD:
168
                  match = parser.OFPMatch(in_port=in_port, eth_dst=dst,
     eth_src=src)
169
                  # verify if we have a valid buffer_id, if yes avoid to send both
                  # flow_mod & packet_out
170
                  if msg.buffer_id != ofproto.OFP_NO_BUFFER:
171
172
                      self.add_flow_tmp(datapath, 1, match, actions,
     msg.buffer_id)
173
                      return
                  else:
174
175
                      self.add_flow_tmp(datapath, 1, match, actions)
176
             data = None
             if msg.buffer_id == ofproto.OFP_NO_BUFFER:
177
                  data = msg.data
178
179
             out = parser.OFPPacketOut(datapath=datapath,
180
     buffer_id=msg.buffer_id,
                                         in_port=in_port, actions=actions,
181
     data=data)
182
             datapath.send_msg(out)
183
184
         @set_ev_cls(ofp_event.EventOFPFlowRemoved, MAIN_DISPATCHER)
185
         def _flow_removed_handler(self, ev):
186
             global slop
187
             global s2op
188
             msg = ev.msg
             dp = msg.datapath
189
190
             ofp = dp.ofproto
             parser = dp.ofproto_parser
191
             if msg.reason == ofp.OFPRR_IDLE_TIMEOUT:
192
193
                  reason = 'IDLE TIMEOUT'
194
             elif msg.reason == ofp.OFPRR_HARD_TIMEOUT:
                      reason = 'HARD TIMEOUT'
195
             elif msg.reason == ofp.OFPRR_DELETE:
196
197
                  reason = 'DELETE'
             elif msg.reason == ofp.OFPRR_GROUP_DELETE:
198
199
                  reason = 'GROUP DELETE'
200
             else:
201
                  reason = 'unknown'
202
             self.logger.debug('OFPFlowRemoved received: ''datapath id=%s'
203
204
                                     'cookie=%d priority=%d reason=%s table_id=%d '
```

```
205
                                    'duration_sec=%d duration_nsec=%d
                                    'idle_timeout=%d hard_timeout=%d '
207
                                    'packet_count=%d byte_count=%d
     match.fields=%s',
208
                                    dp.id, msg.cookie, msg.priority, reason,
     msg.table_id,
                                    msg.duration_sec, msg.duration_nsec,
209
210
                                    msg.idle_timeout, msg.hard_timeout,
                                    msg.packet_count, msg.byte_count, msg.match)
211
212
             if(dp.id==1):
                 if(s1op==1):
213
                      s1op = 2
214
215
                 else:
216
                     s1op = 1
217
             print("slop is "+str(slop))
             actions = [parser.OFPActionOutput(s1op)]
218
219
             match = parser.OFPMatch(in_port=3)
             self.add_flow_tmp(dp, 10, match, actions)
220
```

Test and Result

1. start the controller

```
ryu-manager --verbose per5s.py
```

2. start net topo and connect to the controller

```
sudo python topo.py --controller remote
```

In mininet using h1 ping h2 test the connections. Using sudo ovs-ofct1 -0 OpenFlow13 dump-flows s1 check the flow table.

Mininet Result

```
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=0.991 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.088 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.089 ms
64 bytes from 10.0.0.2: icmp seq=4 ttl=64 time=0.087 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.665 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.089 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.095 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.089 ms
64 bytes from 10.0.0.2: icmp seq=9 ttl=64 time=0.092 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.090 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.090 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=0.091 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=0.090 ms
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=0.091 ms
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=0.091 ms
64 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=0.093 ms
64 bytes from 10.0.0.2: icmp_seq=17 ttl=64 time=0.090 ms
64 bytes from 10.0.0.2: icmp_seq=18 ttl=64 time=0.089 ms
`C
--- 10.0.0.2 ping statistics ---
18 packets transmitted, 18 received, 0% packet loss, time 17393ms
rtt min/avg/max/mdev = 0.087/0.172/0.991/0.238 ms
```

```
EVENT ofp_event->SimpleSwitch13 EventOFPFlowRemoved
DPFPLowRemoved received: datapath id=lcookte=0 priority=10 reason=HARD TIMEOUT table_id=0 duration_sec=5 duration_nsec=16000000 idle_timeout=0 hard_timeout=0 for priority=10 reason=HARD TIMEOUT table_id=0 duration_sec=5 duration_nsec=16000000 idle_timeout=0 hard_timeout=0 for priority=10 reason=HARD TIMEOUT table_id=0 duration_sec=5 duration_nsec=18000000 idle_timeout=0 hard_timeout=0 for priority=10 reason=HARD TIMEOUT table_id=0 duration_sec=5 duration_nsec=18000000 idle_timeout=0 hard_timeout=0 for priority=10 reason=HARD TIMEOUT table_id=0 duration_sec=5 duration_nsec=18000000 idle_timeout=0 hard_timeout=0 for priority=10 reason=HARD TIMEOUT table_id=0 duration_sec=5 duration_nsec=17000000 idle_timeout=0 hard_timeout=5 packet_count=400 match.ftelds=0FPMatch(oxm_fields=('in_port': 3))
slop is 2
EVENT ofp_event->SimpleSwitch13 EventOFPFlowRemoved
DFPFlowRemoved received: datapath id=2cookte=0 priority=10 reason=HARD TIMEOUT table_id=0 duration_sec=5 duration_nsec=19000000 idle_timeout=0 hard_timeout=0 for priority=10 reason=HARD TIMEOUT table_id=0 duration_sec=5 duration_nsec=19000000 idle_timeout=0 hard_timeout=0 for priority=10 reason=HARD TIMEOUT table_id=0 duration_sec=5 duration_nsec=19000000 idle_timeout=0 hard_timeout=0 for priority=10 reason=HARD TIMEOUT table_id=0 duration_sec=5 duration_nsec=19000000 idle_timeout=0 hard_timeout=0 for priority=10 reason=HARD TIMEOUT table_id=0 duration_sec=5 duration_nsec=19000000 idle_timeout=0 hard_timeout=0 for priority=10 reason=HARD TIMEOUT table_id=0 duration_sec=5 duration_nsec=19000000 idle_timeout=0 hard_timeout=0 for priority=10 reason=1 for priority=10 for pri
```

It show the event triggered the handler and made echo.

Flow table:

```
austinguish@austinguish=0i.502VSK ~/果/C/c/i/r/rapp (master)> sudo ovs-ofctl -0 OpenFlow13 dump-flows s2 cookie=0x0, duration=23.995s, table=0, n_packets=83, n_bytes=11974, priority=10,in_port="s2-eth1" actions=output:"s2-eth3" cookie=0x0, duration=23.995s, table=0, n_packets=122, n_bytes=15004, priority=10,in_port="s2-eth2" actions=output:"s2-eth3" cookie=0x0, duration=3.919s, table=0, n_packets=3, n_bytes=294, hard_timeout=5, send_flow_rem priority=10,in_port="s2-eth3" actions=output: "s2-eth1" austinguish@austinguish=0i.502VSK ~/果/C/c/l/r/rapp (master)> sudo ovs-ofctl -0 OpenFlow13 dump-flows s2 cookie=0x0, duration=25.225s, table=0, n_packets=83, n_bytes=11974, priority=10,in_port="s2-eth1" actions=output: "s2-eth3" cookie=0x0, duration=25.225s, table=0, n_packets=124, n_bytes=15000, priority=10,in_port="s2-eth2" actions=output: "s2-eth3" cookie=0x0, duration=0.129s, table=0, n_packets=0, n_bytes=0, hard_timeout=5, send_flow_rem priority=10,in_port="s2-eth3" actions=output: "s2-eth3" act
```

Load Balancer

Requirements

Write a RYU controller that uses both paths to forward packets from h1 to h2.

Ideas:

- 1. set the basic flow in switch_features_handler
- 2. set a group table to handle the forwarding.
- 3. It will have a weight to select action in buckets.
- 4. using iperf -P set multi connection

Code

```
1 from ryu.base import app_manager
2 from ryu.controller import ofp_event
 3 from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
4 from ryu.controller.handler import set_ev_cls
5 from ryu.ofproto import ofproto_v1_3
   from ryu.lib.packet import packet
7
   from ryu.lib.packet import ethernet
   from ryu.lib.packet import ether_types
    class SimpleSwitch13(app_manager.RyuApp):
10
        OFP_VERSIONS = [ofproto_v1_3.0FP_VERSION]
11
12
        def __init__(self, *args, **kwargs):
14
            super(SimpleSwitch13, self).__init__(*args, **kwargs)
            self.mac_to_port = {}
15
16
17
        @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
18
        def switch_features_handler(self, ev):
            datapath = ev.msg.datapath
19
            ofproto = datapath.ofproto
20
            parser = datapath.ofproto_parser
21
            if datapath.id == 1:
22
23
                # add group tables
                self.send_group_mod(datapath)
25
                actions = [parser.OFPActionGroup(group_id=50)]
26
                match = parser.OFPMatch(in_port=3)
```

```
27
                self.add_flow(datapath, 10, match, actions)
28
                #add the return flow for h1 in s1.
29
30
                # h1 is connected to port 3.
                actions = [parser.OFPActionOutput(3)]
31
                match = parser.OFPMatch(in_port=1)
32
                self.add_flow(datapath, 10, match, actions)
33
34
                actions = [parser.OFPActionOutput(3)]
35
                match = parser.OFPMatch(in_port=2)
36
                self.add_flow(datapath, 10, match, actions)
37
38
            # switch s2
40
            if datapath.id == 2:
41
            # add group tables
42
43
                self.send_group_mod(datapath)
                actions = [parser.OFPActionGroup(group_id=50)]
44
                match = parser.OFPMatch(in_port=3)
45
                self.add_flow(datapath, 10, match, actions)
46
47
48
49
                #add the return flow for h2 in s2.
                # h2 is connected to port 3.
50
                actions = [parser.OFPActionOutput(3)]
51
52
                match = parser.OFPMatch(in_port=1)
                self.add_flow(datapath, 10, match, actions)
53
54
                actions = [parser.OFPActionOutput(3)]
55
                match = parser.OFPMatch(in_port=2)
56
57
                self.add_flow(datapath, 10, match, actions)
59
            # switch s4
60
            if datapath.id == 4 :
61
62
                actions = [parser.OFPActionOutput(2)]
                match = parser.OFPMatch(in_port=1)
63
                self.add_flow(datapath, 10, match, actions)
64
65
                actions = [parser.OFPActionOutput(1)]
66
67
                match = parser.OFPMatch(in_port=2)
                self.add_flow(datapath, 10, match, actions)
68
69
70
            # switch s3
71
72
            if datapath.id == 3:
                actions = [parser.OFPActionOutput(2)]
73
74
                match = parser.OFPMatch(in_port=1)
75
                self.add_flow(datapath, 10, match, actions)
76
77
                actions = [parser.OFPActionOutput(1)]
                match = parser.OFPMatch(in_port=2)
78
79
                self.add_flow(datapath, 10, match, actions)
80
81
        def add_flow(self, datapath, priority, match, actions, buffer_id=None):
82
            ofproto = datapath.ofproto
83
            parser = datapath.ofproto_parser
84
```

```
inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
 85
 86
                                                    actions)]
             if buffer id:
 87
                 mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
 88
                                           priority=priority, match=match,
 89
 90
                                           instructions=inst)
 91
             else:
                 mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
 92
                                          match=match, instructions=inst)
 93
             datapath.send_msg(mod)
 94
 95
         def send_group_mod(self, datapath):
 96
             ofproto = datapath.ofproto
 97
             parser = datapath.ofproto_parser
 98
 99
             LB_WEIGHT1 = 50 #percentage
             LB_WEIGHT2 = 50 #percentage
100
101
             watch_port = ofproto_v1_3.0FPP_ANY
102
             watch_group = ofproto_v1_3.0FPQ_ALL
103
104
             actions1 = [parser.OFPActionOutput(1)]
105
106
             actions2 = [parser.OFPActionOutput(2)]
107
             buckets = [parser.OFPBucket(LB_WEIGHT1, watch_port, watch_group,
     actions=actions1),
108
                         parser.OFPBucket(LB_WEIGHT2, watch_port, watch_group,
     actions=actions2)]
109
              req = parser.OFPGroupMod(datapath, ofproto.OFPGC_ADD,
110
                                       ofproto.OFPGT_SELECT, 50, buckets)
             datapath.send_msg(req)
111
```

Test and Result

mininet> h2 iperf -s &

```
mininet> h1 iperf -c h2 -P 4 -t 30

using sudo ovs-ofctl -0 OpenFlow13 dump-group-stats s1 watch the load

austingutsh@austingutsh-GL502VSK =/#/C/C/L/r/r/app (master)> sudo ovs-ofctl -0 OpenFlow13 dump-group-stats s1

opens (Broup let50, duration=732.090s, ref_count=1,packet_count=7465592,byte_count=346749069788,bucket0:packet_count=3707482,byte_count=17154847354

bucket0: Total 3707482 packets

bucket1: Total 3758110 packets

austingutsh@austingutsh-GL502VSK =//#/C/C/L/r/r/app (master)> sudo ovs-ofctl -0 OpenFlow13 dump-group-stats s2

opens (Broup let50, duration=944,588s, ref_count=1,packet_count=5239935,byte_count=345863226,bucket0:packet_count=2652978,byte_count=175109476,bucket1:packet_count=2830997,byte_count=176753750

bucket0: Total 2652978 packets

bucket1: Total 2585957 packets

using sudo ovs-ofctl -0 OpenFlow13 dump-flows s3 s4 watch the load
```

```
dostingutsingutsingutsin-GL302VSK ~/来/C//L/T/T/App (master) > Sudo 0V3-Dittt - Upenrtuwi3 dump-rtuws = Cookie=0x0, duration=834.660s, table=0, n_packets=3758142/, n_bytes=175200601364, priority=10,in_port="s3-eth1" actions=output:"s3-eth1" cookie=0x0, duration=834.660s, table=0, n_packets=2653011, n_bytes=175114686, priority=10,in_port="s3-eth2" actions=output:"s3-eth1" austinguish@austinguish-GL502VSK ~/桌/C/C/l/r/r/app (master)> sudo 0vs-ofctl - 0 0penFlow13 dump-flows s4 cookie=0x0, duration=842.645s, table=0, n_packets=3707514, n_bytes=171548478672, priority=10,in_port="s4-eth1" actions=output:"s4-eth2" cookie=0x0, duration=842.645s, table=0, n_packets=2586990, n_bytes=170758960, priority=10,in_port="s4-eth2" actions=output:"s4-eth1"
```

Fast failover

Requirements

Write a RYU controller that uses the first path (h1-s1-s3-s2-h2) for routing packets from h1 to h2 and uses the second path for backup. Specifically, when the first path experiences a link failure, the network should automatically switch to the second path without causing packet drop. (hint: consider using OFPGT_FF (FF is short for "fast failover") to construct a group table)

Idea

- 1. To switch 1 and switch2 all packets from port 3 are forwarded to group table 50
- 2. Set OFPGT_FF group mode to monitor the port 1 and port 2.
- 3. Auto switch the port.

Code

```
1 from ryu.base import app_manager
 2 from ryu.controller import ofp_event
 3 from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
   from ryu.controller.handler import set_ev_cls
   from ryu.ofproto import ofproto_v1_3
 6 from ryu.lib.packet import packet
    from ryu.lib.packet import ethernet
    from ryu.lib.packet import ether_types
 8
 9
10
    class SimpleSwitch13(app_manager.RyuApp):
11
        OFP_VERSIONS = [ofproto_v1_3.0FP_VERSION]
12
        def __init__(self, *args, **kwargs):
13
            super(SimpleSwitch13, self).__init__(*args, **kwargs)
14
15
            self.mac_to_port = {}
16
        @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
17
        def switch_features_handler(self, ev):
18
            datapath = ev.msg.datapath
19
            ofproto = datapath.ofproto
20
21
            parser = datapath.ofproto_parser
22
            if datapath.id == 1:
23
                # add group tables
                self.send_group_mod(datapath)
24
                actions = [parser.OFPActionGroup(group_id=50)]
25
                match = parser.OFPMatch(in_port=3)
26
27
                self.add_flow(datapath, 10, match, actions)
28
                #add the return flow for h1 in s1.
29
                # h1 is connected to port 3.
30
                actions = [parser.OFPActionOutput(3)]
31
                match = parser.OFPMatch(in_port=1)
32
                self.add_flow(datapath, 10, match, actions)
33
34
35
                actions = [parser.OFPActionOutput(3)]
                match = parser.OFPMatch(in_port=2)
36
37
                self.add_flow(datapath, 10, match, actions)
39
40
            # switch s2
```

```
41
             if datapath.id == 2:
42
            # add group tables
                 self.send_group_mod(datapath)
43
44
                actions = [parser.OFPActionGroup(group_id=50)]
                match = parser.OFPMatch(in_port=3)
45
                 self.add_flow(datapath, 10, match, actions)
46
47
48
                #add the return flow for h2 in s2.
49
                # h2 is connected to port 3.
50
                actions = [parser.OFPActionOutput(3)]
51
                match = parser.OFPMatch(in_port=1)
52
                self.add_flow(datapath, 10, match, actions)
53
54
                actions = [parser.OFPActionOutput(3)]
55
                match = parser.OFPMatch(in_port=2)
56
57
                 self.add_flow(datapath, 10, match, actions)
58
59
            # switch s4
60
            if datapath.id == 4:
61
                 actions = [parser.OFPActionOutput(2)]
63
                match = parser.OFPMatch(in_port=1)
                 self.add_flow(datapath, 10, match, actions)
64
65
66
                actions = [parser.OFPActionOutput(1)]
                match = parser.OFPMatch(in_port=2)
68
                 self.add_flow(datapath, 10, match, actions)
69
70
            # switch s3
71
            if datapath.id == 3:
73
                actions = [parser.OFPActionOutput(2)]
                match = parser.OFPMatch(in_port=1)
74
                self.add_flow(datapath, 10, match, actions)
75
76
                actions = [parser.OFPActionOutput(1)]
77
                match = parser.OFPMatch(in_port= 2)
79
                 self.add_flow(datapath, 10, match, actions)
80
81
        def add_flow(self, datapath, priority, match, actions, buffer_id=None):
            ofproto = datapath.ofproto
82
            parser = datapath.ofproto_parser
83
84
            inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
85
86
                                                   actions)]
            if buffer_id:
87
88
                mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                          priority=priority, match=match,
89
90
                                          instructions=inst)
91
            else:
                mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
92
93
                                         match=match, instructions=inst)
            datapath.send_msg(mod)
94
95
96
        def send_group_mod(self, datapath):
            ofproto = datapath.ofproto
97
98
            parser = datapath.ofproto_parser
```

```
actions1 = [parser.0FPActionOutput(1)]
actions2 = [parser.0FPActionOutput(2)]
buckets = [parser.0FPBucket(watch_port=1,actions=actions1),
parser.0FPBucket(watch_port=2,actions=actions2)]
req = parser.0FPGroupMod(datapath, ofproto.0FPGC_ADD,
ofproto.0FPGT_FF, 50, buckets)
datapath.send_msg(req)
```

Test and Result

```
use link s1 s3 down link s2 s3 down to close the connection
```

```
64 bytes from 10.0.0.2; icmp_seq=27 ttl=64 time=0.087 ms
64 bytes from 10.0.0.2: icmp_seq=28 ttl=64 time=0.094 ms
64 bytes from 10.0.0.2: icmp_seq=29 ttl=64 time=0.089 ms
64 bytes from 10.0.0.2: icmp_seq=30 ttl=64 time=0.092 ms
64 bytes from 10.0.0.2: icmp_seq=31 ttl=64 time=0.055 ms
64 bytes from 10.0.0.2: icmp_seq=32 ttl=64 time=0.106 ms
64 bytes from 10.0.0.2: icmp_seq=33 ttl=64 time=0.085 ms
64 bytes from 10.0.0.2: icmp_seq=34 ttl=64 time=0.057 ms
64 bytes from 10.0.0.2: icmp_seq=35 ttl=64 time=0.091 ms
64 bytes from 10.0.0.2: icmp_seq=36 ttl=64 time=0.054 ms
64 bytes from 10.0.0.2: icmp_seq=37 ttl=64 time=0.062 ms
64 bytes from 10.0.0.2: icmp_seq=38 ttl=64 time=0.050 ms
64 bytes from 10.0.0.2: icmp_seq=39 ttl=64 time=0.063 ms
64 bytes from 10.0.0.2: icmp_seq=40 ttl=64 time=0.103 ms
64 bytes from 10.0.0.2: icmp_seq=41 ttl=64 time=0.091 ms
64 bytes from 10.0.0.2: icmp_seq=45 ttl=64 time=0.441 ms
64 bytes from 10.0.0.2: icmp_seq=46 ttl=64 time=0.098 ms
64 bytes from 10.0.0.2: icmp_seq=47 ttl=64 time=0.105 ms
64 bytes from 10.0.0.2: icmp_seq=48 ttl=64 time=0.094 ms
64 bytes from 10.0.0.2: icmp_seq=49 ttl=64 time=0.093 ms
64 bytes from 10.0.0.2: icmp_seq=50 ttl=64 time=0.091 ms
64 bytes from 10.0.0.2: icmp_seq=51 ttl=64 time=0.089 ms
64 bytes from 10.0.0.2: icmp_seq=52 ttl=64 time=0.089 ms
64 bytes from 10.0.0.2: icmp_seq=53 ttl=64 time=0.045 ms
64 bytes from 10.0.0.2: icmp_seq=54 ttl=64 time=0.085 ms
64 bytes from 10.0.0.2: icmp_seq=55 ttl=64 time=0.091 ms
64 bytes from 10.0.0.2: icmp_seq=56 ttl=64 time=0.090 ms
64 bytes from 10.0.0.2: icmp_seq=57 ttl=64 time=0.090 ms
64 bytes from 10.0.0.2: icmp_seq=58 ttl=64 time=0.086 ms
```

There are some packet loss between the icmp_seq42-44. Because when the connection closed between s1 and s3 the s2 didn't know the path is loss, it still forwards packet to S3. The solution is to set a backup link between the S3 and S4, set fast failover table in S3 and S4.

Solution

new topo

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
```

```
5 | from ryu.ofproto import ofproto_v1_3
 6
    from ryu.lib.packet import packet
    from ryu.lib.packet import ethernet
7
8
   from ryu.lib.packet import ether_types
9
10
    class SimpleSwitch13(app_manager.RyuApp):
        OFP_VERSIONS = [ofproto_v1_3.0FP_VERSION]
11
12
        def __init__(self, *args, **kwargs):
13
14
            super(SimpleSwitch13, self).__init__(*args, **kwargs)
15
            self.mac_to_port = {}
16
        @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
17
        def switch_features_handler(self, ev):
18
            datapath = ev.msg.datapath
19
            ofproto = datapath.ofproto
20
21
            parser = datapath.ofproto_parser
            if datapath.id == 1:
22
                # add group tables
23
                self.send_group_mod(datapath)
24
                actions = [parser.OFPActionGroup(group_id=50)]
25
                match = parser.OFPMatch(in_port=3)
26
27
                self.add_flow(datapath, 10, match, actions)
28
                #add the return flow for h1 in s1.
29
                # h1 is connected to port 3.
30
                actions = [parser.OFPActionOutput(3)]
31
32
                match = parser.OFPMatch(in_port=1)
                self.add_flow(datapath, 10, match, actions)
33
34
35
                actions = [parser.OFPActionOutput(3)]
                match = parser.OFPMatch(in_port=2)
36
37
                self.add_flow(datapath, 10, match, actions)
            # switch s2
40
            if datapath.id == 2:
41
            # add group tables
42
43
                self.send_group_mod(datapath)
                actions = [parser.OFPActionGroup(group_id=50)]
44
45
                match = parser.OFPMatch(in_port=3)
                self.add_flow(datapath, 10, match, actions)
46
47
48
                #add the return flow for h2 in s2.
49
50
                # h2 is connected to port 3.
                actions = [parser.OFPActionOutput(3)]
51
52
                match = parser.OFPMatch(in_port=1)
                self.add_flow(datapath, 10, match, actions)
53
54
55
                actions = [parser.OFPActionOutput(3)]
                match = parser.OFPMatch(in_port=2)
56
57
                self.add_flow(datapath, 10, match, actions)
59
60
            # switch s4
61
            if datapath.id == 4 :
62
                actions = [parser.OFPActionOutput(2)]
```

```
63
                  match = parser.OFPMatch(in_port=1)
 64
                  self.add_flow(datapath, 10, match, actions)
 65
 66
                 actions = [parser.OFPActionOutput(1)]
 67
                 match = parser.OFPMatch(in_port=2)
                  self.add_flow(datapath, 10, match, actions)
 68
                 actions = [parser.OFPActionOutput(1)]
 69
                 match = parser.OFPMatch(in_port=3)
 70
                  self.add_flow(datapath, 10, match, actions)
 71
 72
 73
 74
             # switch s3
             if datapath.id == 3:
 75
                 actions = [parser.OFPActionOutput(2)]
 76
                 match = parser.OFPMatch(in_port=1)
 77
                  self.add_flow(datapath, 10, match, actions)
 78
 79
                  self.send_s3group_mod(datapath)
                 actions = [parser.OFPActionGroup(group_id=51)]
 81
                 match = parser.OFPMatch(in_port=2)
 82
 83
                  self.add_flow(datapath, 10, match, actions)
 84
 85
         def add_flow(self, datapath, priority, match, actions, buffer_id=None):
 86
             ofproto = datapath.ofproto
 87
             parser = datapath.ofproto_parser
 88
 89
 90
             inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                                    actions)]
 91
             if buffer_id:
 92
                 mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
 93
                                           priority=priority, match=match,
 94
 95
                                           instructions=inst)
             else:
                 mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
 97
 98
                                          match=match, instructions=inst)
              datapath.send_msg(mod)
 99
100
         def send_group_mod(self, datapath):
101
             ofproto = datapath.ofproto
102
103
             parser = datapath.ofproto_parser
             actions1 = [parser.OFPActionOutput(1)]
104
             actions2 = [parser.OFPActionOutput(2)]
105
106
             buckets = [parser.OFPBucket(watch_port=1, actions=actions1),
                         parser.OFPBucket(watch_port=2, actions=actions2)]
107
108
             req = parser.OFPGroupMod(datapath, ofproto.OFPGC_ADD,
                                       ofproto.OFPGT_FF, 50, buckets)
109
110
             datapath.send_msg(req)
111
         def send_s3group_mod(self, datapath):
112
113
             ofproto = datapath.ofproto
             parser = datapath.ofproto_parser
114
115
             actions1 = [parser.OFPActionOutput(1)]
             actions2 = [parser.OFPActionOutput(3)]
116
117
             buckets = [parser.OFPBucket(watch_port=1, actions=actions1),
                         parser.OFPBucket(watch_port=3, actions=actions2)]
118
              req = parser.OFPGroupMod(datapath, ofproto.OFPGC_ADD,
119
120
                                       ofproto.OFPGT_FF, 51, buckets)
```

I never set the weight in the topo, so the default path h1 to h2 is h1-s1-s4-s2-h2.

The back route is h2-s2-s3-s1-h1

When I havn't cut the link between s1 s3 the result.

```
OFPST_GROUP reply (OF1.3) (xid=0x6):
group_id=51,duration=238.773s,ref_count=1,packet_count=94,byte_count=10798,bucket0:packet_count=94,byte_count=107
98,bucket1:packet_count=0,byte_count=0
austinguish@austinguish=GL502VSK ~> sudo ovs-ofctl -0 OpenFlow13 dump-group-stats s3
OFPST_GROUP reply (OF1.3) (xid=0x6):
group_id=51,duration=240.722s,ref_count=1,packet_count=96,byte_count=10994,bucket0:packet_count=96,byte_count=109
94,bucket1:packet_count=0,byte_count=0
austinguish@austinguish=GL502VSK ~> sudo ovs-ofctl -0 OpenFlow13 dump-group-stats s3
OFPST_GROUP reply (OF1.3) (xid=0x6):
group_id=51,duration=255.665s,ref_count=1,packet_count=111,byte_count=12436,bucket0:packet_count=111,byte_count=12436,bucket0:packet_count=111,byte_count=12436,bucket0:packet_count=111,byte_count=12436,bucket0:packet_count=111,byte_count=12436,bucket0:packet_count=111,byte_count=12436,bucket0:packet_count=111,byte_count=12436,bucket0:packet_count=111,byte_count=12436,bucket0:packet_count=111,byte_count=12436,bucket0:packet_count=111,byte_count=12436,bucket0:packet_count=111,byte_count=12436,bucket0:packet_count=111,byte_count=12436,bucket0:packet_count=111,byte_count=12436,bucket0:packet_count=111,byte_count=12436,bucket0:packet_count=111,byte_count=12436,bucket0:packet_count=111,byte_count=12436,bucket0:packet_count=111,byte_count=12436,bucket0:packet_count=111,byte_count=12436,bucket0:packet_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111,byte_count=111
```

The bucket 1 is never used

After I cut the link between s1 s3 the result:

```
sustinguish@austinguish-GL502VSK ~> sudo ovs-ofctl -0 OpenFlow13 dump-group-stats s3
DFPST_GROUP reply (OF1.3) (xid=0x6):
group_id=51,duration=375.946s,ref_count=1,packet_count=238,byte_count=24471,bucket0:packet_count=224,byte_count=2
099,bucket1:packet_count=14,byte_count=1372
sustinguish@austinguish-GL502VSK ~> sudo ovs-ofctl -0 OpenFlow13 dump-group-stats s3
DFPST_GROUP reply (OF1.3) (xid=0x6):
group_id=51,duration=376.924s,ref_count=1,packet_count=239,byte_count=24569,bucket0:packet_count=224,byte_count=2
099,bucket1:packet_count=15,byte_count=1470
```

And there is no packet loss:

```
64 bytes from 10.0.0.2; icmp_seq=214 ttl=64 time=0.091 ms
64 bytes from 10.0.0.2; icmp_seq=215 ttl=64 time=0.051 ms
64 bytes from 10.0.0.2; icmp_seq=216 ttl=64 time=0.108 ms
64 bytes from 10.0.0.2; icmp_seq=217 ttl=64 time=0.099 ms
64 bytes from 10.0.0.2; icmp_seq=218 ttl=64 time=0.040 ms
64 bytes from 10.0.0.2; icmp_seq=218 ttl=64 time=0.100 ms
64 bytes from 10.0.0.2; icmp_seq=220 ttl=64 time=0.049 ms
64 bytes from 10.0.0.2; icmp_seq=220 ttl=64 time=0.086 ms
64 bytes from 10.0.0.2; icmp_seq=221 ttl=64 time=0.096 ms
64 bytes from 10.0.0.2; icmp_seq=222 ttl=64 time=0.096 ms
64 bytes from 10.0.0.2; icmp_seq=223 ttl=64 time=0.092 ms
65 bytes from 10.0.0.2; icmp_seq=223 ttl=64 time=0.092 ms
66 to 10.0.0.2 ping statistics ---
67 223 packets transmitted, 223 received, 0% packet loss, time 227243ms
67 rtt min/avg/max/mdev = 0.036/0.093/1.670/0.110 ms
```