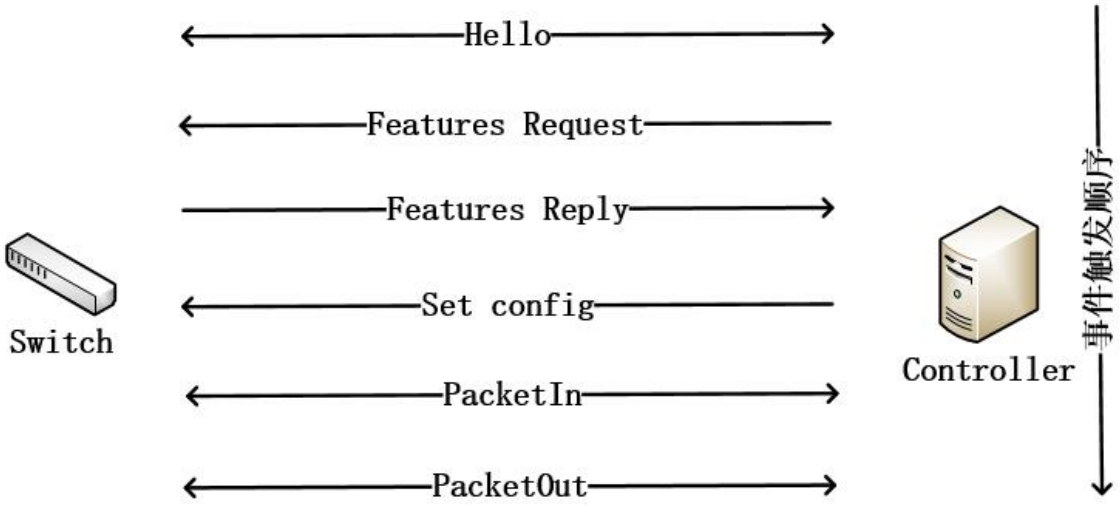
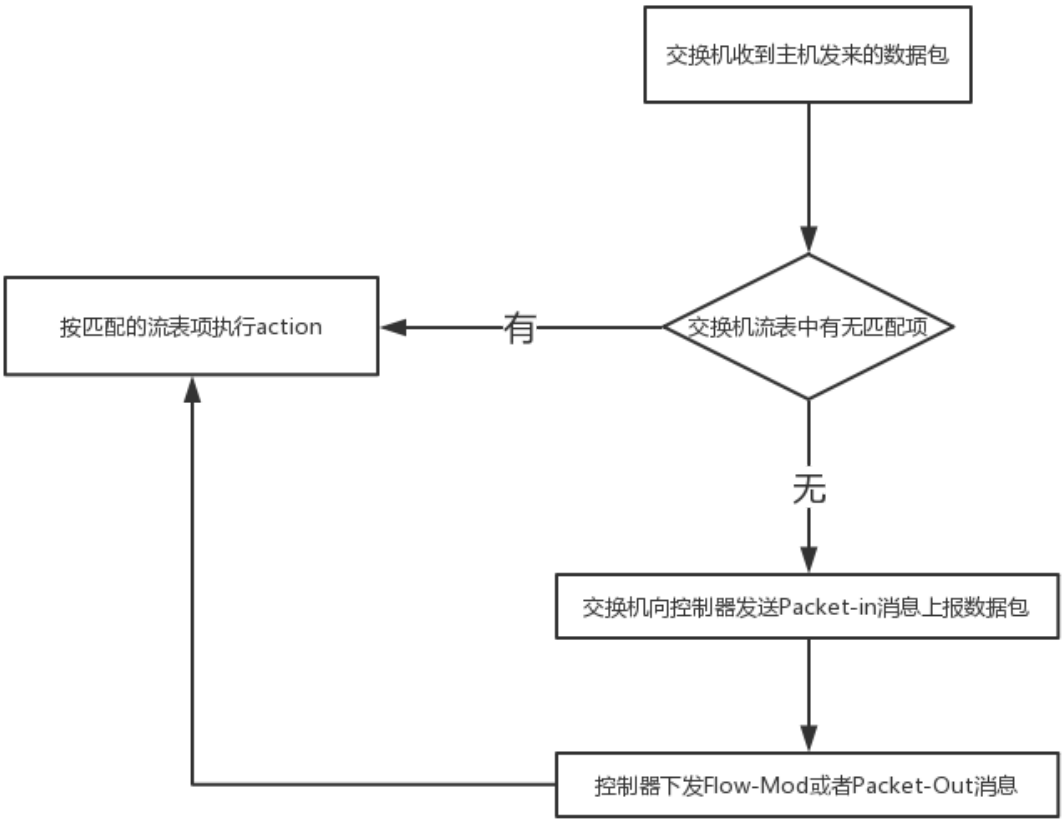
Lab: RYU Programming

1. The Step of Openflow switch interacting with controller



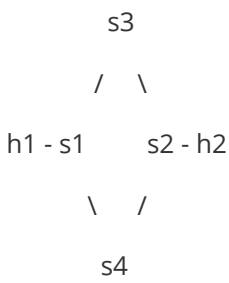


Assignments

Topo

Requirements

Set up the following network first:




```python
#!/usr/bin/python
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.log import setLogLevel
from mininet.cli import CLI
from mininet.node import OVSSwitch, Controller, RemoteController
from time import sleep


class SingleSwitchTopo(Topo):
    "Single switch connected to n hosts."
    def build(self):
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')
        s4 = self.addSwitch('s4')

        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        self.addLink(s1,s4,1,1)
        self.addLink(s1,s3,2,1)
        self.addLink(s4,s2,2,2)
        self.addLink(s3,s2,2,1)
        self.addLink(s1,h1,3,1)
        self.addLink(s2,h2,3,1)

if __name__ == '__main__':
    setLogLevel('info')
    topo = SingleSwitchTopo()
    c1 = RemoteController('c1', ip='127.0.0.1')
    net = Mininet(topo=topo, controller=c1)
    net.start()
    CLI(net)
    net.stop()
```

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

```
#http://docs.openvswitch.org/en/latest/faq/openflow/
#https://mail.openvswitch.org/pipermail/ovs-discuss/2016-August/042394.html
#https://stackoverflow.com/questions/36949861/group-table-issue-openflow-mininet

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
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types
s1op = 1
s2op = 1

class SimpleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
    def add_flow_tmp(self, datapath, priority, match, actions, buffer_id=None):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]
        if buffer_id:
            mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,hard_timeout=5,

 flags=ofproto.OFPFF_SEND_FLOW_REM,priority=priority, match=match,
                                    instructions=inst)
        else:
            mod = parser.OFPFlowMod(datapath=datapath, priority=priority,hard_timeout=5,

 flags=ofproto.OFPFF_SEND_FLOW_REM,match=match, instructions=inst)
```

```
        datapath.send_msg(mod)

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        global s1op
        global s2op
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # install table-miss flow entry
        '''
        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                          ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions)
        '''
        # switch s1
        if datapath.id == 1:

            #add the return flow for h1 in s1.
            # h1 is connected to port 3.
            actions = [parser.OFPActionOutput(s1op)]
            match = parser.OFPMatch(in_port=3)
            self.add_flow_tmp(datapath, 10, match, actions)
            op = 2
            actions = [parser.OFPActionOutput(3)]
            match = parser.OFPMatch(in_port=1)
            self.add_flow(datapath, 10, match, actions)

            actions = [parser.OFPActionOutput(3)]
            match = parser.OFPMatch(in_port=2)
            self.add_flow(datapath, 10, match, actions)


        # switch s2
        if datapath.id == 2:
            actions = [parser.OFPActionOutput(s2op)]
            match = parser.OFPMatch(in_port=3)
            self.add_flow_tmp(datapath, 10, match, actions)
            op = 2
            #add the return flow for h2 in s4.
            # h2 is connected to port 3.
            actions = [parser.OFPActionOutput(3)]
            match = parser.OFPMatch(in_port=1)
            self.add_flow(datapath, 10, match, actions)

            actions = [parser.OFPActionOutput(3)]
            match = parser.OFPMatch(in_port=2)
            self.add_flow(datapath, 10, match, actions)


        # switch s4
        if datapath.id == 4:

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

```
            return

        self.mac_to_port.setdefault(dpid, {})

        self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

        if(dpid==1 or dpid == 4):
            if(op==1):
                out_port=2
                op = 2
            else:
                out_port = 1
                op = 1
        actions = [parser.OFPActionOutput(out_port)]

        # install a flow to avoid packet_in next time
        if out_port != ofproto.OFPP_FLOOD:
            match = parser.OFPMatch(in_port=in_port, eth_dst=dst,
eth_src=src)
            # verify if we have a valid buffer_id, if yes avoid to send both
            # flow_mod & packet_out
            if msg.buffer_id != ofproto.OFP_NO_BUFFER:
                self.add_flow_tmp(datapath, 1, match, actions,
msg.buffer_id)
                return
            else:
                self.add_flow_tmp(datapath, 1, match, actions)
        data = None
        if msg.buffer_id == ofproto.OFP_NO_BUFFER:
            data = msg.data

        out = parser.OFPPacketOut(datapath=datapath,
buffer_id=msg.buffer_id,
                                  in_port=in_port, actions=actions,
data=data)
        datapath.send_msg(out)

    @set_ev_cls(ofp_event.EventOFPFlowRemoved, MAIN_DISPATCHER)
    def _flow_removed_handler(self, ev):
        global s1op
        global s2op
        msg = ev.msg
        dp = msg.datapath
        ofp = dp.ofproto
        parser = dp.ofproto_parser
        if msg.reason == ofp.OFPRR_IDLE_TIMEOUT:
            reason = 'IDLE TIMEOUT'
        elif msg.reason == ofp.OFPRR_HARD_TIMEOUT:
                reason = 'HARD TIMEOUT'
        elif msg.reason == ofp.OFPRR_DELETE:
            reason = 'DELETE'
        elif msg.reason == ofp.OFPRR_GROUP_DELETE:
            reason = 'GROUP DELETE'
        else:
            reason = 'unknown'

        self.logger.debug('OFPFlowRemoved received: ''datapath id=%s'
                          'cookie=%d priority=%d reason=%s table_id=%d '
```

```
                                'duration_sec=%d duration_nsec=%d '
                                'idle_timeout=%d hard_timeout=%d '
                                'packet_count=%d byte_count=%d
match.fields=%s',
                                dp.id,msg.cookie, msg.priority, reason,
msg.table_id,
                                msg.duration_sec, msg.duration_nsec,
                                msg.idle_timeout, msg.hard_timeout,
                                msg.packet_count, msg.byte_count, msg.match)
        if(dp.id==1):
            if(s1op==1):
                s1op = 2
            else:
                s1op = 1
        print("s1op is "+str(s1op))
        actions = [parser.OFPActionOutput(s1op)]
        match = parser.OFPMatch(in_port=3)
        self.add_flow_tmp(dp, 10, match, actions)
```

Test and Result

1. start the controller

   `ryu-manager --verbose per5s.py`

2. start net topo and connect to the controller

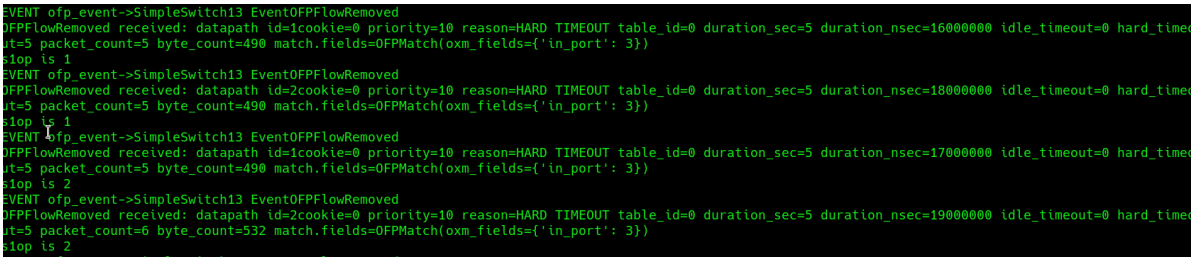
   `sudo python topo.py --controller remote`

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

Mininet Result

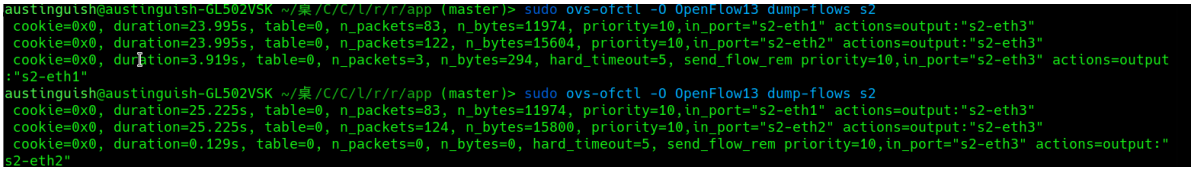
```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.991 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.088 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.089 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.087 ms
64 bytes 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
```

RYU Controller Result:



It show the event triggered the handler and made echo.

Flow table:



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

```python
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
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types

class SimpleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}

    @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
        if datapath.id == 1:
            # add group tables
            self.send_group_mod(datapath)
            actions = [parser.OFPActionGroup(group_id=50)]
            match = parser.OFPMatch(in_port=3)
```

```
            self.add_flow(datapath, 10, match, actions)

            #add the return flow for h1 in s1.
            # h1 is connected to port 3.
            actions = [parser.OFPActionOutput(3)]
            match = parser.OFPMatch(in_port=1)
            self.add_flow(datapath, 10, match, actions)

            actions = [parser.OFPActionOutput(3)]
            match = parser.OFPMatch(in_port=2)
            self.add_flow(datapath, 10, match, actions)


        # switch s2
        if datapath.id == 2:
        # add group tables
            self.send_group_mod(datapath)
            actions = [parser.OFPActionGroup(group_id=50)]
            match = parser.OFPMatch(in_port=3)
            self.add_flow(datapath, 10, match, actions)


            #add the return flow for h2 in s2.
            # h2 is connected to port 3.
            actions = [parser.OFPActionOutput(3)]
            match = parser.OFPMatch(in_port=1)
            self.add_flow(datapath, 10, match, actions)

            actions = [parser.OFPActionOutput(3)]
            match = parser.OFPMatch(in_port=2)
            self.add_flow(datapath, 10, match, actions)


        # switch s4
        if datapath.id == 4 :
            actions = [parser.OFPActionOutput(2)]
            match = parser.OFPMatch(in_port=1)
            self.add_flow(datapath, 10, match, actions)

            actions = [parser.OFPActionOutput(1)]
            match = parser.OFPMatch(in_port=2)
            self.add_flow(datapath, 10, match, actions)


        # switch s3
        if datapath.id == 3:
            actions = [parser.OFPActionOutput(2)]
            match = parser.OFPMatch(in_port=1)
            self.add_flow(datapath, 10, match, actions)

            actions = [parser.OFPActionOutput(1)]
            match = parser.OFPMatch(in_port=2)
            self.add_flow(datapath, 10, match, actions)

    def add_flow(self, datapath, priority, match, actions, buffer_id=None):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

```

```
        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]
        if buffer_id:
            mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                    priority=priority, match=match,
                                    instructions=inst)
        else:
            mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                    match=match, instructions=inst)
        datapath.send_msg(mod)

    def send_group_mod(self, datapath):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        LB_WEIGHT1 = 50 #percentage
        LB_WEIGHT2 = 50 #percentage

        watch_port = ofproto_v1_3.OFPP_ANY
        watch_group = ofproto_v1_3.OFPQ_ALL

        actions1 = [parser.OFPActionOutput(1)]
        actions2 = [parser.OFPActionOutput(2)]
        buckets = [parser.OFPBucket(LB_WEIGHT1, watch_port, watch_group,
actions=actions1),
                   parser.OFPBucket(LB_WEIGHT2, watch_port, watch_group,
actions=actions2)]
        req = parser.OFPGroupMod(datapath, ofproto.OFPGC_ADD,
                                 ofproto.OFPGT_SELECT, 50, buckets)
        datapath.send_msg(req)
```

Test and Result

`mininet> h2 iperf -s &`

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

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

```
austinguish@austinguish-GL502VSK ~/桌/C/C/l/r/r/app (master)> sudo ovs-ofctl -O OpenFlow13 dump-group-stats s1
OFPST_GROUP reply (OF1.3) (xid=0x6):
 group_id=50,duration=732.090s,ref_count=1,packet_count=7465592,byte_count=346749069788,bucket0:packet_count=3707482,byte_count=171548473548,bucket1:packet_count=3758110,byte_count=175200596240
```

bucket0: Total 3707482 packets

bucket1: Total 3758110 packets

```
austinguish@austinguish-GL502VSK ~/桌/C/C/l/r/r/app (master)> sudo ovs-ofctl -O OpenFlow13 dump-group-stats s2
OFPST_GROUP reply (OF1.3) (xid=0x6):
 group_id=50,duration=944.588s,ref_count=1,packet_count=5239935,byte_count=345863226,bucket0:packet_count=2652978,byte_count=175109476,bucket1:packet_count=2586957,byte_count=170753750
```

bucket0: Total 2652978 packets

bucket1: Total 2585957 packets

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

```
austinguish@austinguish-GL502VSK ~/桌/C/C/l/r/r/app (master)> sudo ovs-ofctl -O OpenFlow13 dump-flows s3
 cookie=0x0, duration=834.660s, table=0, n_packets=3758142, n_bytes=175200601364, priority=10,in_port="s3-eth1" actions=output:"s3-eth2"
 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 ovs-ofctl -O OpenFlow13 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"
```

The results show the load on 2 paths is balanced

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

```
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
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types

class SimpleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}

    @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
        if datapath.id == 1:
            # add group tables
            self.send_group_mod(datapath)
            actions = [parser.OFPActionGroup(group_id=50)]
            match = parser.OFPMatch(in_port=3)
            self.add_flow(datapath, 10, match, actions)

            #add the return flow for h1 in s1.
            # h1 is connected to port 3.
            actions = [parser.OFPActionOutput(3)]
            match = parser.OFPMatch(in_port=1)
            self.add_flow(datapath, 10, match, actions)

            actions = [parser.OFPActionOutput(3)]
            match = parser.OFPMatch(in_port=2)
            self.add_flow(datapath, 10, match, actions)


        # switch s2
```

```
        if datapath.id == 2:
        # add group tables
            self.send_group_mod(datapath)
            actions = [parser.OFPActionGroup(group_id=50)]
            match = parser.OFPMatch(in_port=3)
            self.add_flow(datapath, 10, match, actions)


            #add the return flow for h2 in s2.
            # h2 is connected to port 3.
            actions = [parser.OFPActionOutput(3)]
            match = parser.OFPMatch(in_port=1)
            self.add_flow(datapath, 10, match, actions)

            actions = [parser.OFPActionOutput(3)]
            match = parser.OFPMatch(in_port=2)
            self.add_flow(datapath, 10, match, actions)


        # switch s4
        if datapath.id == 4 :
            actions = [parser.OFPActionOutput(2)]
            match = parser.OFPMatch(in_port=1)
            self.add_flow(datapath, 10, match, actions)

            actions = [parser.OFPActionOutput(1)]
            match = parser.OFPMatch(in_port=2)
            self.add_flow(datapath, 10, match, actions)


        # switch s3
        if datapath.id == 3:
            actions = [parser.OFPActionOutput(2)]
            match = parser.OFPMatch(in_port=1)
            self.add_flow(datapath, 10, match, actions)

            actions = [parser.OFPActionOutput(1)]
            match = parser.OFPMatch(in_port= 2)
            self.add_flow(datapath, 10, match, actions)

    def add_flow(self, datapath, priority, match, actions, buffer_id=None):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]
        if buffer_id:
            mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                    priority=priority, match=match,
                                    instructions=inst)
        else:
            mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                    match=match, instructions=inst)
        datapath.send_msg(mod)

    def send_group_mod(self, datapath):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
```

```
        actions1 = [parser.OFPActionOutput(1)]
        actions2 = [parser.OFPActionOutput(2)]
        buckets = [parser.OFPBucket(watch_port=1,actions=actions1),
                   parser.OFPBucket(watch_port=2,actions=actions2)]
        req = parser.OFPGroupMod(datapath, ofproto.OFPGC_ADD,
                                 ofproto.OFPGT_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

```
              Switch3
             /    |     \
h1 ---Switch1     |     Switch2-----h2
             \    |     /
              Switch4
```

```
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
```

```
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types

class SimpleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}

    @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
        if datapath.id == 1:
            # add group tables
            self.send_group_mod(datapath)
            actions = [parser.OFPActionGroup(group_id=50)]
            match = parser.OFPMatch(in_port=3)
            self.add_flow(datapath, 10, match, actions)

            #add the return flow for h1 in s1.
            # h1 is connected to port 3.
            actions = [parser.OFPActionOutput(3)]
            match = parser.OFPMatch(in_port=1)
            self.add_flow(datapath, 10, match, actions)

            actions = [parser.OFPActionOutput(3)]
            match = parser.OFPMatch(in_port=2)
            self.add_flow(datapath, 10, match, actions)


        # switch s2
        if datapath.id == 2:
        # add group tables
            self.send_group_mod(datapath)
            actions = [parser.OFPActionGroup(group_id=50)]
            match = parser.OFPMatch(in_port=3)
            self.add_flow(datapath, 10, match, actions)


            #add the return flow for h2 in s2.
            # h2 is connected to port 3.
            actions = [parser.OFPActionOutput(3)]
            match = parser.OFPMatch(in_port=1)
            self.add_flow(datapath, 10, match, actions)

            actions = [parser.OFPActionOutput(3)]
            match = parser.OFPMatch(in_port=2)
            self.add_flow(datapath, 10, match, actions)


        # switch s4
        if datapath.id == 4 :
            actions = [parser.OFPActionOutput(2)]
```

```
            match = parser.OFPMatch(in_port=1)
            self.add_flow(datapath, 10, match, actions)

            actions = [parser.OFPActionOutput(1)]
            match = parser.OFPMatch(in_port=2)
            self.add_flow(datapath, 10, match, actions)
            actions = [parser.OFPActionOutput(1)]
            match = parser.OFPMatch(in_port=3)
            self.add_flow(datapath, 10, match, actions)


        # switch s3
        if datapath.id == 3:
            actions = [parser.OFPActionOutput(2)]
            match = parser.OFPMatch(in_port=1)
            self.add_flow(datapath, 10, match, actions)

            self.send_s3group_mod(datapath)
            actions = [parser.OFPActionGroup(group_id=51)]
            match = parser.OFPMatch(in_port=2)
            self.add_flow(datapath, 10, match, actions)


    def add_flow(self, datapath, priority, match, actions, buffer_id=None):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]
        if buffer_id:
            mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                    priority=priority, match=match,
                                    instructions=inst)
        else:
            mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                    match=match, instructions=inst)
        datapath.send_msg(mod)

    def send_group_mod(self, datapath):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        actions1 = [parser.OFPActionOutput(1)]
        actions2 = [parser.OFPActionOutput(2)]
        buckets = [parser.OFPBucket(watch_port=1,actions=actions1),
                   parser.OFPBucket(watch_port=2,actions=actions2)]
        req = parser.OFPGroupMod(datapath, ofproto.OFPGC_ADD,
                                 ofproto.OFPGT_FF, 50, buckets)
        datapath.send_msg(req)

    def send_s3group_mod(self, datapath):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        actions1 = [parser.OFPActionOutput(1)]
        actions2 = [parser.OFPActionOutput(3)]
        buckets = [parser.OFPBucket(watch_port=1,actions=actions1),
                   parser.OFPBucket(watch_port=3,actions=actions2)]
        req = parser.OFPGroupMod(datapath, ofproto.OFPGC_ADD,
                                 ofproto.OFPGT_FF, 51, buckets)
```

```
121             datapath.send_msg(req)
```

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 -O 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 -O 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=1
2436,bucket1:packet_count=0,byte_count=0
```

The bucket 1 is never used

After I cut the link between `s1` `s3` the result:

```
austinguish@austinguish-GL502VSK ~> sudo ovs-ofctl -O OpenFlow13 dump-group-stats s3
OFPST_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
3099,bucket1:packet_count=14,byte_count=1372
austinguish@austinguish-GL502VSK ~> sudo ovs-ofctl -O OpenFlow13 dump-group-stats s3
OFPST_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
3099,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=219 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=221 ttl=64 time=0.086 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
^C
--- 10.0.0.2 ping statistics ---
223 packets transmitted, 223 received, 0% packet loss, time 227243ms
rtt min/avg/max/mdev = 0.036/0.093/1.670/0.110 ms
```