

Write SDN Controller

计算机网络 CS339

李子龙 518070910095

2021 年 11 月 4 日

目录

1	建立网络	2
2	定时切换	3
3	使用双路	5
4	断路备用	7
A	定时切换代码	8
B	使用双路代码	10

Ryu provides software components with well defined API's that make it easy for developers to create new network management and control applications.

1 建立网络

Set up the following network first:

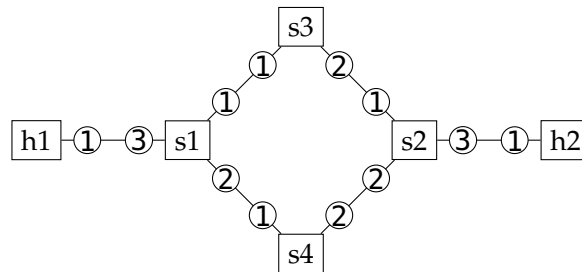


图 1: 网络拓扑

使用给出的示例代码。但是为了处理上的方便，将会指定链路连接的端口号。

Listing 1: [loopnet.py](#)

```
1  #!/usr/bin/python
2  """Sample Code"""
3  from mininet.topo import Topo
4  from mininet.net import Mininet
5  from mininet.node import OVSBridge, OVSSwitch, OVSKernelSwitch
6  from mininet.node import CPULimitedHost
7  from mininet.node import RemoteController
8  from mininet.link import TCLink
9  from mininet.util import dumpNodeConnections
10 from mininet.log import setLogLevel, info
11 from mininet.cli import CLI
12 from sys import argv
13 def Test():
14     "Create network and run simple performance test"
15     net = Mininet( switch=OVSSwitch, host=CPULimitedHost, link=TCLink,
16                   autoStaticArp=False, controller=RemoteController)
17     switch1 = net.addSwitch('s1')
18     switch2 = net.addSwitch('s2')
19     switch3 = net.addSwitch('s3')
20     switch4 = net.addSwitch('s4')
21     host1 = net.addHost('h1', cpu=.25)
22     host2 = net.addHost('h2', cpu=.25)
23     net.addLink(host1, switch1, 1, 3, bw=10, delay='5ms', loss=0, use_htb=
24                 True)
25     net.addLink(host2, switch2, 1, 3, bw=10, delay='5ms', loss=0, use_htb=
26                 True)
27     net.addLink(switch1, switch3, 1, 1, bw=10, delay='5ms', loss=0, use_htb=
28                 True)
29     net.addLink(switch1, switch4, 2, 1, bw=10, delay='5ms', loss=0, use_htb=
30                 True)
```

```

26 net.addLink(switch2, switch3, 1, 2, bw=10, delay='5ms', loss=0, use_htb=
27 net.addLink(switch2, switch4, 2, 2, bw=10, delay='5ms', loss=0, use_htb=
    True)
28 c1 = net.addController('c1', controller=RemoteController, ip="127.0.0.1",
    port=6653)
29 net.build()
30 c1.start()
31 s1, s2, s3, s4 = net.getNodeByName('s1', 's2', 's3', 's4')
32 s1.start([c1])
33 s2.start([c1])
34 s3.start([c1])
35 s4.start([c1])
36 net.start()
37 info( "Dumping host connections\n" )
38 dumpNodeConnections(net.hosts)
39 h1, h2 = net.getNodeByName('h1', 'h2')
40 CLI(net)
41 net.stop()
42 if __name__ == '__main__':
43     # setLogLevel( 'debug' )
44     setLogLevel('info')
45     Test()

```

2 定时切换

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

查看修改流的定义函数。其中参数 `hard_timeout` 用于定义丢弃流前的最大秒数。

Listing 2: `../ryu/ryu/ofproto/ofproto_v1_3_parser.py`

```

2703 def __init__(self, datapath, cookie=0, cookie_mask=0, table_id=0,
2704             command=ofproto.OFPFC_ADD,
2705             idle_timeout=0, hard_timeout=0,
2706             priority=ofproto.OFP_DEFAULT_PRIORITY,
2707             buffer_id=ofproto.OFP_NO_BUFFER,
2708             out_port=0, out_group=0, flags=0,
2709             match=None,
2710             instructions=None):

```

参数 `flags` 可以被指定为 `OFPFF_SEND_FLOW_REM`，可以用于在丢弃流后发出事件用于相关处理。

Flow-Removed: Inform the controller about the removal of a flow entry from a flow table. Flow-Removed messages are only sent for flow entries with the

`OFPFF_SEND_FLOW_REM`

flag set. They are generated as the result of a controller flow delete requests or the switch flow expiry process when one of the flow timeout is exceeded (see 5.5).^[1]

Listing 3: `../ryu/ryu/ofproto/ofproto_v1_3.py`

```
371 OFPFF_SEND_FLOW_REM = 1 << 0      # Send flow removed message when flow
372                                     # expires or is deleted.
```

处理丢弃事件，RYU 源码给出了例子：

Listing 4: `../ryu/ryu/ofproto/ofproto_v1_3_parser.py`

```
2377 @set_ev_cls(ofp_event.EventOFPFlowRemoved, MAIN_DISPATCHER)
2378 def flow_removed_handler(self, ev):
2379     msg = ev.msg
2380     dp = msg.datapath
2381     ofp = dp.ofproto
2382
2383     if msg.reason == ofp.OFPRR_IDLE_TIMEOUT:
2384         reason = 'IDLE TIMEOUT'
2385     elif msg.reason == ofp.OFPRR_HARD_TIMEOUT:
2386         reason = 'HARD TIMEOUT'
2387     elif msg.reason == ofp.OFPRR_DELETE:
2388         reason = 'DELETE'
2389     elif msg.reason == ofp.OFPRR_GROUP_DELETE:
2390         reason = 'GROUP DELETE'
2391     else:
2392         reason = 'unknown'
2393
2394     self.logger.debug('OFPFlowRemoved received: '
2395                      'cookie=%d priority=%d reason=%s table_id=%d '
2396                      'duration_sec=%d duration_nsec=%d '
2397                      'idle_timeout=%d hard_timeout=%d '
2398                      'packet_count=%d byte_count=%d match.fields=%s'
2399                      ',
2400                      msg.cookie, msg.priority, reason, msg.table_id,
2401                      msg.duration_sec, msg.duration_nsec,
2402                      msg.idle_timeout, msg.hard_timeout,
2403                      msg.packet_count, msg.byte_count, msg.match)
```

`datapath.id` 用于识别交换机，`s1`对应1号，`s2`对应2号，依次类推。

A terminal window titled 'root@ubuntu: /VMShared/Networking/lab05' showing the execution of 'ryu-manager task2.py'. The output includes: 'loading app task2.py', 'loading app ryu.controller.ofp_handler', 'instantiating app task2.py of PeriodicSwitch', 'instantiating app ryu.controller.ofp_handler of OFPHandler', and a list of switch configurations: 'CONFIG switch id: 1', 'CONFIG switch id: 2', 'CONFIG switch id: 3', and 'CONFIG switch id: 4'.

图 2: 交换机编号

使用下面的命令可以可视化地观察流信息^[2]，并启动控制器。

```
ryu/ryu/app/gui_topology$ ryu-manager --observe-links  
gui_topology.py ../../../../lab05/task2.py
```

```
root@ubuntu: /VMShared/Networking/lab05  
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=92.0 ms  
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=49.7 ms  
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=47.9 ms  
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=44.7 ms  
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=47.7 ms  
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=45.9 ms  
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=45.2 ms  
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=44.9 ms  
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=45.8 ms  
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=46.2 ms  
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=46.7 ms  
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=46.5 ms  
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=46.0 ms  
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=44.2 ms  
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=48.9 ms  
64 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=43.5 ms  
64 bytes from 10.0.0.2: icmp_seq=17 ttl=64 time=45.0 ms  
64 bytes from 10.0.0.2: icmp_seq=18 ttl=64 time=46.0 ms  
64 bytes from 10.0.0.2: icmp_seq=19 ttl=64 time=45.5 ms  
^C  
--- 10.0.0.2 ping statistics ---  
19 packets transmitted, 19 received, 0% packet loss, time 18026ms  
rtt min/avg/max/mdev = 43.500/48.595/92.015/10.347 ms  
mininet>
```

图 3: 测试连接

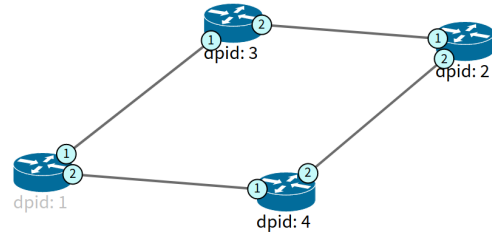


图 4: gui_topology 展示的拓扑结构

由图 4 可见，可以通过定时切换 s1 和 s2 的输出端口，来达到切换链路的功能。切换为 3 → 1 采用上面的链路，切换为 3 → 2 采用下面的链路。由于有两个流会超时，但是临近的两个超时应当只改变一次端口状态，所以会设置一个状态变量用于避免不同步情况的设置延迟导致的丢包，如图 5 所示。在图 3 中可见是能够 ping 通的。相关代码见附录 A。

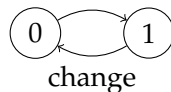


图 5: 状态机

3 使用双路

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

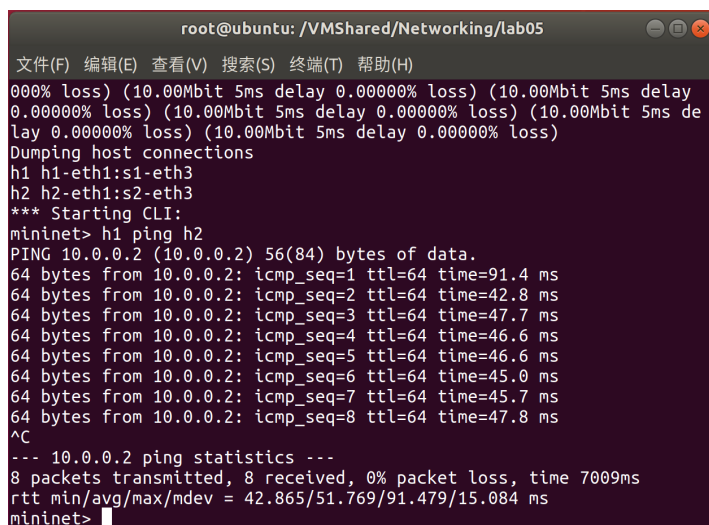
select: Execute one bucket in the group. Packets are processed by a single bucket in the group, based on a switch-computed selection algorithm (e.g. hash on some user-configured tuple or simple round robin). All configuration and state for the selection algorithm is external to OpenFlow. The selection algorithm should implement equal load sharing and can optionally be based on bucket weights. When a port specified in a bucket in a select group goes down, the switch may restrict bucket selection

to the remaining set (those with forwarding actions to live ports) instead of dropping packets destined to that port. This behavior may reduce the disruption of a downed link or switch.^[1]

代码见附录 B，在使用组之前，需要先注册 `OFPGT_SELECT` 组。使用给出的示例代码发送请求组信息。这里 `watch_port` 被设定为 `OFPP_ANY`，而 `watch_group` 被设定为 `OFPQ_ALL`^[3]，定义如下：

Listing 5: `../ryu/ryu/ofproto/ofproto_v1_3.py`

```
109 OFPP_ANY = 0xffffffff          # Not associated with a physical port.
110
111 # All ones is used to indicate all queues in a port (for stats retrieval).
112 OFPQ_ALL = 0xffffffff
```

A terminal window titled 'root@ubuntu: /VMShared/Networking/lab05' showing the output of a network test. The output includes a series of ping results for 10.0.0.2, showing 0% packet loss and a time of 7009ms. The terminal text is as follows:

```
root@ubuntu: /VMShared/Networking/lab05
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
000% loss) (10.00Mbit 5ms delay 0.00000% loss) (10.00Mbit 5ms delay
0.00000% loss) (10.00Mbit 5ms delay 0.00000% loss) (10.00Mbit 5ms de
lay 0.00000% loss) (10.00Mbit 5ms delay 0.00000% loss)
Dumping host connections
h1 h1-eth1:s1-eth3
h2 h2-eth1:s2-eth3
*** 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=91.4 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=42.8 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=47.7 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=46.6 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=46.6 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=45.0 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=45.7 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=47.8 ms
^C
--- 10.0.0.2 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7009ms
rtt min/avg/max/mdev = 42.865/51.769/91.479/15.084 ms
mininet>
```

图 6: 检查可达性

```
root@ubuntu: /VMShared/Networking/lab05
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
mininet> dpctl dump-flows
*** s1 ***
cookie=0x0, duration=115.179s, table=0, n_packets=12, n_bytes=908, priority=10,in_port="s1-eth3" actions=group:1
cookie=0x0, duration=115.179s, table=0, n_packets=51, n_bytes=6076, priority=10,in_port="s1-eth1" actions=output:"s1-eth3"
cookie=0x0, duration=115.179s, table=0, n_packets=56, n_bytes=6310, priority=10,in_port="s1-eth2" actions=output:"s1-eth3"
*** s2 ***
cookie=0x0, duration=115.126s, table=0, n_packets=13, n_bytes=994, priority=10,in_port="s2-eth3" actions=group:1
cookie=0x0, duration=115.126s, table=0, n_packets=56, n_bytes=6382, priority=10,in_port="s2-eth1" actions=output:"s2-eth3"
cookie=0x0, duration=115.126s, table=0, n_packets=51, n_bytes=6004, priority=10,in_port="s2-eth2" actions=output:"s2-eth3"
*** s3 ***
cookie=0x0, duration=115.038s, table=0, n_packets=32, n_bytes=3491, priority=10,in_port="s3-eth1" actions=output:"s3-eth2"
cookie=0x0, duration=115.038s, table=0, n_packets=27, n_bytes=3185, priority=10,in_port="s3-eth2" actions=output:"s3-eth1"
*** s4 ***
cookie=0x0, duration=114.985s, table=0, n_packets=27, n_bytes=3113, priority=10,in_port="s4-eth1" actions=output:"s4-eth2"
cookie=0x0, duration=114.985s, table=0, n_packets=32, n_bytes=3419, priority=10,in_port="s4-eth2" actions=output:"s4-eth1"
mininet>
```

图 7: 流表

图 6 显示了其可达性正常。图 7 使用

```
dpctl dump-flows
```

显示流表信息，可以看到 s1 和 s2 对应的端口 1 和 2 负载是均衡的，说明了功能的正常实现。

4 断路备用

Write an 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 `0FPGT_FF` (FF is short for “fast failover”) to construct a group table)

fast failover: Execute the first live bucket. Each action bucket is associated with a specific port and/or group that controls its liveness. The buckets are evaluated in the order defined by the group, and the first bucket which is associated with a live port/group is selected. This group type enables the switch to change forwarding without requiring a round trip to the controller. If no buckets are live, packets are dropped. This group type must implement a *liveness mechanism*(see 6.5).^[1]

参考文献

- [1] Open Networking Foundation. OpenFlow switch specification[M/OL]. 2012. <https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf>.

- [2] 梵高的向日葵丶. SDN(三) RYU控制器相关笔记[EB/OL]. 2020. https://blog.csdn.net/weixin_42094589/article/details/104160571.
- [3] 李呈. 基于RYU应用开发之负载均衡[EB/OL]. 2015. <https://www.sdnlab.com/10211.html>.

A 定时切换代码

Listing 6: [task2.py](#)

```
1  # 2. Write an RYU controller that switches paths (h1-s1-s3-s2-h2 or h1-s1-s4-
   s2-h2) between h1 and h2 every 5 seconds.
2
3  from ryu.base import app_manager
4  from ryu.controller import ofp_event
5  from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER,
   set_ev_cls
6  from ryu.lib import packet
7  from ryu.lib.packet import ether_types, ethernet
8  from ryu.lib.packet import in_proto as inet
9  from ryu.ofproto import ofproto_v1_3
10
11  pathport = 1
12  pathstate = 1
13  # 0 -> 1 -> 0 (change)
14
15  class PeriodicSwitich(app_manager.RyuApp):
16      OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
17
18      def __init__(self, *_args, **_kwargs):
19          super(PeriodicSwitich, self).__init__(_args, **_kwargs)
20
21      @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
22      def switch_features_handler(self, ev):
23          datapath = ev.msg.datapath
24          ofproto = datapath.ofproto
25          parser = datapath.ofproto_parser
26          out_port = 1
27
28          match = parser.OFPMatch(in_port=1, eth_type=ether_types.ETH_TYPE_IP,
   ipv4_src='10.0.0.1', ipv4_dst='10.0.0.2', ip_proto=inet.IPPROTO_UDP,
   udp_dst=5555)
29          actions = [parser.OFPActionOutput(out_port)]
30          self.add_flow(datapath, 0, match, actions)
31
32      def add_flow(self, datapath, priority, match, actions, buffer_id=None):
33          """
34          Default adding flow.
35          """
36          ofproto = datapath.ofproto
37          parser = datapath.ofproto_parser
38
39          inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
```



```

40             actions)]
41         if buffer_id:
42             mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
43                                     priority=priority, match=match,
44                                     instructions=inst)
45         else:
46             mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
47                                     match=match, instructions=inst)
48         datapath.send_msg(mod)
49
50     def add_flow_timeout(self, datapath, priority, match, actions, buffer_id=
51     None):
52         """
53         Add a flow that timeout in 5 sec.
54         """
55         ofproto = datapath.ofproto
56         parser = datapath.ofproto_parser
57         inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
58                                             actions)]
59         if buffer_id:
60             mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
61                                     priority=priority, match=match,
62                                     instructions=inst, hard_timeout=5, flags=
63         ofproto.OFPFF_SEND_FLOW_REM)
64         else:
65             mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
66                                     match=match, instructions=inst,
67                                     hard_timeout=5, flags=ofproto.OFPFF_SEND_FLOW_REM)
68         datapath.send_msg(mod)
69
70     @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
71     def switch_features_handler(self, ev):
72         global pathport
73
74         datapath = ev.msg.datapath
75         ofproto = datapath.ofproto
76         parser = datapath.ofproto_parser
77
78         # Since the switches are added in order,
79         # The id is appended in order as well.
80         print('CONFIG switch id: ' + str(datapath.id))
81
82         if datapath.id == 1 or datapath.id == 2:
83             # forward flow h1 -> s1(s2)
84             # input from port 3, output to the selected port.
85             match = parser.OFPMatch(in_port=3)
86             actions = [parser.OFPActionOutput(pathport)] #
87             self.add_flow_timeout(datapath, 2, match, actions)
88
89             # return flow s1(s2) -> h1
90             # 2 possible flows: from port 1, from port 2.
91             match = parser.OFPMatch(in_port=1)
92             actions = [parser.OFPActionOutput(3)]
93             self.add_flow(datapath, 2, match, actions)
94             match = parser.OFPMatch(in_port=2)

```

```

93         actions = [parser.OFPActionOutput(3)]
94         self.add_flow(datapath, 2, match, actions)
95     elif datapath.id == 3 or datapath.id == 4:
96         # s3 / s4
97         match = parser.OFPMatch(in_port=1)
98         actions = [parser.OFPActionOutput(2)]
99         self.add_flow(datapath, 2, match, actions)
100        match = parser.OFPMatch(in_port=2)
101        actions = [parser.OFPActionOutput(1)]
102        self.add_flow(datapath, 2, match, actions)
103
104    @set_ev_cls(ofp_event.EventOFPFlowRemoved, MAIN_DISPATCHER)
105    def flow_removed_handler(self, ev):
106        global pathport, pathstate
107
108        msg = ev.msg
109        datapath = msg.datapath
110        ofproto = datapath.ofproto
111        parser = datapath.ofproto_parser
112
113        if msg.reason == ofproto.OFPRR_HARD_TIMEOUT:
114            pathstate += 1
115            if pathstate == 2:
116                pathstate = 0
117                pathport = 2 if pathport==1 else 1
118                # change on pathport could only be invoked once in one round.
119                print('Switich to port: ' + str(pathport))
120                print('OFPFlowRemoved received: ' + str(datapath.id))
121                match = parser.OFPMatch(in_port=3)
122                actions = [parser.OFPActionOutput(pathport)]
123                self.add_flow_timeout(datapath, 2, match, actions)

```

B 使用双路代码

Listing 7: [task3.py](#)

```

1  # 3. Write an RYU controller that uses both paths to forward packets from h1
   to h2.
2
3  from ryu.base import app_manager
4  from ryu.controller import ofp_event
5  from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER,
   set_ev_cls
6  from ryu.lib import packet
7  from ryu.lib.packet import ether_types, ethernet
8  from ryu.lib.packet import in_proto as inet
9  from ryu.ofproto import ofproto_v1_3
10
11  class PeriodicSwitich(app_manager.RyuApp):
12      OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
13
14      def __init__(self, *_args, **_kwargs):
15          super(PeriodicSwitich, self).__init__(_args, **_kwargs)

```

```

16
17 @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
18 def switch_features_handler(self, ev):
19     datapath = ev.msg.datapath
20     ofproto = datapath.ofproto
21     parser = datapath.ofproto_parser
22
23     if datapath.id == 1 or datapath.id == 2:
24         # s1 / s2
25         # bucket group
26         actions1 = [parser.OFPActionOutput(1)]
27         actions2 = [parser.OFPActionOutput(2)]
28         weight1 = 50
29         weight2 = 50
30         watch_port = ofproto_v1_3.OFPP_ANY
31         watch_group = ofproto_v1_3.OFPQ_ALL
32         buckets = [
33             parser.OFPBucket(weight1, watch_port, watch_group, actions1),
34             parser.OFPBucket(weight2, watch_port, watch_group, actions2)]
35
36         group_id = 1
37         req = parser.OFPGroupMod(datapath, ofproto.OFPGC_ADD, ofproto.
OFPGT_SELECT, group_id, buckets)
38         datapath.send_msg(req)
39
40         match = parser.OFPMatch(in_port=3)
41         self.add_flow_group(datapath, 10, match, group_id)
42
43         # return flow s1(s2) -> h1
44         # 2 possible flows: from port 1, from port 2.
45         match = parser.OFPMatch(in_port=1)
46         actions = [parser.OFPActionOutput(3)]
47         self.add_flow(datapath, 10, match, actions)
48         match = parser.OFPMatch(in_port=2)
49         actions = [parser.OFPActionOutput(3)]
50         self.add_flow(datapath, 10, match, actions)
51     elif datapath.id == 3 or datapath.id == 4:
52         # s3 / s4
53         match = parser.OFPMatch(in_port=1)
54         actions = [parser.OFPActionOutput(2)]
55         self.add_flow(datapath, 10, match, actions)
56         match = parser.OFPMatch(in_port=2)
57         actions = [parser.OFPActionOutput(1)]
58         self.add_flow(datapath, 10, match, actions)
59
60     def add_flow(self, datapath, priority, match, actions, buffer_id=None):
61         ofproto = datapath.ofproto
62         parser = datapath.ofproto_parser
63
64         inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
65                                             actions)]
66         if buffer_id:
67             mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
68                                     priority=priority, match=match,
69                                     instructions=inst)
70         else:

```

```
71         mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
72                                   match=match, instructions=inst)
73         datapath.send_msg(mod)
74
75     def add_flow_group(self, datapath, priority, match, group_id):
76         ofproto = datapath.ofproto
77         parser = datapath.ofproto_parser
78
79         actions = [parser.OFPActionGroup(group_id=group_id)]
80         inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
81       actions)]
81         mod = parser.OFPFlowMod(datapath=datapath, priority=priority, match=
82       match, instructions=inst)
82         datapath.send_msg(mod)
```