

Write SDN Controller

计算机网络 CS339

李子龙 518070910095

2021 年 11 月 30 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	12
B	使用双路代码	15
C	断路备用代码	16

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:

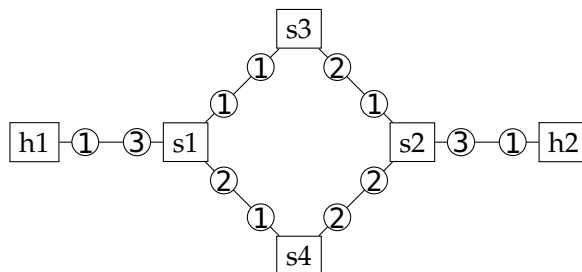


图 1: 网络拓扑

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。



```
root@ubuntu: /VMShared/Networking/lab05
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
^Croot@ubuntu: /VMShared/Networking/lab05# ryu-manager task2.py
loading app task2.py
loading app ryu.controller.ofp_handler
instantiating app task2.py of PeriodicSwitch
instantiating app ryu.controller.ofp_handler of OFPHandler
CONFIG switch id: 1
CONFIG switch id: 2
CONFIG switch id: 3
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

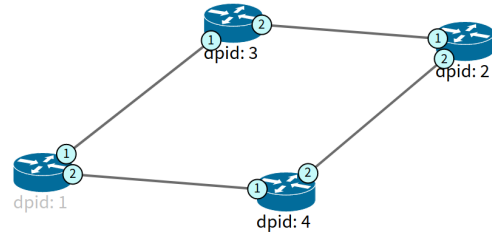


图 4: gui_topology 展示的拓扑结构

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。

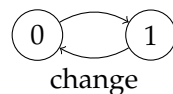


图 5: 状态机

3 使用双路

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

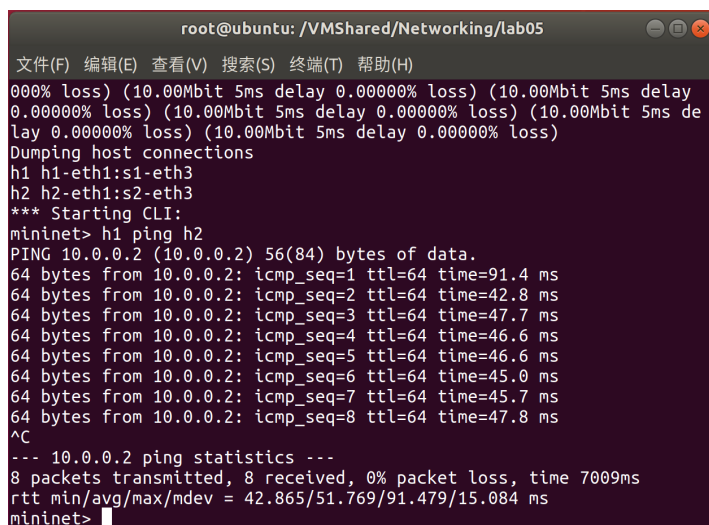
select: Execute one bucket in the group. Packets are processed by a single bucket in the group, based on a switch-computed selection algorithm (e.g. hash on some user-configured tuple or simple round robin). All configuration and state for the selection algorithm is external to OpenFlow. The selection 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 test was initiated by 'mininet> h1 ping h2'.

```
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]

将组表的构造参数变更为 `0FPGT_FF`，并分别监视 1 号和 2 号端口。在 Mininet CLI 中输入

```
link s1 s3 down
```

断开 s1 和 s3 之间的链路^[4]。

在前一个任务的代码上更改参数。但需要注意，此时输出组不能被设定权重。否则会报错

```
| -- type: OFPET_BAD_ACTION(2)
| -- code: OFPBAC_BAD_OUT_GROUP(9)
' -- data: version=0x4, msg_type=0xe, msg_len=0x50, xid=0x8b06b84c
      ' -- msg_type: OFPT_FLOW_MOD(14)
```

这是因为 Fast Failover 的要求就是当一个链路断开的时候能够使用另一个链路，而这个时候一条链路的权值会变成 0，如果赋予权值就会产生冲突。

Listing 6: [task4.py](#)

```
39         actions1 = [parser.OFPActionOutput(1)]
40         actions2 = [parser.OFPActionOutput(2)]
41         buckets = [
42             parser.OFPBucket(watch_port=1, actions=actions1),
43             parser.OFPBucket(watch_port=2, actions=actions2)]
44
45         group_id = 2
46         self.req_group(datapath, group_id, buckets)
47
48         match = parser.OFPMatch(in_port=3)
49         self.add_flow_group(datapath, 10, match, group_id)
```

Listing 7: [task4.py](#)

```
110         req = parser.OFPGroupMod(datapath, ofproto.OFPGC_ADD, ofproto.
        OFPGT_FF, group_id, buckets)
```

测试方法如下，将 ping 的数据写入文件，同时切断某条链路的连接查看反应。

```
h1 ping h2 -c 20 > pingtest.txt &
link s1 s3 down
link s2 s3 down
```

Listing 8: [pingtest.txt](#)

```
1 PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
2 64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=90.1 ms
3 64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=45.3 ms
4 64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=44.6 ms
5 64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=45.5 ms
6 64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=45.2 ms
7 64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=45.5 ms
8 64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=46.8 ms
9 64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=45.7 ms
10 64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=131 ms
11 64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=46.8 ms
12 64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=47.8 ms
13 64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=46.5 ms
14 64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=45.1 ms
```



```

15 64 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=128 ms
16 64 bytes from 10.0.0.2: icmp_seq=17 ttl=64 time=46.4 ms
17 64 bytes from 10.0.0.2: icmp_seq=18 ttl=64 time=45.1 ms
18 64 bytes from 10.0.0.2: icmp_seq=19 ttl=64 time=45.5 ms
19 64 bytes from 10.0.0.2: icmp_seq=20 ttl=64 time=47.9 ms
20
21 --- 10.0.0.2 ping statistics ---
22 20 packets transmitted, 18 received, 10% packet loss, time 19074ms
23 rtt min/avg/max/mdev = 44.647/57.821/131.614/27.488 ms

```

我们看到第3号包和第6号包之间出现了丢包。而且现在的版本如果不将另一条链路（s2和s3之间）也切断的话，会导致s2不知道这条链路已经切断，从而不可达。

Listing 9: [pingfail.txt](#)

```

1 PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
2 64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=91.1 ms
3 64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=47.8 ms
4
5 --- 10.0.0.2 ping statistics ---
6 20 packets transmitted, 2 received, 90% packet loss, time 19412ms
7 rtt min/avg/max/mdev = 47.896/69.541/91.186/21.645 ms

```

解决丢包问题这里主要有三种方案：

1. 改变当前拓扑结构，在 s3 和 s4 之间添加链路。
2. 通知另一侧交换机改变流表，关闭受损的链路。
3. 调度采用最小生成树算法，改变拓扑时即重新计算。

第一种在 B4 中叫做 *sidelink*^[5]，但是这种方法的缺点很明显：要改变拓扑，如果地理位置较远这种方法是不值得的，而且对于新加入的备用链路，出口处的传包如何安排流表也是问题：如果按照预期受损，单一转发方向还好，如图 8。如果只指向一个方向，那么另一侧链路受损时将不可达，如图 9。如果尝试对 s4 的 3 号端口采用 FF 转发的话，由于优先的那个端口并不是断开的，所以还是会传输，并不能避免丢包。（FF仅适用于端口完全断开连接）

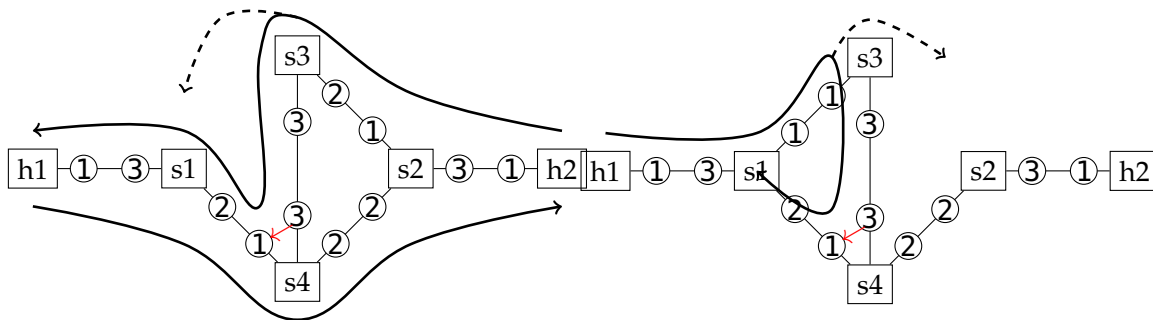


图 8: 正常备用转发

图 9: 备用不可达

第三种方法不是题面提示要求我们做的（需要使用 FF）。

下面主要采用第二种方法。EventOFPortStatus 可以检测链路改变。

Listing 10: `../ryu/ryu/ofproto/ofproto_v1_3_parser.py`

```
2521 @set_ev_cls(ofp_event.EventOFPortStatus, MAIN_DISPATCHER)
2522 def port_status_handler(self, ev):
2523     msg = ev.msg
2524     dp = msg.datapath
2525     ofp = dp.ofproto
2526
2527     if msg.reason == ofp.OFPPR_ADD:
2528         reason = 'ADD'
2529     elif msg.reason == ofp.OFPPR_DELETE:
2530         reason = 'DELETE'
2531     elif msg.reason == ofp.OFPPR_MODIFY:
2532         reason = 'MODIFY'
2533     else:
2534         reason = 'unknown'
2535
2536     self.logger.debug('OFPortStatus received: reason=%s desc=%s',
2537                       reason, msg.desc)
```

相关代码见附录 C。首先会在 s1 和 s2 设立两条备用转发规则（更高优先级）：如果出现了从 1 号端口回传的包，如果目的地不是该侧的 IP，那么就会被转发到 2 号端口；对 2 号端口也进行同样的设定。

之后在 EventOFPortStatus 事件中针对 MODIFY 进行处理，采用一个计时器来跟踪临近的识别操作（1s 为阈值），并排除开始的网络建立阶段，以及对 s3 和 s4 建立 FF 流组，如图 10。之后采用类内的变量来确定断开的链路是哪个链路，并通知对应的路由改变流表。注意由于需要通知不同的路由，需要得到当前网络的拓扑结构，启动时需要添加 `--observe-links` 参数发现路由。

```
ryu-manager --observe-links task4.py
```

如图 11，如果遇到断开包会原路返回（蓝色的流），而且此时 s2 已经不再使用 FF 组，而是直接转发来自 3 端口到 2 端口，以及直接将 1 端口转发到 2 端口以收取残留的原路返回包。

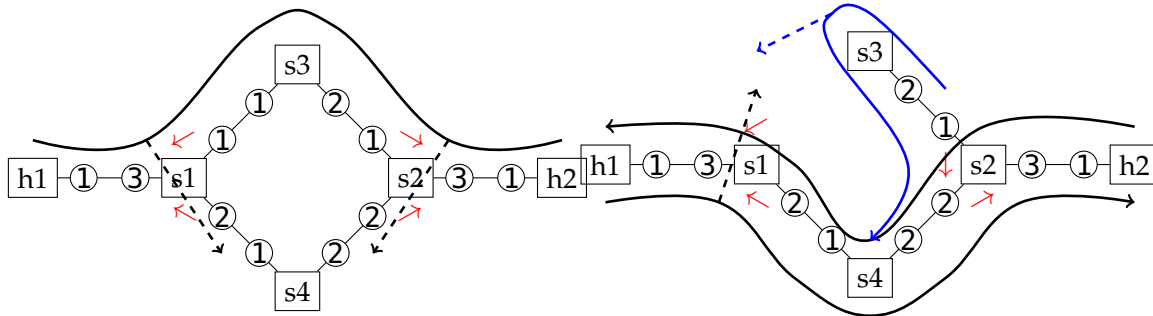


图 10: 正常链路流表

图 11: 断开时流表与转发

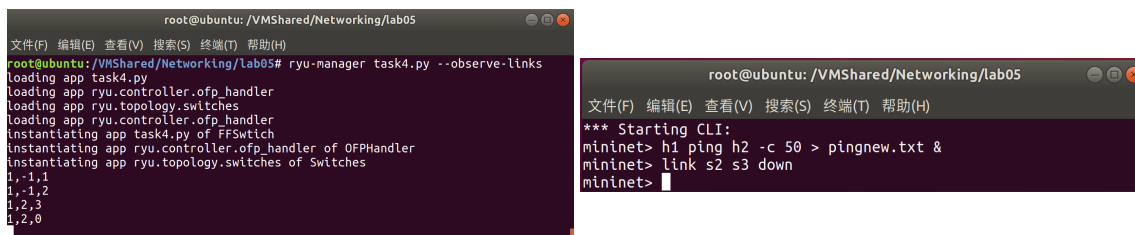


图 13: 断开链路

图 12: 状态改变

经过测试，这种方案将不会丢包。

Listing 11: pingnew.txt

```

1 PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
2 64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=93.9 ms
3 64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=47.8 ms
4 64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=45.7 ms
5 64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=45.8 ms
6 64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=46.8 ms
7 64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=46.7 ms
8 64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=45.8 ms
9 64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=45.8 ms
10 64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=46.5 ms
11 64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=44.8 ms
12 64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=43.6 ms
13 64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=47.0 ms
14 64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=46.1 ms
15 64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=45.1 ms
16 64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=45.0 ms
17 64 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=44.8 ms
18 64 bytes from 10.0.0.2: icmp_seq=17 ttl=64 time=45.8 ms
19 64 bytes from 10.0.0.2: icmp_seq=18 ttl=64 time=290 ms

```

```

20 64 bytes from 10.0.0.2: icmp_seq=19 ttl=64 time=131 ms
21 64 bytes from 10.0.0.2: icmp_seq=20 ttl=64 time=49.2 ms
22
23 --- 10.0.0.2 ping statistics ---
24 20 packets transmitted, 20 received, 0% packet loss, time 19043ms
25 rtt min/avg/max/mdev = 43.656/64.973/290.673/55.824 ms

```

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、. 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>.
- [4] 二十年后20. mininet命令[EB/OL]. 2017. <https://www.cnblogs.com/cing/p/8025239.html>.
- [5] HONG C Y, MANDAL S, AL-FARES M, et al. B4 and after[C/OL]//ACM, 2018. DOI: 10.1145/3230543.3230545.

A 定时切换代码

Listing 12: `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,
29         ipv4_src='10.0.0.1', ipv4_dst='10.0.0.2', ip_proto=inet.IPPROTO_UDP,
30         udp_dst=5555)
31         actions = [parser.OFPActionOutput(out_port)]
32         self.add_flow(datapath, 0, match, actions)
33
34     def add_flow(self, datapath, priority, match, actions, buffer_id=None):
35         """
36         Default adding flow.
37         """
38         ofproto = datapath.ofproto
39         parser = datapath.ofproto_parser
40
41         inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
42         actions)]
43         if buffer_id:
44             mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
45             priority=priority, match=match,
46             instructions=inst)
47         else:
48             mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
49             match=match, instructions=inst)
50         datapath.send_msg(mod)
51
52     def add_flow_timeout(self, datapath, priority, match, actions, buffer_id=
53     None):
54         """
55         Add a flow that timeout in 5 sec.
56         """
57         ofproto = datapath.ofproto
58         parser = datapath.ofproto_parser
59
60         inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
61         actions)]
62         if buffer_id:
63             mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
64             priority=priority, match=match,
65             instructions=inst, hard_timeout=5, flags=
66             ofproto.OFPFF_SEND_FLOW_REM)
67         else:
68             mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
69             match=match, instructions=inst,
70             hard_timeout=5, flags=ofproto.OFPFF_SEND_FLOW_REM)
71         datapath.send_msg(mod)
72
73     @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
74     def switch_features_handler(self, ev):

```

```

70     global pathport
71
72     datapath = ev.msg.datapath
73     ofproto = datapath.ofproto
74     parser = datapath.ofproto_parser
75
76     # Since the switches are added in order,
77     # The id is appended in order as well.
78     print('CONFIG switch id: ' + str(datapath.id))
79
80     if datapath.id == 1 or datapath.id == 2:
81         # forward flow h1 -> s1(s2)
82         # input from port 3, output to the selected port.
83         match = parser.OFPMatch(in_port=3)
84         actions = [parser.OFPActionOutput(pathport)] #
85         self.add_flow_timeout(datapath, 2, match, actions)
86
87         # return flow s1(s2) -> h1
88         # 2 possible flows: from port 1, from port 2.
89         match = parser.OFPMatch(in_port=1)
90         actions = [parser.OFPActionOutput(3)]
91         self.add_flow(datapath, 2, match, actions)
92         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.EventOFPPFlowRemoved, 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('Switch to port: ' + str(pathport))
120                print('OFPPFlowRemoved 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 13: [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 BalancedSwitch(app_manager.RyuApp):
12     OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
13
14     def __init__(self, *_args, **_kwargs):
15         super(BalancedSwitch, 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)]
82         mod = parser.OFPFlowMod(datapath=datapath, priority=priority, match=
83                                 match, instructions=inst)
84         datapath.send_msg(mod)

```

C 断路备用代码

Listing 14: [task4.py](#)

```

1  # 4. Write an RYU controller that uses the first path (h1-s1-s3-s2-h2) for
2  routing packets from h1 to h2 and uses the second path for backup.
3  Specifically, when the first path experiences a link failure, the network
4  should automatically switch to the second path without causing packet
5  drop. (hint: consider using \verb"OFPGT_FF" (FF is short for "fast
6  failover")) to construct a group table)
7
8  from ryu.base import app_manager
9  from ryu.controller import ofp_event
10 from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER,
11     set_ev_cls
12 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 from ryu.topology.api import get_switch, get_link
12 from ryu.topology import event, switches
13 from time import time
14
15 LEFT = 0
16 RIGHT = 1
17 UPPER = 2
18 BOTTOM = 3
19
20 class FFSwtich(app_manager.RyuApp):
21     OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
22
23     def __init__(self, *_args, **_kwargs):
24         super(FFSwtich, self).__init__(*_args, **_kwargs)
25         self.lr = -1
26         self.ub = -1
27         self.prev_time = -1
28         self.change_state = 0
29
30     @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
31     def switch_features_handler(self, ev):
32         datapath = ev.msg.datapath
33         ofproto = datapath.ofproto
34         parser = datapath.ofproto_parser
35
36         if datapath.id == 1 or datapath.id == 2:
37             # s1 / s2
38             # bucket group
39             actions1 = [parser.OFPActionOutput(1)]
40             actions2 = [parser.OFPActionOutput(2)]
41             buckets = [
42                 parser.OFPBucket(watch_port=1, actions=actions1),
43                 parser.OFPBucket(watch_port=2, actions=actions2)]
44
45             group_id = 2
46             self.req_group(datapath, group_id, buckets)
47
48             match = parser.OFPMatch(in_port=3)
49             self.add_flow_group(datapath, 10, match, group_id)
50
51             match_src = '10.0.0.1' if datapath.id == 1 else '10.0.0.2'
52             match_dst = '10.0.0.2' if datapath.id == 1 else '10.0.0.1'
53             match = parser.OFPMatch(in_port=1, eth_type=ether_types.
54                                     ETH_TYPE_IP, ipv4_src=match_src, ipv4_dst=match_dst)
55             self.add_flow(datapath, 15, match, actions2)
56             match = parser.OFPMatch(in_port=2, eth_type=ether_types.
57                                     ETH_TYPE_IP, ipv4_src=match_src, ipv4_dst=match_dst)
58             self.add_flow(datapath, 15, match, actions1)
59
60             # return flow s1(s2) -> h1
61             # 2 possible flows: from port 1, from port 2.
62             match = parser.OFPMatch(in_port=1)

```

```

61         actions = [parser.OFPActionOutput(3)]
62         self.add_flow(datapath, 10, match, actions)
63         match = parser.OFPMatch(in_port=2)
64         actions = [parser.OFPActionOutput(3)]
65         self.add_flow(datapath, 10, match, actions)
66     elif datapath.id == 3 or datapath.id == 4:
67         # s3 / s4
68
69         actions1 = [parser.OFPActionOutput(1)]
70         actions2 = [parser.OFPActionOutput(2)]
71         actions3 = [parser.OFPActionOutput(3)]
72
73         # fail over for port 1
74         buckets1 = [
75             parser.OFPBucket(watch_port=2, actions=actions2),
76             parser.OFPBucket(watch_port=3, actions=actions3)]
77         group_id = 3
78         self.req_group(datapath, group_id, buckets1)
79         match = parser.OFPMatch(in_port=1)
80         self.add_flow_group(datapath, 10, match, group_id)
81
82         # fail over for port 2
83         buckets2 = [
84             parser.OFPBucket(watch_port=1, actions=actions1),
85             parser.OFPBucket(watch_port=3, actions=actions3)]
86         group_id = 4
87         self.req_group(datapath, group_id, buckets2)
88         match = parser.OFPMatch(in_port=2)
89         self.add_flow_group(datapath, 10, match, group_id)
90
91     def add_flow(self, datapath, priority, match, actions, buffer_id=None):
92         ofproto = datapath.ofproto
93         parser = datapath.ofproto_parser
94
95         inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
96                                             actions)]
97         if buffer_id:
98             mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
99                                     priority=priority, match=match,
100                                     instructions=inst)
101         else:
102             mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
103                                     match=match, instructions=inst)
104         datapath.send_msg(mod)
105
106     def req_group(self, datapath, group_id, buckets):
107         ofproto = datapath.ofproto
108         parser = datapath.ofproto_parser
109
110         req = parser.OFPGroupMod(datapath, ofproto.OFPGC_ADD, ofproto.
111 OFPGT_FF, group_id, buckets)
112         datapath.send_msg(req)
113
114     def add_flow_group(self, datapath, priority, match, group_id):
115         ofproto = datapath.ofproto
116         parser = datapath.ofproto_parser

```

```

116         actions = [parser.OFPActionGroup(group_id=group_id)]
117         inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
118         actions)]
119         mod = parser.OFPFlowMod(datapath=datapath, priority=priority, match=
match, instructions=inst)
120         datapath.send_msg(mod)
121
122     @set_ev_cls(ofp_event.EventOFPPortStatus, MAIN_DISPATCHER)
123     def port_status_handler(self, ev):
124         msg = ev.msg
125         datapath = msg.datapath
126         ofproto = datapath.ofproto
127         if msg.reason == ofproto.OFPPR_MODIFY:
128             if self.prev_time < 0 or time()-self.prev_time <= 1:
129                 # not recorded: at the build stage.
130                 # it is too close as the same op: don't make duplicated work.
131                 self.prev_time = time() # refresh time and move on.
132                 return
133             # regard it as different op.
134             # will not refresh the previous time
135             # as we need to locate the location of the broken link.
136             if datapath.id == 1:
137                 # left
138                 self.lr = LEFT
139             elif datapath.id == 2:
140                 # right
141                 self.lr = RIGHT
142             elif datapath.id == 3:
143                 # upper link
144                 self.ub = UPPER
145             elif datapath.id == 4:
146                 # bottom link
147                 self.ub = BOTTOM
148             self.change_state = self.change_state + 1 if self.change_state <
3 else 0
149             print(str(self.lr) + "," + str(self.ub) + "," + str(self.
change_state))
150             if self.change_state == 0:
151                 # make the change.
152                 # the information is efficient enough to make adjustment.
153                 switch_list = get_switch(self)
154                 if self.lr == LEFT:
155                     # notify s2
156                     target_switch = switch_list[1]
157                 else:
158                     # notify s1
159                     target_switch = switch_list[0]
160                 dp = target_switch.dp
161                 ofp = datapath.ofproto
162                 parser = dp.ofproto_parser
163                 if self.ub == UPPER:
164                     # port3 -> port2
165                     match = parser.OFPMatch(in_port=3)
166                     actions = [parser.OFPActionOutput(2)]
167                     self.add_flow(dp, 20, match, actions)

```

```
168         else:
169             # port3 -> port1
170             match = parser.OFPMatch(in_port=3)
171             actions = [parser.OFPACTIONOutput(1)]
172             self.add_flow(dp, 20, match, actions)
```