Write SDN Controller

计算机网络 CS339

李子龙 518070910095

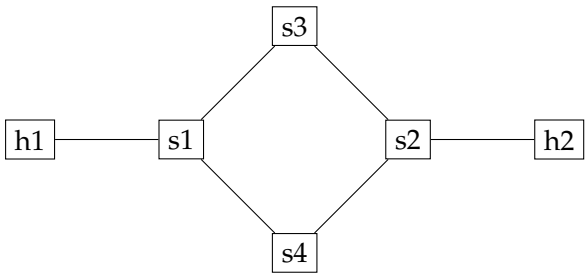
2021 年 11 月 1 日

目录

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:



使用给出的示例代码。

Listing 1: loopnet.py

```
#!/usr/bin/python
"""Sample Code"""
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import OVSBridge, OVSSwitch, OVSKernelSwitch
from mininet.node import CPULimitedHost
from mininet.node import RemoteController
from mininet.link import TCLink
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel, info
from mininet.cli import CLI
from sys import argv
def Test():
    "Create network and run simple performance test"
    net = Mininet( switch=OVSSwitch,host=CPULimitedHost, link=TCLink,
    autoStaticArp=False, controller=RemoteController)
    switch1 = net.addSwitch('s1')
    switch2 = net.addSwitch('s2')
    switch3 = net.addSwitch('s3')
    switch4 = net.addSwitch('s4')
    host1 = net.addHost('h1', cpu=.25)
    host2 = net.addHost('h2', cpu=.25)
    net.addLink(host1, switch1, bw=10, delay='5ms', loss=0, use_htb=True)
    net.addLink(host2, switch2, bw=10, delay='5ms', loss=0, use_htb=True)
    net.addLink(switch1, switch3, bw=10, delay='5ms', loss=0, use_htb=True)
    net.addLink(switch1, switch4, bw=10, delay='5ms', loss=0, use_htb=True)
    net.addLink(switch2, switch3, bw=10, delay='5ms', loss=0, use_htb=True)
    net.addLink(switch2, switch4, bw=10, delay='5ms', loss=0, use_htb=True)
    c1 = net.addController('c1', controller=RemoteController, ip="127.0.0.1",
     port=6653)
    net.build()
    c1.start()
    s1, s2, s3, s4 = net.getNodeByName('s1', 's2', 's3', 's4')
    s1.start([c1])
    s2.start([c1])
    s3.start([c1])
    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                               msg.cookie, msg.priority, reason, msg.table_id,
2400                               msg.duration_sec, msg.duration_nsec,
2401                               msg.idle_timeout, msg.hard_timeout,
2402                               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。

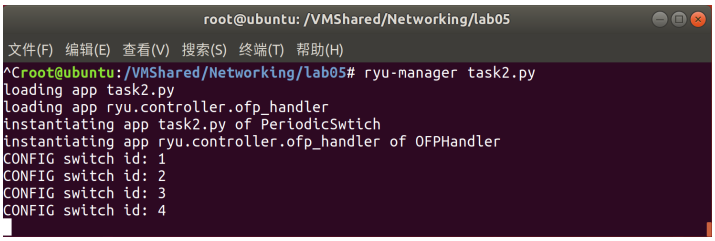


图 1: 交换机编号

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
```

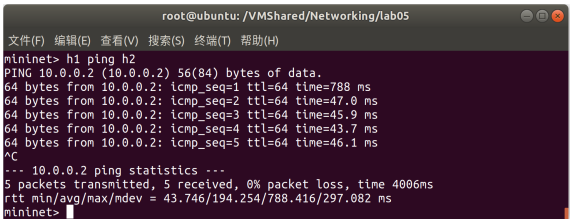


图 2: 测试连接

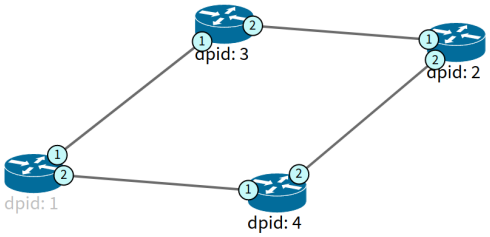


图 3: `gui_topology` 展示的拓扑结构

由图 3 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。在图 2 中可见是能够 `ping` 通的。相关代码见附录 A。

3

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

4

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 `OFPGT_FF` (FF is short for "fast failover") to construct a group table)

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.

A 定时切换代码

Listing 5: task2.py

```python
# 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.

from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER, set_ev_cls
from ryu.lib import packet
from ryu.lib.packet import ether_types, ethernet
from ryu.lib.packet import in_proto as inet
from ryu.ofproto import ofproto_v1_3

pathport = 1

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

    def __init__(self, *_args, **_kwargs):
        super(PeriodicSwtich, self).__init__(*_args, **_kwargs)

    @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
        out_port = 1

        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)
        actions = [parser.OFPActionOutput(out_port)]
        self.add_flow(datapath, 0, match, actions)

    def add_flow(self, datapath, priority, match, actions, buffer_id=None):
        """
        Default adding flow.
        """
        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)

```

```
48      def add_flow_timeout(self, datapath, priority, match, actions, buffer_id=
        None):
49          """
50          Add a flow that timeout in 5 sec.
51          """
52          ofproto = datapath.ofproto
53          parser = datapath.ofproto_parser
54
55          inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
56                                               actions)]
57          if buffer_id:
58              mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
59                                      priority=priority, match=match,
60                                      instructions=inst, hard_timeout=5, flags=
        ofproto.OFPFF_SEND_FLOW_REM)
61          else:
62              mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
63                                      match=match, instructions=inst,
        hard_timeout=5, flags=ofproto.OFPFF_SEND_FLOW_REM)
64          datapath.send_msg(mod)
65
66      @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
67      def switch_features_handler(self, ev):
68          global pathport
69
70          datapath = ev.msg.datapath
71          ofproto = datapath.ofproto
72          parser = datapath.ofproto_parser
73
74          # Since the switches are added in order,
75          # The id is appended in order as well.
76          print('CONFIG switch id: '+ str(datapath.id))
77
78          if datapath.id == 1 or datapath.id == 2:
79              # forward flow h1 -> s1(s2)
80              # input from port 3, output to the selected port.
81              match = parser.OFPMatch(in_port=3)
82              actions = [parser.OFPActionOutput(pathport)] #
83              self.add_flow_timeout(datapath, 2, match, actions)
84
85              # return flow s1(s2) -> h1
86              # 2 possible flows: from port 1, from port 2.
87              match = parser.OFPMatch(in_port=1)
88              actions = [parser.OFPActionOutput(3)]
89              self.add_flow(datapath, 2, match, actions)
90              match = parser.OFPMatch(in_port=2)
91              actions = [parser.OFPActionOutput(3)]
92              self.add_flow(datapath, 2, match, actions)
93          # elif datapath.id == 3:
94          elif datapath.id == 3 or datapath.id == 4:
95              # s3
96              match = parser.OFPMatch(in_port=1)
97              actions = [parser.OFPActionOutput(2)]
98              self.add_flow(datapath, 2, match, actions)
99          # elif datapath.id == 4:
100             # s4
```

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

    @set_ev_cls(ofp_event.EventOFPFlowRemoved, MAIN_DISPATCHER)
    def flow_removed_handler(self, ev):
        global pathport

        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        if msg.reason == ofproto.OFPRR_HARD_TIMEOUT:
            if datapath.id == 1:
                pathport = 2 if pathport==1 else 1
                # change on pathport could only be invoked once.
                print('Swtich to port: ' + str(pathport))
            print('OFPFlowRemoved received: ' + str(datapath.id))
            match = parser.OFPMatch(in_port=3)
            actions = [parser.OFPActionOutput(pathport)]
            self.add_flow_timeout(datapath, 2, match, actions)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        if ev.msg.msg_len < ev.msg.total_len:
            self.logger.debug("packet truncated: only %s of %s bytes",
                              ev.msg.msg_len, ev.msg.total_len)
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']

        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocols(ethernet.ethernet)[0]

        if eth.ethertype == ether_types.ETH_TYPE_LLDP:
            # ignore lldp packet
            return
        dst = eth.dst
        src = eth.src

        actions = [parser.OFPActionOutput(pathport)]

        # 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_timeout(datapath, 1, match, actions, msg.buffer_id)
            return
        else:
            self.add_flow_timeout(datapath, 1, match, actions)
        data = None
```

```
157             if msg.buffer_id == ofproto.OFP_NO_BUFFER:
158                 data = msg.data
159 
160             out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
161                                       in_port=in_port, actions=actions, data=data
        )
162             datapath.send_msg(out)
```