流量监控

```python
from operator import attrgetter

from ryu.app import simple_switch_13
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER, DEAD_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.lib import hub


class SimpleMonitor13(simple_switch_13.SimpleSwitch13):

    def __init__(self, *args, **kwargs):
        super(SimpleMonitor13, self).__init__(*args, **kwargs)
        self.datapaths = {}
        self.monitor_thread = hub.spawn(self._monitor)

    @set_ev_cls(ofp_event.EventOFPStateChange,
                [MAIN_DISPATCHER, DEAD_DISPATCHER])
    def _state_change_handler(self, ev):
        datapath = ev.datapath
        if ev.state == MAIN_DISPATCHER:
            if datapath.id not in self.datapaths:
                self.logger.debug('register datapath: %016x', datapath.id)
                self.datapaths[datapath.id] = datapath
        elif ev.state == DEAD_DISPATCHER:
            if datapath.id in self.datapaths:
                self.logger.debug('unregister datapath: %016x',
datapath.id)
                del self.datapaths[datapath.id]

    def _monitor(self):
        while True:
            for dp in self.datapaths.values():
                self._request_stats(dp)
            hub.sleep(10)

    def _request_stats(self, datapath):
        self.logger.debug('send stats request: %016x', datapath.id)
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        req = parser.OFPFlowStatsRequest(datapath)
        datapath.send_msg(req)

        req = parser.OFPPortStatsRequest(datapath, 0, ofproto.OFPP_ANY)
        datapath.send_msg(req)

    @set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
    def _flow_stats_reply_handler(self, ev):
        body = ev.msg.body
        self.logger.info('datapath         '
                         'in-port  eth-dst           '
                         'out-port packets  bytes')
        self.logger.info('---------------- '
                         '-------- ----------------- '
                         '-------- -------- --------')
        for stat in sorted([flow for flow in body if flow.priority == 1],
                           key=lambda flow: (flow.match['in_port'],
                                             flow.match['eth_dst'])):
            self.logger.info('%016x %8x %17s %8x %8d %8d',
                             ev.msg.datapath.id,
                             stat.match['in_port'], stat.match['eth_dst'],
                             stat.instructions[0].actions[0].port,
                             stat.packet_count, stat.byte_count)

    @set_ev_cls(ofp_event.EventOFPPortStatsReply, MAIN_DISPATCHER)
    def _port_stats_reply_handler(self, ev):
        body = ev.msg.body

        self.logger.info('datapath         port     '
                         'rx-pkts  rx-bytes rx-error '
                         'tx-pkts  tx-bytes tx-error')
        self.logger.info('---------------- -------- '
                         '-------- -------- -------- '
                         '-------- -------- --------')
        for stat in sorted(body, key=attrgetter('port_no')):
            self.logger.info('%016x %8x %8d %8d %8d %8d %8d %8d',
                             ev.msg.datapath.id, stat.port_no,
                             stat.rx_packets, stat.rx_bytes, stat.rx_errors,
                             stat.tx_packets, stat.tx_bytes, stat.tx_errors)
```

代码详解

(1)simple_switch_13.SimpleSwitch13是样例代码，实现了自学习交换机类似的功能

多了个关于交换机是否上传全部packet还是只上传buffer_id

```

class SimpleMonitor13(simple_switch_13.SimpleSwitch13):

```

(2)协程实现伪并发self.monitor_thread = hub.spawn(self._monitor)

```
    def __init__(self, *args, **kwargs):
        super(SimpleMonitor13, self).__init__(*args, **kwargs)
        self.datapaths = {}
        self.monitor_thread = hub.spawn(self._monitor)
```

(3)在协程中实现周期性请求交换机信息

```
    def _monitor(self):
        while True:
            for dp in self.datapaths.values():
                self._request_stats(dp)
            hub.sleep(10)
```

(4)当交换机状态发生变化时，会触发该事件处理函数。该函数首先获取事件中的datapath信息，并判断datapath的状态是MAIN_DISPATCHER还是DEAD_DISPATCHER。

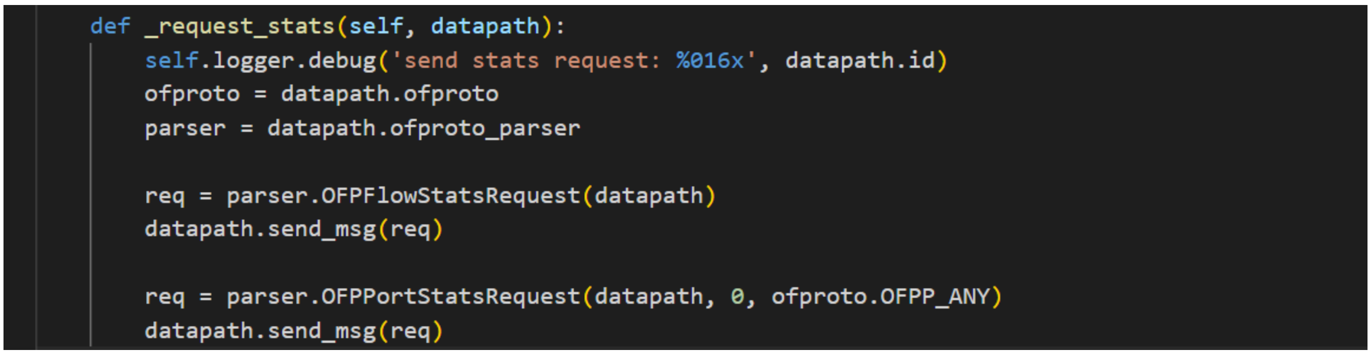
如果datapath的状态是MAIN_DISPATCHER，说明该交换机已经连接到控制器， 需要将其信息保存到控制器的datapahts字典中。如果该交换机的id已经存在于datapahts字典中，则不需要再次保存。最后打印调试信息，表示已经成功注册该datapath。

如果datapath的状态是DEAD_DISPATCHER，说明该交换机已经从控制器中断开连接， 需要将其信息从datapahts字典中删除。如果该交换机的id已经不存在于datapahts字典中， 则不需要再次删除。最后打印调试信息，表示已经成功注销该datapath。

```
    @set_ev_cls(ofp_event.EventOFPStateChange,
                [MAIN_DISPATCHER, DEAD_DISPATCHER])
    def _state_change_handler(self, ev):
        datapath = ev.datapath
        if ev.state == MAIN_DISPATCHER:
            if datapath.id not in self.datapaths:
                self.logger.debug('register datapath: %016x', datapath.id)
                self.datapaths[datapath.id] = datapath
        elif ev.state == DEAD_DISPATCHER:
            if datapath.id in self.datapaths:
                self.logger.debug('unregister datapath: %016x', datapath.id)
                del self.datapaths[datapath.id]
```

（5）主动下发消息，请求交换机信息OFPFlowStatsRequest

这里我们请求两个(端口和协议信息)，我们使用两个函数来分别处理port和flow响应

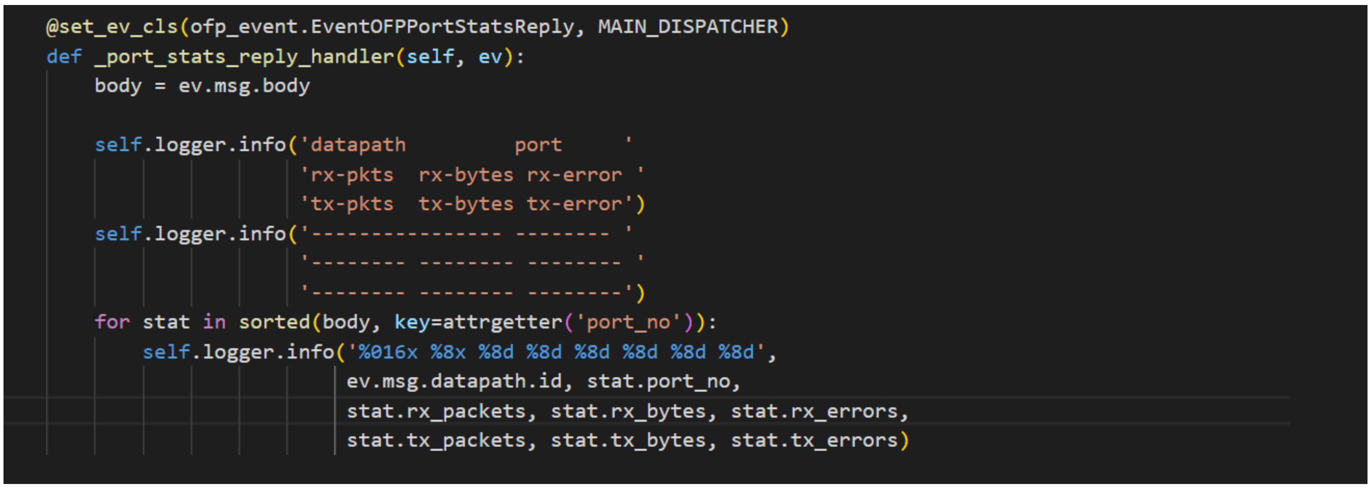


```python
    def _request_stats(self, datapath):
        self.logger.debug('send stats request: %016x', datapath.id)
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        req = parser.OFPFlowStatsRequest(datapath)
        datapath.send_msg(req)

        req = parser.OFPPortStatsRequest(datapath, 0, ofproto.OFPP_ANY)
        datapath.send_msg(req)
```

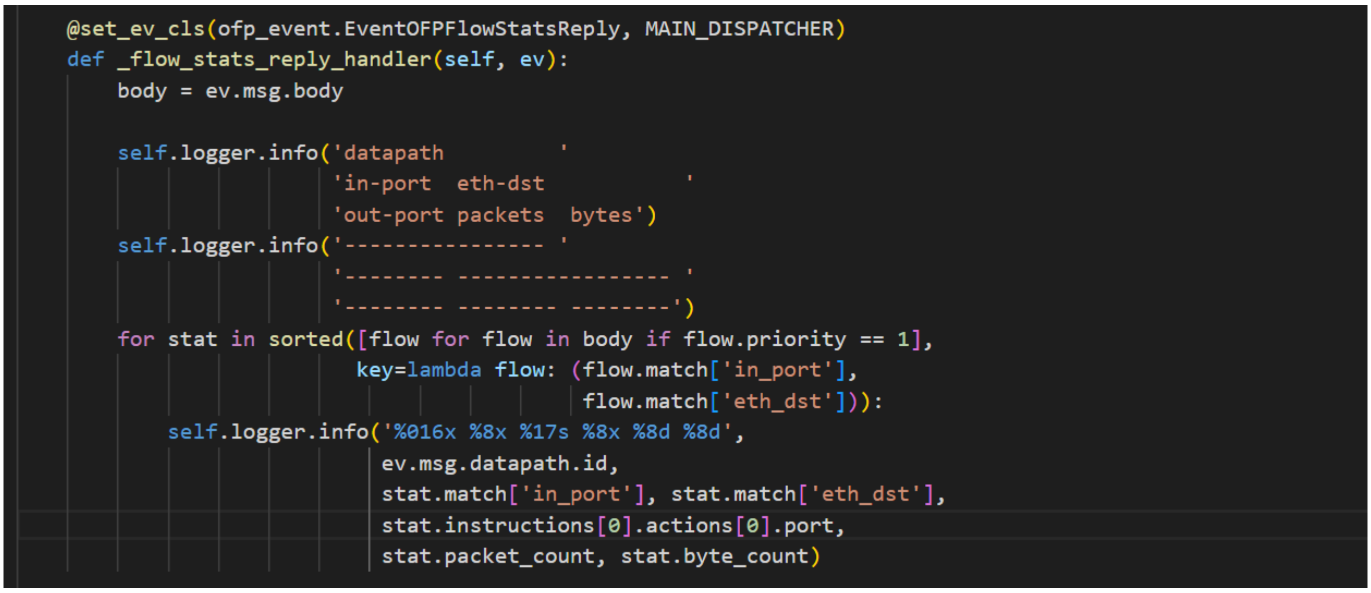
(6)获取端口响应消息ofp_event.EventOFPPortStatsReply



```python
@set_ev_cls(ofp_event.EventOFPPortStatsReply, MAIN_DISPATCHER)
def _port_stats_reply_handler(self, ev):
    body = ev.msg.body

    self.logger.info('datapath         port     '
                     'rx-pkts  rx-bytes rx-error '
                     'tx-pkts  tx-bytes tx-error')
    self.logger.info('---------------- -------- '
                     '-------- -------- -------- '
                     '-------- -------- --------')
    for stat in sorted(body, key=attrgetter('port_no')):
        self.logger.info('%016x %8x %8d %8d %8d %8d %8d %8d',
                         ev.msg.datapath.id, stat.port_no,
                         stat.rx_packets, stat.rx_bytes, stat.rx_errors,
                         stat.tx_packets, stat.tx_bytes, stat.tx_errors)
```

（7）获取flow协议响应消息ofp_event.EventOFPFlowStatsReply



```python
@set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
def _flow_stats_reply_handler(self, ev):
    body = ev.msg.body

    self.logger.info('datapath         '
                     'in-port  eth-dst           '
                     'out-port packets  bytes')
    self.logger.info('---------------- '
                     '-------- ----------------- '
                     '-------- -------- --------')
    for stat in sorted([flow for flow in body if flow.priority == 1],
                       key=lambda flow: (flow.match['in_port'],
                                         flow.match['eth_dst'])):
        self.logger.info('%016x %8x %17s %8x %8d %8d',
                         ev.msg.datapath.id,
                         stat.match['in_port'], stat.match['eth_dst'],
                         stat.instructions[0].actions[0].port,
                         stat.packet_count, stat.byte_count)
```