Ryu的应用开发（三）流量监控

一：实现流量监控

掌握基于Ryu开发流量监控应用：主动下发逻辑

（一）流量监控原理

控制器向交换机周期下发获取统计消息，请求交换机信息
- 端口流量统计信息
- 请求流表项统计信息（提高）

根据交换机统计信息计算计算流量信息
- 流速公式：speed = (s(t1) - s(t0))/(t1-t0)
- 剩余带宽公式： free_bw = capability - speed

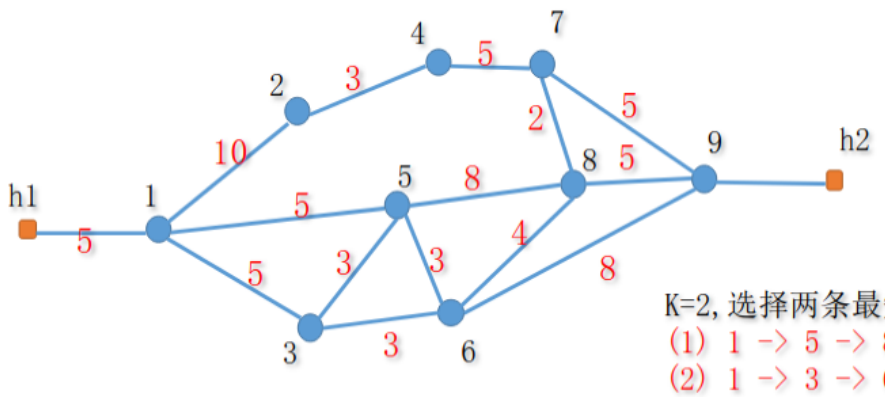
其中控制器向交换机周期下发获取统计消息，请求交换机消息-------是主动下发过程

流速公式：是（t1时刻的流量-t0时刻的流量）/（t1-t0）

剩余带宽公式：链路总带宽-流速---------是这一个这一个,例如s2-s3（不是一条，例如：h1->s1->s2->s3->h2）的剩余带宽

路径有效带宽是只：这一整条路径中，按照最小的剩余带宽处理

基于流量的最有路径转发示意图



K=2, 选择两条最短跳数的路径
(1) 1 -> 5 -> 8 -> 9
(2) 1 -> 3 -> 6 -> 9

然后根据可用带宽来确定最优路径：
(1) min(5, 8, 5) = 5
(2) min(5, 3, 8) = 3
所以最优路径选择（1）。

二：代码实现

（一）代码框架

Python 复制代码

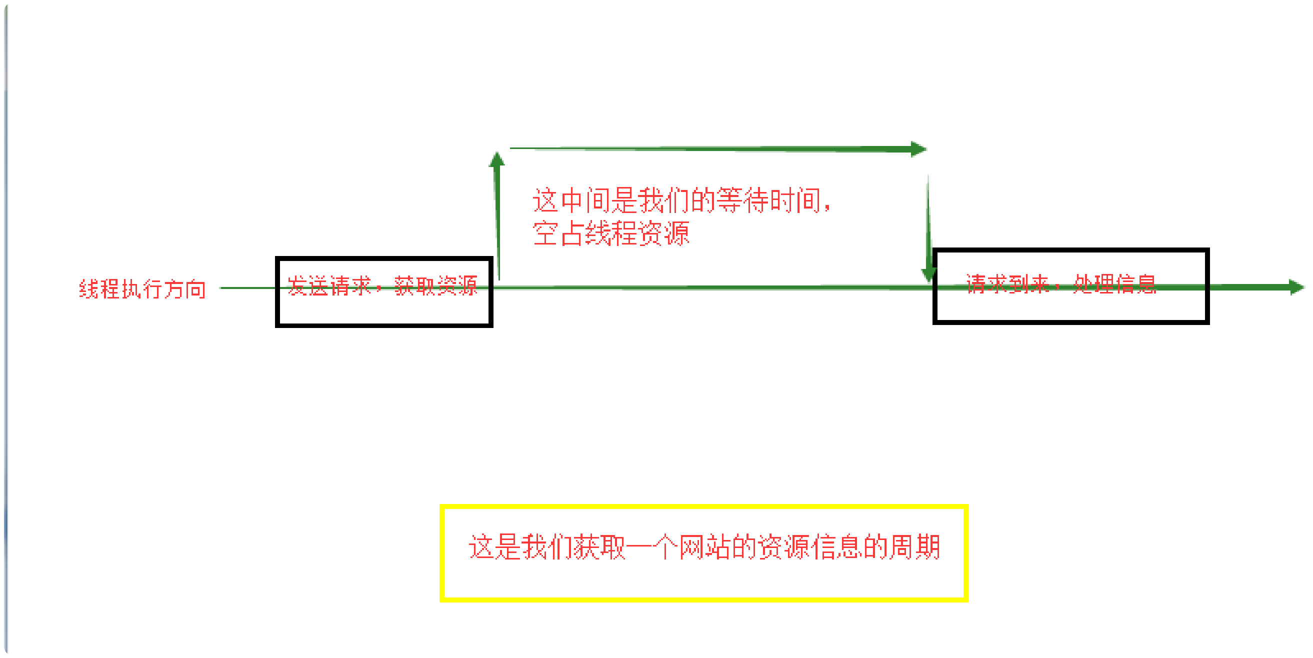
```python
from ryu.app import simple_switch_13
from ryu.controller.handler import set_ev_cls
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER,DEAD_DISPATCHER

class MyMonitor(simple_switch_13):    #simple_switch_13 is same as the last experiment which named self_learn_switch
    '''
    design a class to achvie managing the quantity of flow
    '''

    def __init__(self,*args,**kwargs):
        super(MyMonitor,self).__init__(*args,**kwargs)

    @set_ev_cls(ofp_event.EventOFPStateChange,[MAIN_DISPATCHER,DEAD_DISPATCHER])
    def _state_change_handler(self,ev):
        '''
        design a handler to get switch state transition condition
        '''
        pass

    def _monitor(self):
        '''
        design a monitor on timing system to request switch infomations about port and flow
        '''
        pass

    def _request_stats(self,datapath):
        '''
        the function is to send requery to datapath
        '''
        pass

    @set_ev_cls(ofp_event.EventOFPPortStatsReply,MAIN_DISPATCHER)
    def _port_stats_reply_handler(self,ev):
        '''
        monitor to require the port state, then this function is to get infomation for port`s info
        '''
        pass

    @set_ev_cls(ofp_event.EventOFPFlowStatsReply,MAIN_DISPATCHER)
    def _port_stats_reply_handler(self,ev):
        '''
        monitor to require the flow state, then this function is to get in
fomation for flow`s info
        '''
        pass
```

（二）推文：协程

https://www.cnblogs.com/ssyfj/p/9030165.html

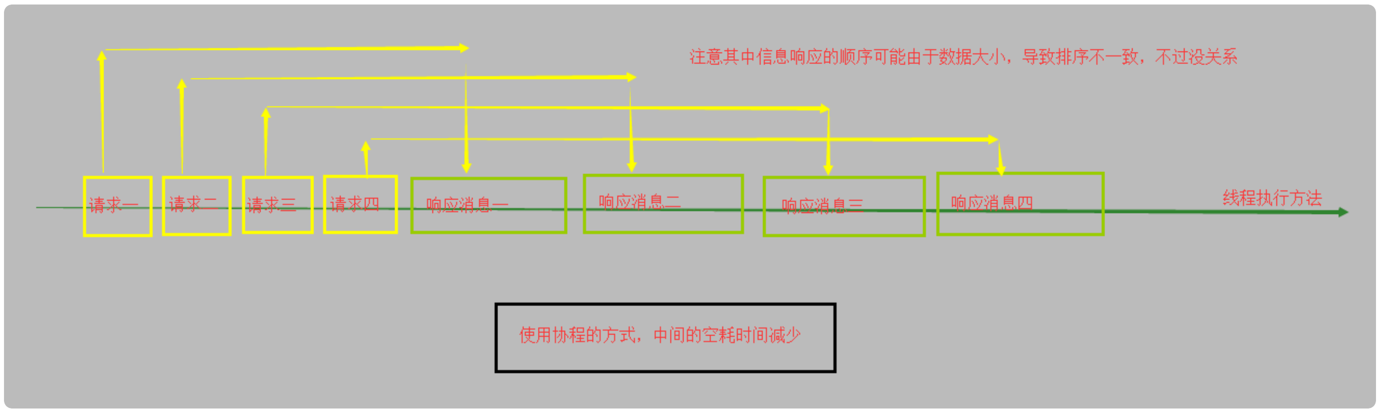
优点：使用gevent协程，可以更好的利用线程资源。（基于线程实现）

需求：使用一个线程，去请求多个网站的资源（注意，请求上会有延时）<实际上是去请求了大量的网站信息，我们使用了多线程，只不过每个线程依旧会分配到多个网站资源，这里我们只需要去讨论这一条线程即可>


线程执行方向
发送请求，获取资源
这中间是我们的等待时间，空占线程资源
请求到来，处理信息
这是我们获取一个网站的资源信息的周期

可以看出，由于网络延迟等因素，当我们去获取信息时，有一段时间被浪费用于空等信息返回，当我们去获取大量网站的时候，那这个时间是非常大的。我们需要去避免他。

解决方案：使用协程，充分利用我们中间等待的这一段时间，去做其他的事情，比如其请求下一个网站，或者下几个网站。然后连续去接收信息，就可以充分的利用空耗的时间


注意其中信息响应的顺序可能由于数据大小，导致排序不一致，不过没关系
请求一
请求二
请求三
请求四
响应消息一
响应消息二
响应消息三
响应消息四
线程执行方法
使用协程的方式，中间的空耗时间减少

1.协程的简单使用：

Shell 复制代码

```shell
pip3 install gevent    # gevent模块若是没有，只需要先下载
```

开始使用：

Python 复制代码

```python
import gevent
from gevent import monkey

monkry.patch_all()    #可以提高效率

def foo():
    print("foo函数开始运行")
    gevent.sleep(0)
    print("又回到了foo函数")

def bar():
    print("bar函数开始运行")
    gevent.sleep(0)
    print("又回到了bar函数")

gevent.joinall([
    gevent.spawn(foo),
    gevent.spawn(bar),
])
```

输出结果：

foo函数开始运行

bar函数开始运行
又回到了foo函数
又回到了bar函数

2.协程的了解：对于上面的例子来说，有点不太容易理解，我们使用计时去了解其中流程，再去讨论上面代码

（1）上面sleep(0)和下面的sleep(3)相比，得出两个函数的执行时间是一致的（几乎是）

```python
import gevent
import time

begin = time.time()

def foo():
    fs = time.time() - begin
    print("foo函数开始运行",fs)

    gevent.sleep(3)

    fe = time.time() - begin
    print("又回到了foo函数",fe)

def bar():
    bs = time.time() - begin
    print("bar函数开始运行",bs)

    gevent.sleep(3)

    be = time.time() - begin
    print("又回到了bar函数",be)


gevent.joinall([
    gevent.spawn(foo),
    gevent.spawn(bar),
])
```

foo函数开始运行 0.010000705718994141
bar函数开始运行 0.010000705718994141

又回到了foo函数 3.0101723670959473
又回到了bar函数 3.0101723670959473

注意输出结果

我们可以看出两个函数都是在统一时间执行第一句输出，在三秒后去执行的第二句输出

（2）sleep(3)和sleep(1)

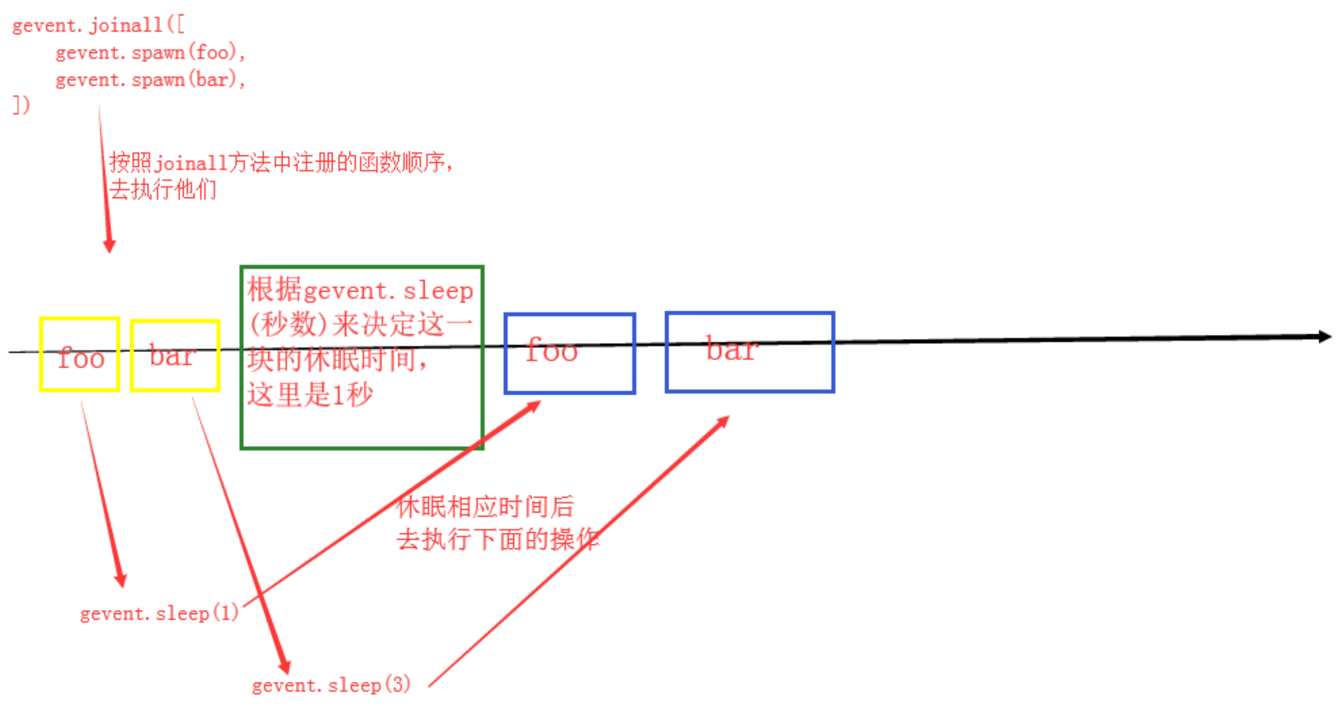
Python 复制代码

```python
import gevent
import time

begin = time.time()

def foo():
    fs = time.time() - begin
    print("foo函数开始运行",fs)

    gevent.sleep(1)

    fe = time.time() - begin
    print("又回到了foo函数",fe)

def bar():
    bs = time.time() - begin
    print("bar函数开始运行",bs)

    gevent.sleep(3)

    be = time.time() - begin
    print("又回到了bar函数",be)


gevent.joinall([
    gevent.spawn(foo),
    gevent.spawn(bar),
])
```

注意输出结果：几乎在同一时间执行两个函数（顺序和joinall方法中注册顺序有关），在我们设定的sleep时间后去继续执行函数

foo函数开始运行 0.0060002803802490234

```
bar函数开始运行 0.0060002803802490234
又回到了foo函数 1.0060575008392334
又回到了bar函数 3.006171941757202
```



gevent.joinall([
    gevent.spawn(foo),
    gevent.spawn(bar),
])
按照joinall方法中注册的函数顺序，去执行他们
foo
bar
根据gevent.sleep(秒数)来决定这一块的休眠时间，这里是1秒
foo
bar
休眠相应时间后去执行下面的操作
gevent.sleep(1)
gevent.sleep(3)

所以说对于最上面简单使用中的执行顺序先是根据joinall的注册顺序去打印

foo函数开始运行

bar函数开始运行

然后由于sleep(0)间隔是0，所以立即去执行下面的打印程序（当sleep的时间是一致时，顺序还是和注册时一致）

又回到了foo函数

又回到了bar函数

（3）使用time.sleep()去更加深刻了解协程

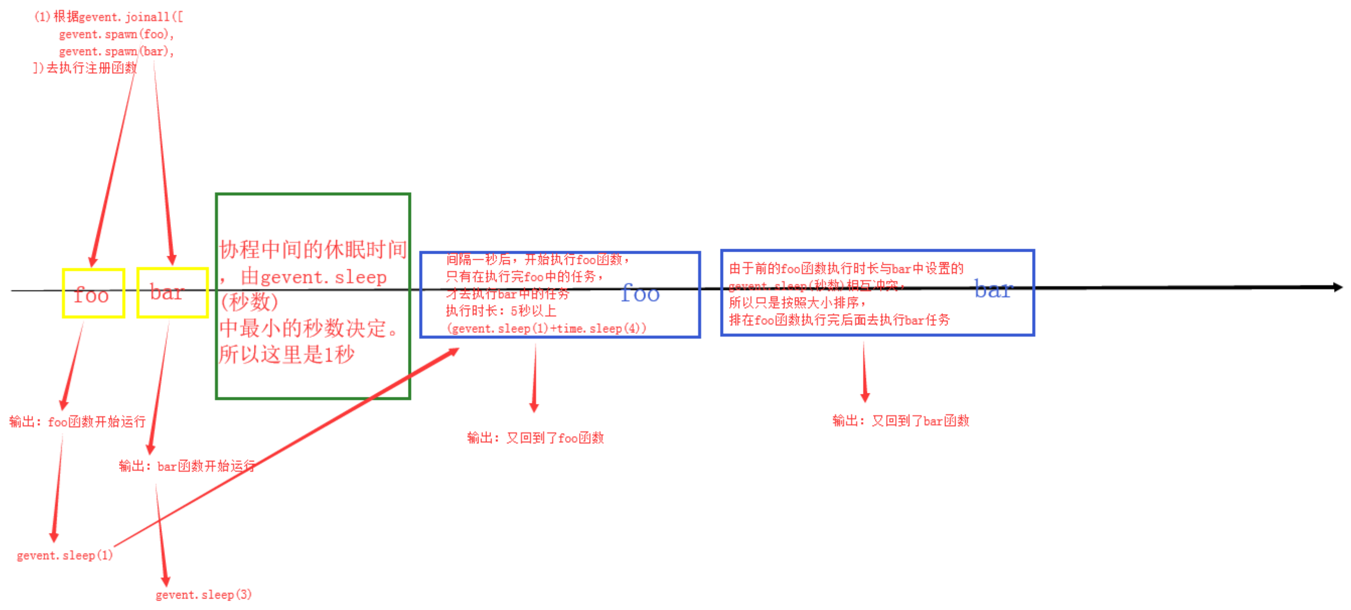
Python 复制代码

```python
import gevent
import time

begin = time.time()

def foo():
    fs = time.time() - begin
    print("foo函数开始运行",fs)

    gevent.sleep(1)

    time.sleep(4)    #这里睡眠4秒

    fe = time.time() - begin
    print("又回到了foo函数",fe)

def bar():
    bs = time.time() - begin
    print("bar函数开始运行",bs)

    gevent.sleep(3)

    be = time.time() - begin
    print("又回到了bar函数",be)


gevent.joinall([
    gevent.spawn(foo),
    gevent.spawn(bar),
])
```

注意输出结果：发现对于我们在foo中设置的time.sleep(4)对bar方法也有影响。

```
foo函数开始运行 0.005000114440917969
bar函数开始运行 0.0060002803802490234
又回到了foo函数 5.006286144256592
又回到了bar函数 5.007286310195923
```

原因：gevent设置了我们协程的苏醒时间，但是当苏醒时间与我们的执行时间相冲突，那么会以执行时间为主（毕竟这是单线程，不会考虑其他的），而原来的设置的gevent.sleep(秒数)则变成了大小比较，谁在后，谁就后执行



任务框架：

Python 复制代码

```python
import gevent
import time

begin = time.time()

def foo(url,index):
    fs = time.time() - begin
    print("%s:发送请求到%s, 等待返回"%(index,url),fs) #这里可以模拟发送请求
    gevent.sleep(0)
    fe = time.time() - begin
    print("%s:获取信息从%s, 开始处理"%(index,url),fe) #这里模拟处理信息

gevent.joinall([
    gevent.spawn(foo,"www.baidu.com",1),    #注意传参方式
    gevent.spawn(foo,"www.sina.com.sn",2),
])
```

输出结果：

```
1:发送请求到www.baidu.com，等待返回 0.005000114440917969
2:发送请求到www.sina.com.sn，等待返回 0.005000114440917969
1:获取信息从www.baidu.com，开始处理 0.005000114440917969
2:获取信息从www.sina.com.sn，开始处理 0.005000114440917969
```

补充：greenlet协程（gevent是基于greenlet实现，所以有必要去了解下）

Python 复制代码

```python
from greenlet import greenlet

def foo():
    print("开始执行foo")
    gr2.switch()
    print("又回到foo")
    gr2.switch()

def bar():
    print("开始执行bar")
    gr1.switch()
    print("又回到bar")

gr1 = greenlet(foo)
gr2 = greenlet(bar)
gr1.switch()     #以gr1开始执行，switch中也可以传递参数
```

输出结果：

开始执行foo

开始执行bar

又回到foo

又回到bar

（三）全部代码实现

Python 复制代码

```python
from operator import attrgetter

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

class MyMonitor(simple_switch_13.SimpleSwitch13):    #simple_switch_13 is same as the last experiment which named self_learn_switch
    '''
    design a class to achvie managing the quantity of flow
    '''

    def __init__(self,*args,**kwargs):
        super(MyMonitor,self).__init__(*args,**kwargs)
        self.datapaths = {}
        #use gevent to start monitor
        self.monitor_thread = hub.spawn(self._monitor)

    @set_ev_cls(ofp_event.EventOFPStateChange,[MAIN_DISPATCHER,DEAD_DISPATCHER])
    def _state_change_handler(self,ev):
        '''
        design a handler to get switch state transition condition
        '''
        #first get ofprocotol info
        datapath = ev.datapath
        ofproto = datapath.ofproto
        ofp_parser = datapath.ofproto_parser

        #judge datapath`s status to decide how to operate
        if datapath.state == MAIN_DISPATCHER:    #should save info to dictation
            if datapath.id not in self.datapaths:
                self.datapaths[datapath.id] = datapath
                self.logger.debug("Regist datapath: %16x",datapath.id)
        elif datapath.state == DEAD_DISPATCHER:    #should remove info from dictation
            if datapath.id in self.datapaths:
                del self.datapaths[datapath.id]
                self.logger.debug("Unregist datapath: %16x",datapath.id)


    def _monitor(self):
        '''
        design a monitor on timing system to request switch infomations about port and flow
        '''
        while True:    #initiatie to request port and flow info all the time
            for dp in self.datapaths.values():
                self._request_stats(dp)
            hub.sleep(5)    #pause to sleep to wait reply, and gave time to other gevent to request

    def _request_stats(self,datapath):
        '''
        the function is to send requery to datapath
        '''
        self.logger.debug("send stats reques to datapath: %16x for port and flow info",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.EventOFPPortStatsReply,MAIN_DISPATCHER)
    def _port_stats_reply_handler(self,ev):
        '''
        monitor to require the port state, then this function is to get infomation for port`s info
        print("6666666666port info:")
        print(ev.msg)
        print(dir(ev.msg))
        '''
        body = ev.msg.body
        self.logger.info('datapath             port     '
                        'rx_packets            tx_packets'
                        'rx_bytes            tx_bytes'
                        'rx_errors            tx_errors'
                        )
        self.logger.info('---------------    --------'
                        '--------    --------'
                        '--------    --------'
                        '--------    --------'
                        )
        for port_stat in sorted(body,key=attrgetter('port_no')):
                self.logger.info('%016x %8x %8d %8d %8d %8d %8d %8d',
                    ev.msg.datapath.id,port_stat.port_no,port_stat.rx_packets,port_stat.tx_packets,
                    port_stat.rx_bytes,port_stat.tx_bytes,port_stat.rx_errors,port_stat.tx_errors
                        )


    @set_ev_cls(ofp_event.EventOFPFlowStatsReply,MAIN_DISPATCHER)
    def _flow_stats_reply_handler(self,ev):
        '''
        monitor to require the flow state, then this function is to get infomation for flow`s info
        print("777777777flow info:")
        print(ev.msg)
        print(dir(ev.msg))
        '''
        body = ev.msg.body

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

补充：注意---每个事件的属性可能不同，需要我们进行Debug，例如上面就出现了ev.msg.body（之前hub实现中没有）

（四）代码讲解

1.class MyMonitor(simple_switch_13.SimpleSwitch13):

simple_switch_13.SimpleSwitch13是样例代码，其中实现了和我们上一次实验中，自学习交换机类似的功能（稍微多了个关于交换机是否上传全部packet还是只上传buffer_id），所以我们直接继承，可以减少写代码时间

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

```python
    def __init__(self,*args,**kwargs):
        super(MyMonitor,self).__init__(*args,**kwargs)
        self.datapaths = {}
        #use gevent to start monitor
        self.monitor_thread = hub.spawn(self._monitor)
```

3.在协程中实现周期请求交换机信息

```python
    def _monitor(self):
        '''
        design a monitor on timing system to request switch infomations about port and flow
        '''
        while True:    #initiatie to request port and flow info all the time
            for dp in self.datapaths.values():
                self._request_stats(dp)
            hub.sleep(5)    #pause to sleep to wait reply, and gave time to other gevent to request
```

4.主动下发消息，请求交换机信息OFPFlowStatsRequest------注意：我们这里请求两个（端口和协议信息），所以我们要使用两个函数来分别处理port和flow响应

Python 复制代码

```python
def _request_stats(self,datapath):
        '''
        the function is to send requery to datapath
        '''
        self.logger.debug("send stats reques to datapath: %16x for port and flow info",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) #可以向上面一样省略默认参数
        datapath.send_msg(req)
```

源码查看参数

```python
@_set_stats_type(ofproto.OFPMP_FLOW, OFPFlowStats)
@_set_msg_type(ofproto.OFPT_MULTIPART_REQUEST)
class OFPFlowStatsRequest(OFPFlowStatsRequestBase):
    """
    Individual flow statistics request message

    The controller uses this message to query individual flow statistics.

    ================ ======================================================
    Attribute        Description
    ================ ======================================================
    flags            Zero or ``OFPMPF_REQ_MORE``
    table_id         ID of table to read
    out_port         Require matching entries to include this as an output
                     port
    out_group        Require matching entries to include this as an output
                     group
    cookie           Require matching entries to contain this cookie value
    cookie_mask      Mask used to restrict the cookie bits that must match
    match            Instance of ``OFPMatch``
    ================ ======================================================

    Example::

        def send_flow_stats_request(self, datapath):
            ofp = datapath.ofproto
            ofp_parser = datapath.ofproto_parser

            cookie = cookie_mask = 0
            match = ofp_parser.OFPMatch(in_port=1)
            req = ofp_parser.OFPFlowStatsRequest(datapath, 0,
                                                 ofp.OFPTT_ALL,
                                                 ofp.OFPP_ANY, ofp.OFPG_ANY,
                                                 cookie, cookie_mask,
                                                 match)
            datapath.send_msg(req)
    """
```

5.获取端口响应信息ofp_event.EventOFPPortStatsReply

```python
@set_ev_cls(ofp_event.EventOFPPortStatsReply,MAIN_DISPATCHER)
    def _port_stats_reply_handler(self,ev):
        '''
        monitor to require the port state, then this function is to get infomation for port`s info
        print("6666666666port info:")
        print(ev.msg)
        print(dir(ev.msg))
        '''
        body = ev.msg.body
        self.logger.info('datapath             port     '
                        'rx_packets            tx_packets'
                        'rx_bytes            tx_bytes'
                        'rx_errors            tx_errors'
                        )
        self.logger.info('---------------    --------'
                        '--------    --------'
                        '--------    --------'
                        '--------    --------'
                        )
        for port_stat in sorted(body,key=attrgetter('port_no')):
                self.logger.info('%016x %8x %8d %8d %8d %8d %8d %8d',
                    ev.msg.datapath.id,port_stat.port_no,port_stat.rx_packets,port_stat.tx_packets,
                    port_stat.rx_bytes,port_stat.tx_bytes,port_stat.rx_errors,port_stat.tx_errors
                        )
```

端口信息：《参考》

```python
6666666666port info:
version=0x4,msg_type=0x13,msg_len=0x1d0,xid=0x8dcd9187,
OFPPortStatsReply(
    body=[
OFPPortStats(port_no=4294967294,rx_packets=0,tx_packets=0,rx_bytes=0,tx_bytes=0,rx_dropped=65,tx_dropped=0,rx_errors=0,tx_errors=0,rx_frame_err=0,rx_over_err=0,rx_crc_err=0,collisions=0,duration_sec=1912,duration_nsec=331000000), OFPPortStats(port_no=1,rx_packets=154,tx_packets=225,rx_bytes=11660,tx_bytes=19503,rx_dropped=0,tx_dropped=0,rx_errors=0,tx_errors=0,rx_frame_err=0,rx_over_err=0,rx_crc_err=0,collisions=0,duration_sec=1912,duration_nsec=333000000), OFPPortStats(port_no=2,rx_packets=186,tx_packets=257,rx_bytes=14516,tx_bytes=22343,rx_dropped=0,tx_dropped=0,rx_errors=0,tx_errors=0,rx_frame_err=0,rx_over_err=0,rx_crc_err=0,collisions=0,duration_sec=1912,duration_nsec=334000000), OFPPortStats(port_no=3,rx_packets=220,tx_packets=232,rx_bytes=18439,tx_bytes=19311,rx_dropped=0,tx_dropped=0,rx_errors=0,tx_errors=0,rx_frame_err=0,rx_over_err=0,rx_crc_err=0,collisions=0,duration_sec=1912,duration_nsec=333000000)
]
,flags=0,type=4)


OFPPortStats(
```

6.获取flow协议响应信息ofp_event.EventOFPFlowStatsReply

Python 复制代码

```python
@set_ev_cls(ofp_event.EventOFPFlowStatsReply,MAIN_DISPATCHER)
    def _flow_stats_reply_handler(self,ev):
        '''
        monitor to require the flow state, then this function is to get infomation for flow`s info
        print("777777777flow info:")
        print(ev.msg)
        print(dir(ev.msg))
        '''
        body = ev.msg.body

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

协议信息《参考》

Python 复制代码

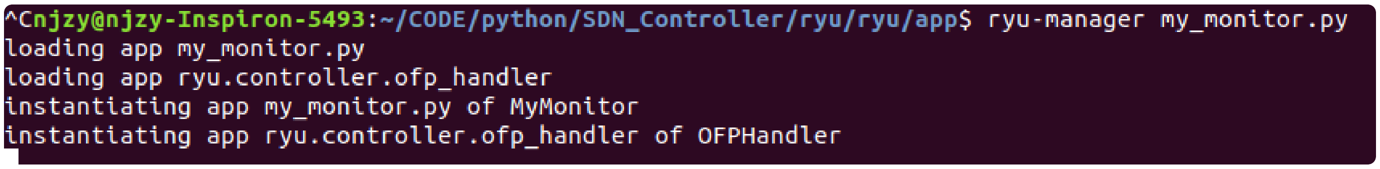
```python
777777777flow info:
version=0x4,msg_type=0x13,msg_len=0x200,xid=0x9e448a1a,
OFPFlowStatsReply(
    body=[
        OFPFlowStats(byte_count=5446,cookie=0,duration_nsec=552000000,duration_sec=1893,flags=0,hard_timeout=0,idle_timeout=0,instructions=[OFPInstructionActions(actions=[OFPActionOutput(len=16,max_len=65509,port=1,type=0)],len=24,type=4)],
        length=104,match=OFPMatch(oxm_fields={'in_port': 2, 'eth_src': '8a:06:6a:2c:10:fc', 'eth_dst': '26:20:2f:85:5a:9a'}),packet_count=71,priority=1,table_id=0),     OFPFlowStats(byte_count=5348,cookie=0,duration_nsec=549000000,duration_sec=1893,flags=0,hard_timeout=0,idle_timeout=0,instructions=[OFPInstructionActions(actions=[OFPActionOutput(len=16,max_len=65509,port=2,type=0)],len=24,type=4)],
        length=104,match=OFPMatch(oxm_fields={'in_port': 1, 'eth_src': '26:20:2f:85:5a:9a', 'eth_dst': '8a:06:6a:2c:10:fc'}),packet_count=70,priority=1,table_id=0), OFPFlowStats(byte_count=8302,cookie=0,duration_nsec=438000000,duration_sec=1887,flags=0,hard_timeout=0,idle_timeout=0,instructions=[OFPInstructionActions(actions=[OFPActionOutput(len=16,max_len=65509,port=1,type=0)],len=24,type=4)],
        length=104,match=OFPMatch(oxm_fields={'in_port': 2, 'eth_src': 'ca:9e:a1:af:b9:5f', 'eth_dst': '26:20:2f:85:5a:9a'}),packet_count=103,priority=1,table_id=0), OFPFlowStats(byte_count=8204,cookie=0,duration_nsec=436000000,duration_sec=1887,flags=0,hard_timeout=0,idle_timeout=0,instructions=[OFPInstructionActions(actions=[OFPActionOutput(len=16,max_len=65509,port=2,type=0)],len=24,type=4)]
        ,length=104,match=OFPMatch(oxm_fields={'in_port': 1, 'eth_src': '26:20:2f:85:5a:9a', 'eth_dst': 'ca:9e:a1:af:b9:5f'}),packet_count=102,priority=1,table_id=0), OFPFlowStats(byte_count=6739,cookie=0,duration_nsec=807000000,duration_sec=9,flags=0,hard_timeout=0,idle_timeout=0,instructions=[OFPInstructionActions(actions=[OFPActionOutput(len=16,max_len=65535,port=4294967293,type=0)],len=24,type=4)],
        length=80,match=OFPMatch(oxm_fields={}),packet_count=74,priority=0,table_id=0)
]
,flags=0,type=1)


OFPFlowStats(
byte_count=5446,                ----------
cookie=0,
duration_nsec=552000000,
duration_sec=1893,
flags=0,
hard_timeout=0,
idle_timeout=0,
instructions=[
    OFPInstructionActions(
        actions=[
            OFPActionOutput(
                len=16,
                max_len=65509,
                port=1,          -----------
                type=0)
            ],
            len=24,
            type=4
    )
],
length=104,
match=OFPMatch(oxm_fields={
    'in_port': 2,                    -----------
    'eth_src': '8a:06:6a:2c:10:fc',      -----------
    'eth_dst': '26:20:2f:85:5a:9a'        -----------
}),
packet_count=71,                 -----------
priority=1,
table_id=0
)

['_STATS_MSG_TYPES', '_TYPE', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', '_base_attributes', '_class_prefixes', '_class_suffixes', '_decode_value', '_encode_value', '_get_decoder', '_get_default_decoder', '_get_default_encoder', '_get_encoder', '_get_type', '_is_class', '_opt_attributes', '_restore_args', '_serialize_body', '_serialize_header', '_serialize_pre', 'body', 'buf', 'cls_body_single_struct', 'cls_from_jsondict_key', 'cls_msg_type', 'cls_stats_body_cls', 'cls_stats_type',
'datapath'                    -----------
, 'flags', 'from_jsondict', 'msg_len', 'msg_type', 'obj_from_jsondict', 'parser', 'parser_stats', 'parser_stats_body', 'register_stats_type', 'serialize', 'set_buf', 'set_classes', 'set_headers', 'set_xid', 'stringify_attrs', 'to_jsondict', 'type', 'version', 'xid']
```

三：实验演示

（一）开启Ryu

Shell 复制代码

```shell
ryu-manager my_monitor.py
```

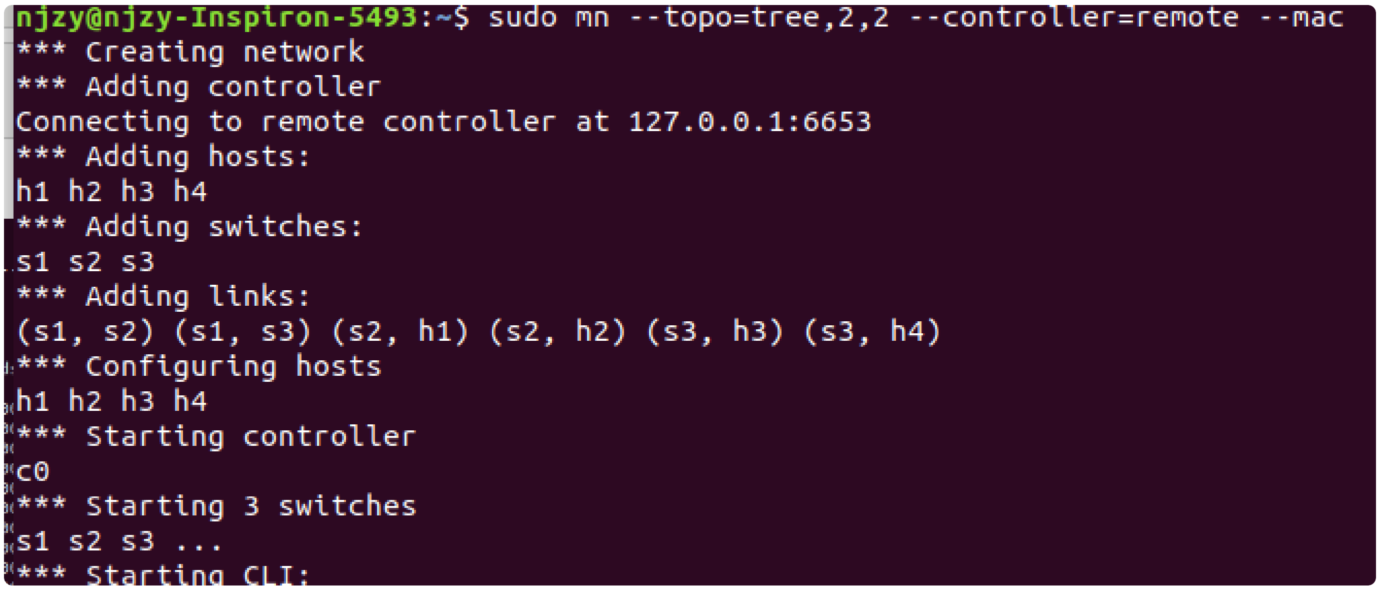


```
^Cnjzy@njzy-Inspiron-5493:~/CODE/python/SDN_Controller/ryu/ryu/app$ ryu-manager my_monitor.py
loading app my_monitor.py
loading app ryu.controller.ofp_handler
instantiating app my_monitor.py of MyMonitor
instantiating app ryu.controller.ofp_handler of OFPHandler
```

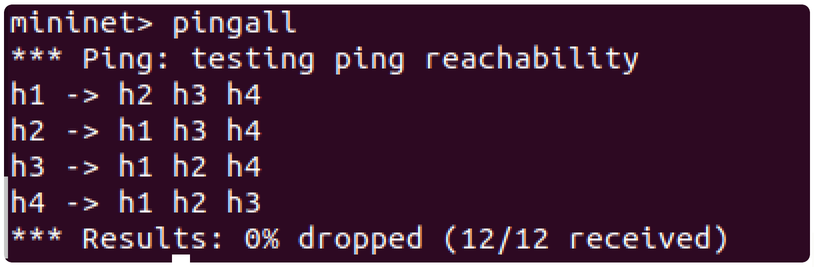
（二）开启Mininet

Shell 复制代码

```shell
sudo mn --topo=tree,2,2 --controller=remote --mac
```

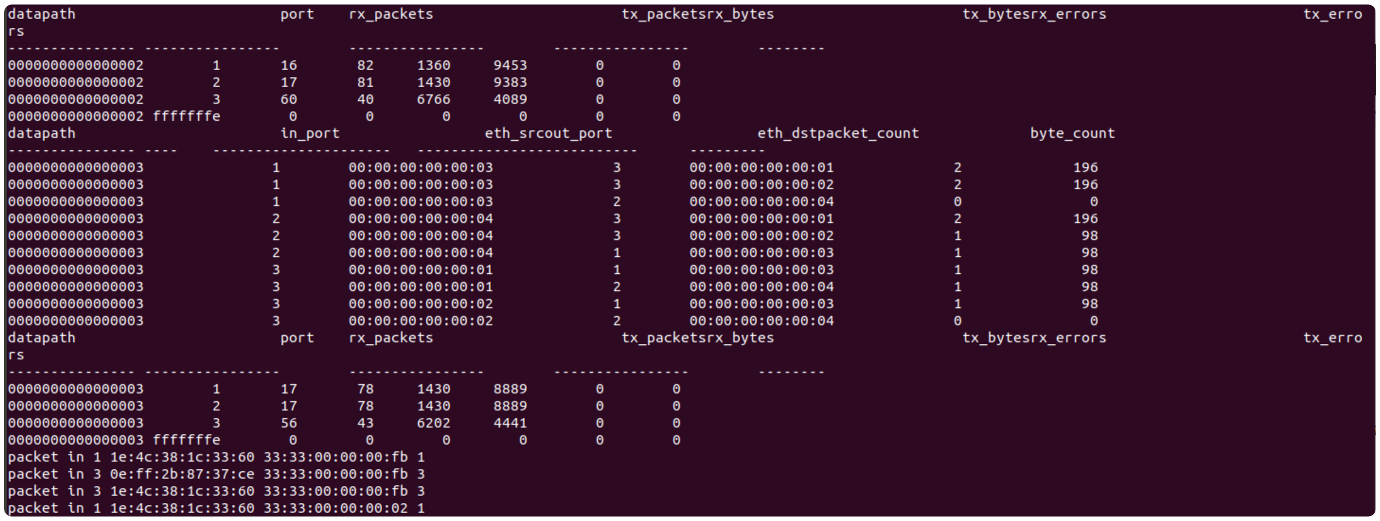


```
njzy@njzy-Inspiron-5493:~$ sudo mn --topo=tree,2,2 --controller=remote --mac
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3
*** Adding links:
(s1, s2) (s1, s3) (s2, h1) (s2, h2) (s3, h3) (s3, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
```



```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

（三）Ryu显示结果



```
datapath                        port    rx_packets                      tx_packetsrx_bytes                      tx_bytesrx_errors                       tx_erro
rs
--------------- ----------------        ----------------        ----------------        --------
0000000000000002        1       16       82     1360     9453        0        0
0000000000000002        2       17       81     1430     9383        0        0
0000000000000002        3       60       40     6766     4089        0        0
0000000000000002 fffffffe        0        0        0        0        0        0
datapath                        in_port                 eth_srcout_port                 eth_dstpacket_count             byte_count
--------------- ----    ---------------------   --------------------------      ---------
0000000000000003               1        00:00:00:00:00:03              3        00:00:00:00:00:01              2             196
0000000000000003               1        00:00:00:00:00:03              3        00:00:00:00:00:02              2             196
0000000000000003               1        00:00:00:00:00:03              2        00:00:00:00:00:04              0               0
0000000000000003               2        00:00:00:00:00:04              3        00:00:00:00:00:01              2             196
0000000000000003               2        00:00:00:00:00:04              3        00:00:00:00:00:02              1              98
0000000000000003               2        00:00:00:00:00:04              1        00:00:00:00:00:03              1              98
0000000000000003               3        00:00:00:00:00:01              1        00:00:00:00:00:03              1              98
0000000000000003               3        00:00:00:00:00:01              2        00:00:00:00:00:04              1              98
0000000000000003               3        00:00:00:00:00:02              1        00:00:00:00:00:03              1              98
0000000000000003               3        00:00:00:00:00:02              2        00:00:00:00:00:04              0               0
datapath                        port    rx_packets                      tx_packetsrx_bytes                      tx_bytesrx_errors                       tx_erro
rs
--------------- ----------------        ----------------        ----------------        --------
0000000000000003        1       17       78     1430     8889        0        0
0000000000000003        2       17       78     1430     8889        0        0
0000000000000003        3       56       43     6202     4441        0        0
0000000000000003 fffffffe        0        0        0        0        0        0
packet in 1 1e:4c:38:1c:33:60 33:33:00:00:00:fb 1
packet in 3 0e:ff:2b:87:37:ce 33:33:00:00:00:fb 3
packet in 3 1e:4c:38:1c:33:60 33:33:00:00:00:fb 3
packet in 1 1e:4c:38:1c:33:60 33:33:00:00:00:02 1
```

（四）还需要去了解返回的字段含义才可以