**ryu毕业设计**

**毕设任务**1 基于SDN实现流量监控

（1）用ryu搭建控制器

（2）用mininet搭建SDN网络，与（1）连通

（3）基于ryu进行二次开发，实现流量监视

（4）流量监视改进：流量监测数据存入文件（或数据库），根据数据画出流量曲线。

（5）流量监视改进：开发一个方便用户监测流量的web界面。希望在该web界面上，能查询mininet那边的网络有几个交换机，有几个接口。我可以选择某个交换机的某个接口，进行流量监测，画出流量曲线。

（6）通过qos规则的方案进行流量控制，对某条流实施限速。通过流量监视曲线可以看到这种限速。

**毕设任务2 基于SDN实现路由算法**

（1）用ryu搭建控制器

（2）用mininet搭建SDN网络，与（1）连通

（3）基于ryu进行二次开发，实现最短路径转发

（4）基于ryu进行二次开发，实现基于时延的最短路径转发

（5）基于ryu进行二次开发，实现带宽敏感的k-最短路径转发（放弃）

**B站参考资料**

---------------------------------------------------

**第一关 理论储备 SDN和OpenFlow**

---------------------------------------------------

资源：bilibili，讲的不是很好，可以倍速了解，＋百度扩展。

**1 SDN**

[1.1 SDN是什么\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1ft4y1a7ip?p=1&vd_source=7a86919b6781481d051e5748ad066889)

SDN已经到了商业落地阶段

[2.1 SDN典型应用\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1ft4y1a7ip?p=2&spm_id_from=pageDriver&vd_source=7a86919b6781481d051e5748ad066889)

一是云数据中心网络，SD-DCN

二是WAN，即SD-WAN

**2 OpenFlow switch**

[1.1 openflow概述\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1ft4y1a7ip?p=4&spm_id_from=pageDriver&vd_source=7a86919b6781481d051e5748ad066889)

[2.1 openflow通用抽象上\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1ft4y1a7ip?p=5&vd_source=7a86919b6781481d051e5748ad066889)

[2.2 openflow通用抽象下\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1ft4y1a7ip?p=6&spm_id_from=pageDriver&vd_source=7a86919b6781481d051e5748ad066889)

openflow switch。了解一些基本概念，例如流表，meter表

**3 OpenFlow 协议**

[3.1 openflow协议上\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1ft4y1a7ip?p=7&vd_source=7a86919b6781481d051e5748ad066889)

[3.2 openflow协议下\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1ft4y1a7ip?p=8&vd_source=7a86919b6781481d051e5748ad066889)

openflow protocol。了解控制器和交换机之间的常用信息交互，例如flow\_mod消息，用来配置流表。

[4.1 从openflow到p4\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1ft4y1a7ip?p=9&spm_id_from=pageDriver&vd_source=7a86919b6781481d051e5748ad066889)

[5.1 openflow演示实验\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1ft4y1a7ip/?p=10&spm_id_from=pageDriver&vd_source=7a86919b6781481d051e5748ad066889)

用wireshark抓包，熟悉openflow protocol。

要求：关于SDN和OpenFlow，形成介绍文档，PPT，录一个简洁易懂的SDN和OpenFlow介绍视频。

--------------------------------------------------------

**第二关 Mininet+ryu搭建一个SDN网络**

--------------------------------------------------------

资源：bilibili，《高级网络通信原理实战》

[1.1 Mininet是什么\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1ft4y1a7ip/?p=12&spm_id_from=pageDriver&vd_source=7a86919b6781481d051e5748ad066889)

《高级网络通信原理实战-1001》

第10章 SDN环境搭建

包含vmware,ubuntu,mininet,和opendaylight

[2.1 Mininet安装部署\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1ft4y1a7ip/?p=13&spm_id_from=pageDriver&vd_source=7a86919b6781481d051e5748ad066889)

[3.1 Mininet命令详解\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1ft4y1a7ip/?p=14&spm_id_from=pageDriver&vd_source=7a86919b6781481d051e5748ad066889)

[4.1 Mininet可视化\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1ft4y1a7ip/?p=15&spm_id_from=pageDriver&vd_source=7a86919b6781481d051e5748ad066889)

[5.1 Mininet中流表应用实战--玩转流表\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1ft4y1a7ip/?p=16&spm_id_from=pageDriver&vd_source=7a86919b6781481d051e5748ad066889)

[1.1 Ryu的安装部署\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1ft4y1a7ip?p=19&vd_source=7a86919b6781481d051e5748ad066889)

这一节提前在这里。

[6.1 模拟多数据中心带宽实验\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1ft4y1a7ip/?p=17&spm_id_from=pageDriver&vd_source=7a86919b6781481d051e5748ad066889)

基于mininet在多数据中心网络中实现随机流量模型

（1）自定义py命令，实现流量随机模型。

（2）自定义多数据中心网络拓扑，用opendaylight查看拓扑以验证。

（3）打流测试（因为有环路，ryu侧要用到stp）。

--------------------------------

**第三关 ryu应用开发**

--------------------------------

**第一步，开发储备**

[2.1 Hub+Learing Switch应用开发（上）\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1ft4y1a7ip?p=20&vd_source=7a86919b6781481d051e5748ad066889)

[2.2 Hub+Learing Switch应用开发（下）\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1ft4y1a7ip?p=21&vd_source=7a86919b6781481d051e5748ad066889)

[3.1 流量监控+最短路径转发\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1ft4y1a7ip?p=22&vd_source=7a86919b6781481d051e5748ad066889)

流量监控

[4.1 流量监控+最短路径转发\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1ft4y1a7ip?p=23&vd_source=7a86919b6781481d051e5748ad066889)

最短路径转发

[5.1 Ryu REST API使用教程\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1ft4y1a7ip?p=24&vd_source=7a86919b6781481d051e5748ad066889)

用curl命令和postman来测试rest api

[6.1 源码解析：Ryu运行机制分析\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1ft4y1a7ip?p=25&vd_source=7a86919b6781481d051e5748ad066889) （选看）

===============================================================================

**RYU的一些官方参考资料**

===============================================================================

RYU的github上的文档：

http://osrg.github.io/ryu-book/en/html/index.html

Example of the operation of QoS by using Meter Table

RYU的官方文档：

[Welcome to RYU the Network Operating System(NOS) — Ryu 4.34 documentation](https://ryu.readthedocs.io/en/latest/)

<https://ryu.readthedocs.io/en/latest/>

文本编辑器 sublime text 使用手册

<https://www.w3cschool.cn/sublimetext/m5si8ozt.html>

**附录1：ryu的安装**

ubuntu版本16.04，自带的python版本是2.7

进入用户目录

pip注意升级到20.2.4，如果升级到20.3会报错。

python -m pip install --user --upgrade pip==20.2.4

查看pip版本

pip –V

pip升降级后需要：

hash –r

下载ryu源码

git clone https://github.com/osrg/ryu.git

因为使用python2.7，所以ryu 不要选择太高版本的。

进入文件夹ryu

cd ryu

git tag

列出所有版本，我选择了3.26版本

git checkout v3.26

先安装依赖包

pip install -r tools/pip-requires

需要较长时间

安装ryu

python setup.py install

ryu-manager - -version

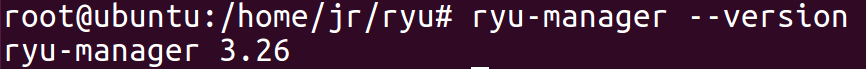
测试，报错：ImportError: cannot import name 'ALREADY\_HANDLED'

降级eventlet，安装旧的eventlet版本

pip install eventlet==0.30.2

再一次测试

ryu-manager - -version

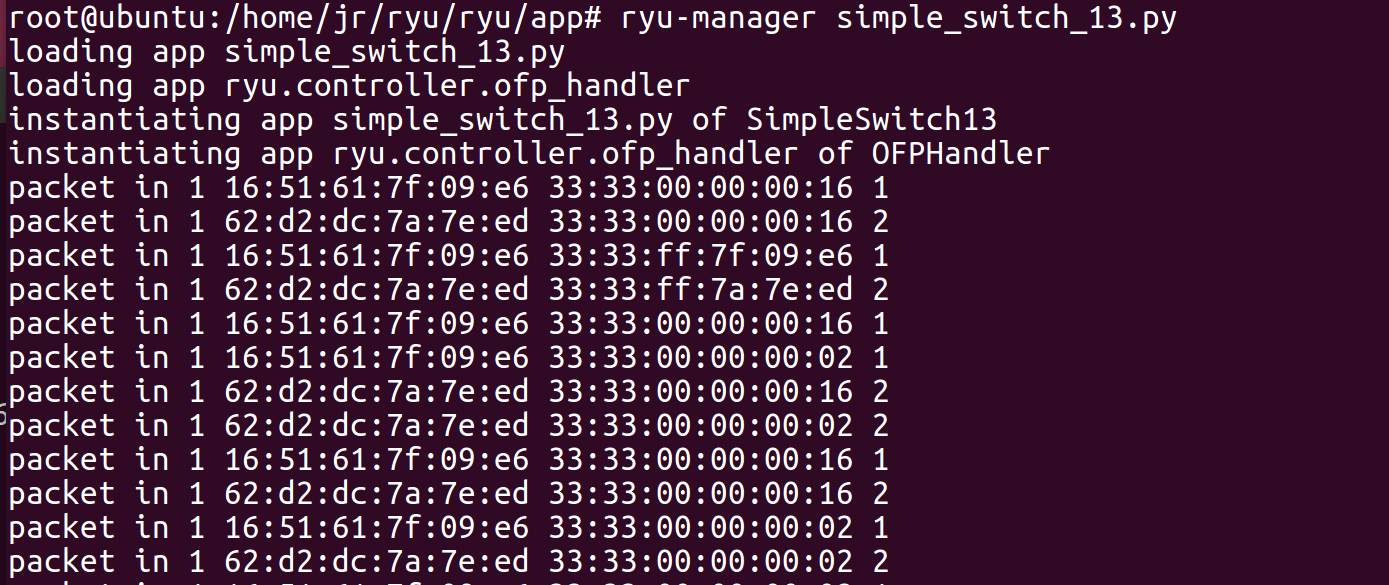


进入home/xx/ryu/ryu文件夹，里面有一个app文件夹，里面有很多示例，随便选一个执行，

 ryu-manager simple\_switch\_13.py

打开另一个terminal，ryu能和mininet连接上，有消息通信就成功了。

mn --controller=remote



ctl+c结束ryu

exit退出mininet

**附录2：流量监控的实现**

simple\_monitor\_13.py解读：类初始化时，启动监视线程，调用\_monitor()方法，调用\_request\_stats()方法，挨个向交换机查询统计信息。

当有事件返回时，打印端口统计信息。

下面做流量监控：

ryu/ryu/app下面有simple\_monitor\_13.py，实现流量监测。如果没有这个文件，可以拷贝到该目录下。

注意，如果ryu目录没有写的权限，先用chmod –R 777 ryu修改权限。

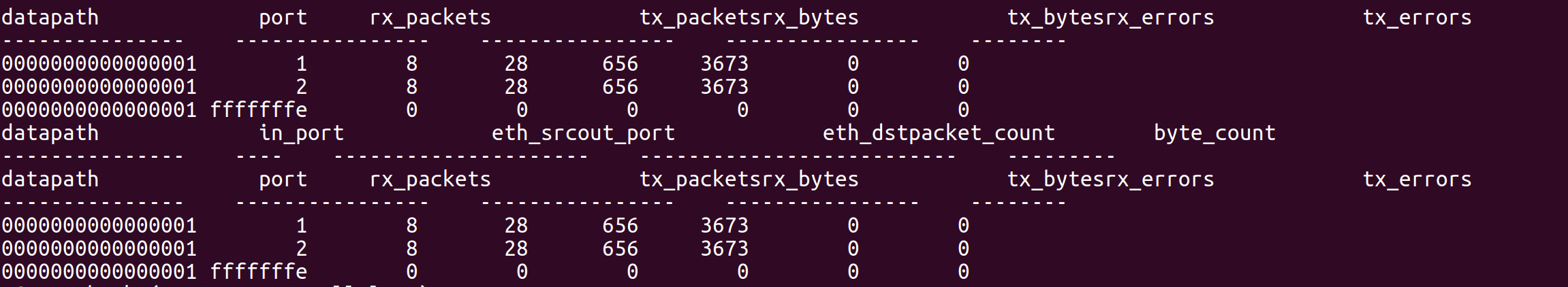
ryu侧

ryu-manager simple\_monitor\_13.py

mininet侧

mn - -controller remote

在ryu侧就可以看到流量监测信息



**附录3：流量数据存入csv文件**

[python将数据存为csv文件\_mob649e816594b7的技术博客\_51CTO博客](https://blog.51cto.com/u_16175505/7736959)

<https://blog.51cto.com/u_16175505/7736959>

将流量监控获得的数据存入.csv文件。再用plotly读csv文件绘制折线图。

改流量监控代码，首先是《my2\_monitor\_13.py》

思路：增加代码，把监测到的数据存入data.csv文件中。

\_monitor(self)函数中创建data.csv，写好表头。for循环每隔2秒去查询接口统计信息

在事件处理函数\_port\_stats\_reply\_handler(self,ev)中，在data.csv中增加新采集到的接口统计信息，同时还添加了个时间信息。

\_monitor(self)函数读取data.csv，打印。

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

#jr add

import csv

#jr add

import datetime

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

'''

#jr add

with open('data.csv', 'w') as file:

writer = csv.writer(file)

writer.writerow(['time', 'switch', 'port', 'rx\_bytes', 'tx\_bytes'])

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

#jr add

hub.sleep(2)

with open('data.csv', 'r') as file:

reader=csv.reader(file)

for row in reader:

print(row)

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

)

#jr add

current\_time = datetime.datetime.now()

print(current\_time)

#jr add

with open('data.csv', 'a') as file:

writer = csv.writer(file)

writer.writerow([current\_time,ev.msg.datapath.id,port\_stat.port\_no,port\_stat.rx\_bytes,port\_stat.tx\_bytes])

测试

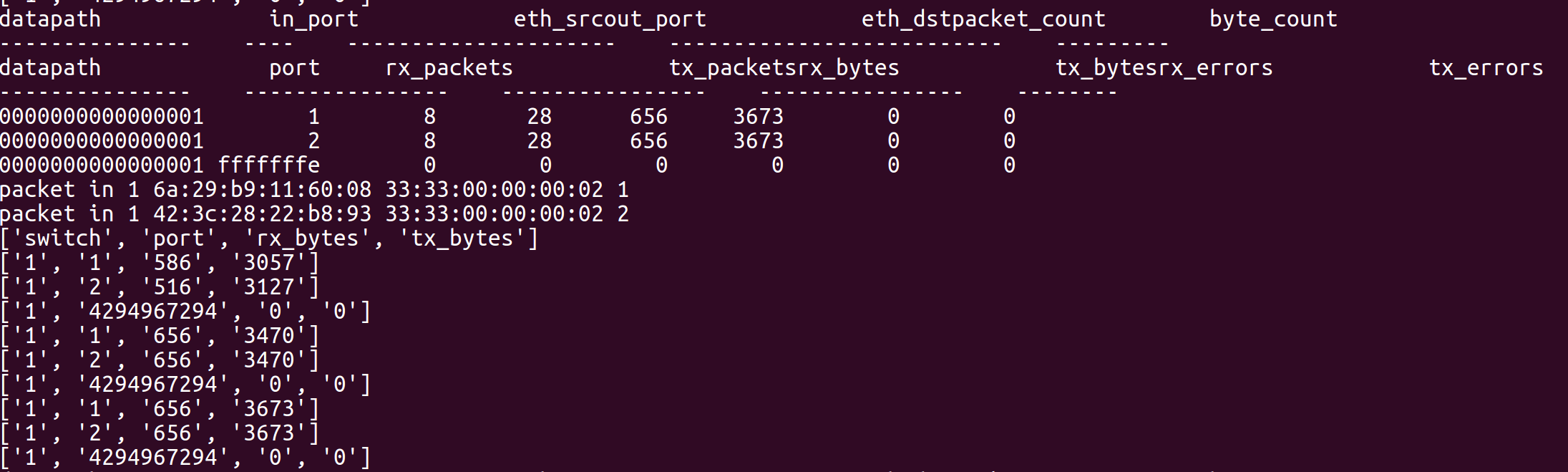
ryu侧

# ryu-manager my2\_monitor\_13.py

mininet侧，创建一个包含1个交换机，两台主机的简单网络。

# mn --controller remote

在ryu侧可以看到流量数据：



注意：不能Ping， 一ping就会导致monitor运行出现错误。

解决：不能Ping错误提示是在\_flow\_stats\_reply\_handler(self,ev)函数中，其实我们不需要流的统计信息，我们只需要端口的统计信息，所以我们把这个函数注释掉。同时，把\_request\_stats(self,datapath)中的 '''

req = parser.OFPFlowStatsRequest(datapath)

datapath.send\_msg(req)

'''

这两行也注释掉就OK了。你可以使用ping，也可以使用iperf来改变网络的流量，改变端口的速率。

在mininet中，

xterm h1

xterm h2

在h1中，

iperf –c

在h2中，

iperf –s 10.0.0.1

此时会产生比较大的流量。采集到的流量数据也可以看到变大了。

**附录4：plotly绘图**

散点图/线图

[python可视化入门 plotly ，小白请进\_plotly官方文档-CSDN博客](https://blog.csdn.net/weixin_43347550/article/details/106163458)

<https://blog.csdn.net/weixin_43347550/article/details/106163458>

Pandas是数据分析的核心工具包，它是基于Numpy构建的，可以理解为是为了数据分析而存在的。

接下去写画图代码《scatter.py》

思路，把data.csv中数据端口1的数据过滤出来放到port1\_data.csv中，并且增加两列，一列是接收和发送的bit数总和。另一列是这10秒内的bit速率，就是比特率。

参考代码：

<https://blog.csdn.net/weixin_43347550/article/details/106163458>

#import part

import plotly.graph\_objects as go

import plotly.offline as of

import pandas as pd

import csv

import numpy as np

#read data.csv

csv\_read = pd.read\_csv(r'data.csv')

#put data in array

csv\_read\_array=np.array(csv\_read)

filter\_data=[]

#if data belong port 1, filter them

for item in csv\_read\_array:

string= item[2]

if 1 == string:

filter\_data.append(item)

#put colums into lists

date=[i[0] for i in filter\_data]

switch=[i[1] for i in filter\_data]

port=[i[2] for i in filter\_data]

rx\_bytes=[i[3] for i in filter\_data]

tx\_bytes=[i[4] for i in filter\_data]

#bits data of receiving and sending

bits=[(i[3]+i[4])\*8 for i in filter\_data]

#compute bit rate

length = len(bits)

bps=[0]\*length

i = 1

while i < length:

bps[i]=(bits[i]-bits[i-1])/10

i += 1

#convert lists into csv file, port1\_data.csv

rows = zip(date,switch,port,rx\_bytes,tx\_bytes,bits,bps)

with open('port1\_data.csv', 'w') as f:

writer = csv.writer(f)

header = ['time', 'switch', 'port', 'rx\_bytes', 'tx\_bytes', 'bits','bps']

csvrow1 = []

csvrow2 = []

csvrow3 = []

csvrow4 = []

csvrow5 = []

csvrow6 = []

csvrow7 = []

csvrow1.extend('time')

csvrow2.extend('switch')

csvrow3.extend('port')

csvrow4.extend('rx\_bytes')

csvrow5.extend('tx\_bytes')

csvrow6.extend('bits')

csvrow7.extend('bps')

writer.writerow(header)

writer.writerows(rows)

data = pd.read\_csv(r'port1\_data.csv')

print(data.head())

# draw Scatter figure

#line1 = go.Scatter(y=data['bits'], x=data['time'], name='bits')

line2 = go.Scatter(y=data['bps'], x=data['time'], name='bps')

#fig = go.Figure([line1, line2])

fig = go.Figure(line2)

fig.update\_layout(

title = 'flow chart',

xaxis\_title = 'time',

yaxis\_title = 'bps'

)

of.plot(fig)

测试：

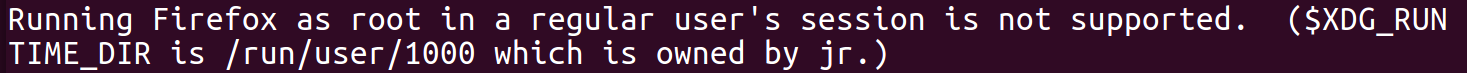
pip install plotly

pip install pandas

运行：

python scatter.py，

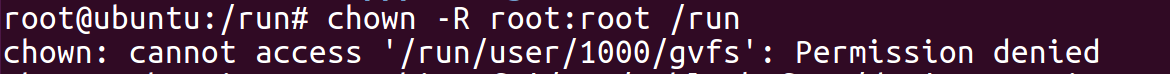
注意：报错：



chown -R root:root /run

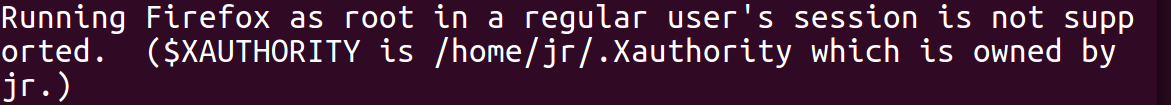
chown -R root:root /run/user

还是报错



不理会，

重新运行python scatter.py，



chown -R root:root /home/jr/.Xauthority

重新运行python scatter.py，

可以运行了。

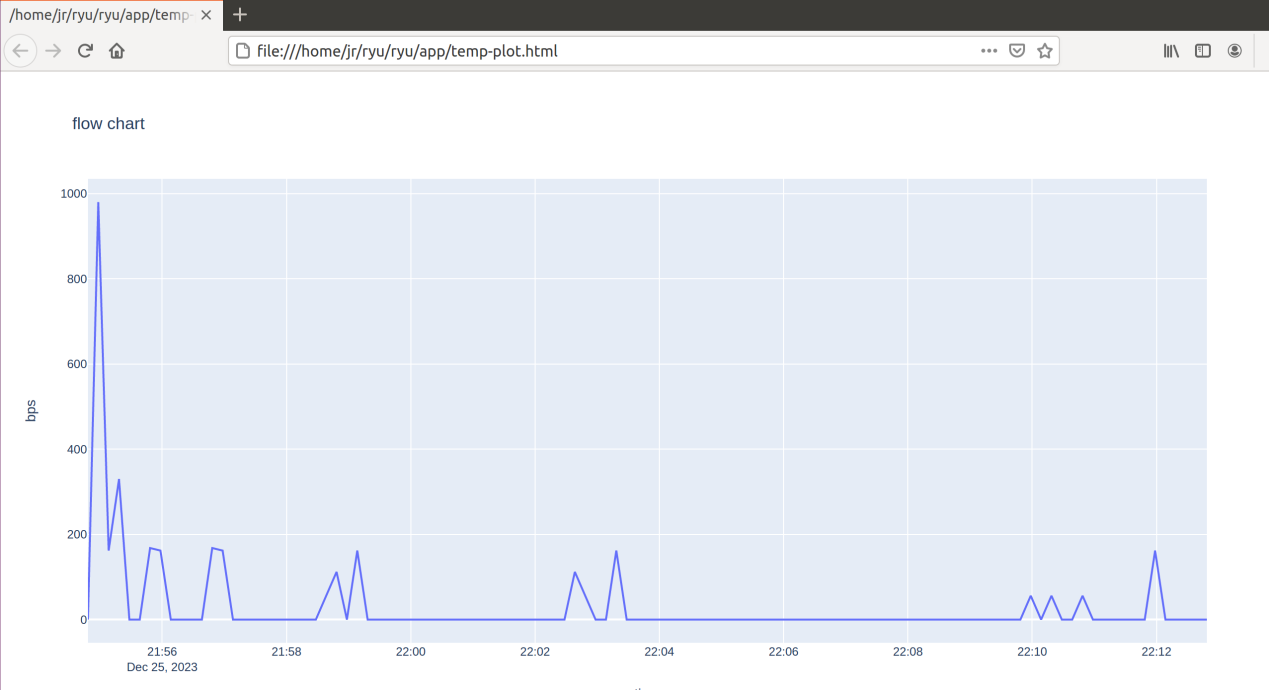
注意注意：流量监控绘图，root用户下执行scatter.py文件时，报错提示Xauthority文件权限属于本机用户，用chown -R root:root /home/{用户名}/.Xauthority 命令将权限开给root用户会导致下次开机打不开，会在登录界面循环。可以下面方法恢复。   
绘图时直接在普通用户下执行scatter.py文件即可。

如果已经root用户下执行scatter.py文件时，则参考[linux点用户输密码时就调回登录框,Linux系统登陆输入密码正确 闪回登陆界面-CSDN博客](https://blog.csdn.net/weixin_29268637/article/details/116554240?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522170374437316777224443987%2522%252C%2522scm%2522%253A%252220140713.130102334.pc%255Fall.%2522%257D&request_id=170374437316777224443987&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_ecpm_v1~rank_v31_ecpm-1-116554240-null-null.142%5ev99%5epc_search_result_base3&utm_term=linux%E7%94%A8%E6%88%B7%E8%BE%93%E5%85%A5%E5%AF%86%E7%A0%81%E9%87%8D%E6%96%B0%E8%B7%B3%E8%BD%AC%E8%BE%93%E5%85%A5%E5%AF%86%E7%A0%81%E7%95%8C%E9%9D%A2&spm=1018.2226.3001.4187)，用命令

$sudo chown jr:jr .Xauthority

改回来即可。

即可画出流量曲线图。



注意时间不准确，是因为linux系统的时间没有调准确。

**附录5：QoS**

参考：[QoS — Ryubook 1.0 documentation (osrg.github.io)](http://osrg.github.io/ryu-book/en/html/rest_qos.html)

<http://osrg.github.io/ryu-book/en/html/rest_qos.html>

本节介绍如何使用REST来设置QoS。

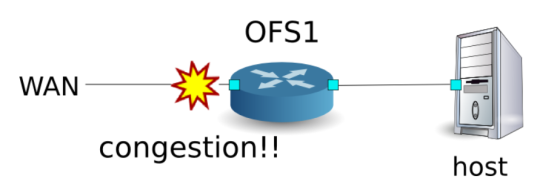
**关于QoS**

Qos能根据数据的优先级来传输数据，能为特定的通信预留固定的通信带宽。

**为流设定QoS的操作示例**

先要创建示例网络拓扑，然后添加队列和规则来预留带宽。

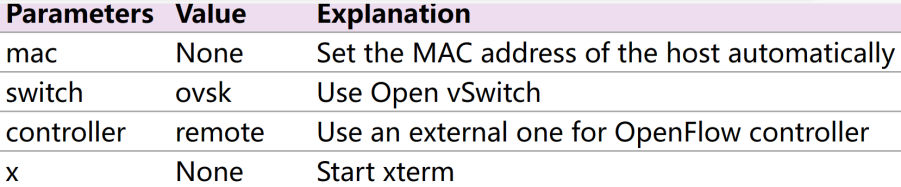
本例显示WAN侧接口OFS1的流量整形。



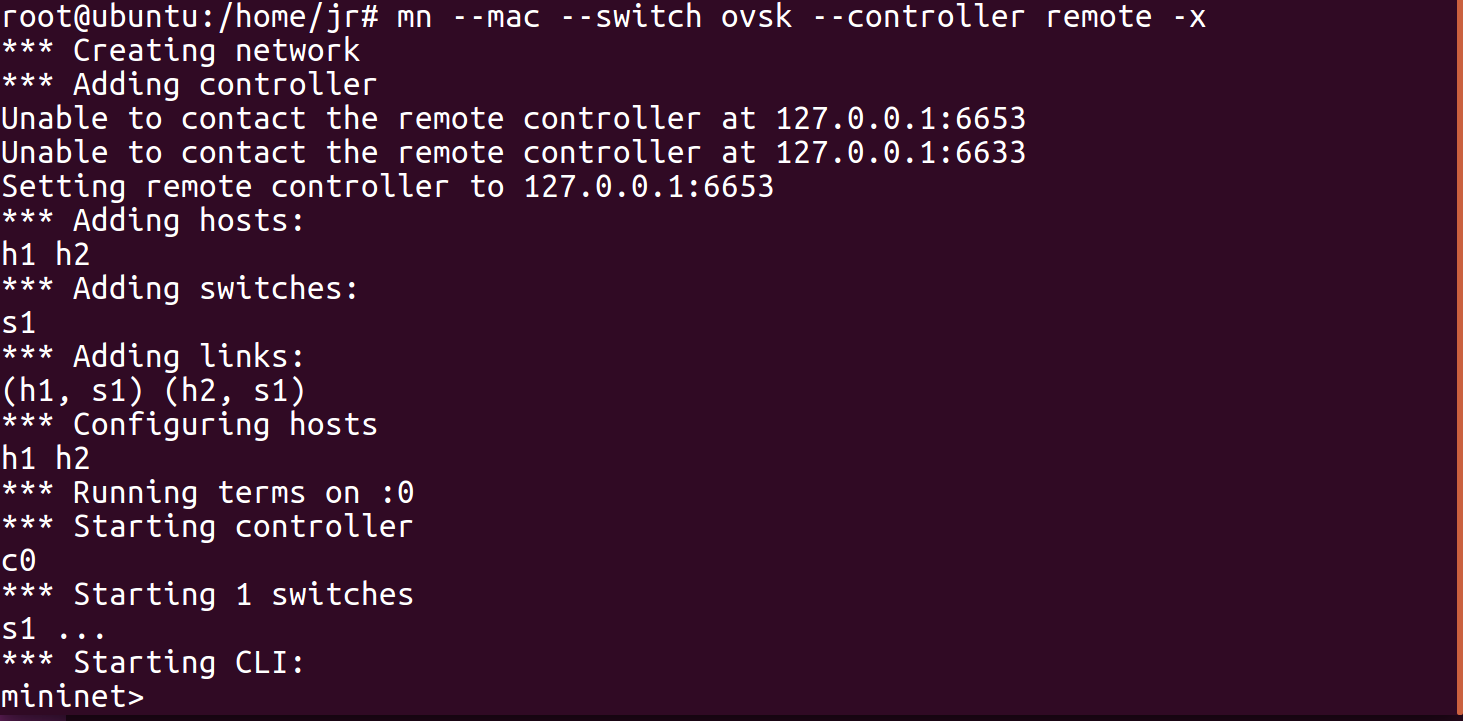
1 新启一个terminal，创建mininet网络环境。

mn --mac --switch ovsk --controller remote -x

参数的意思如下：



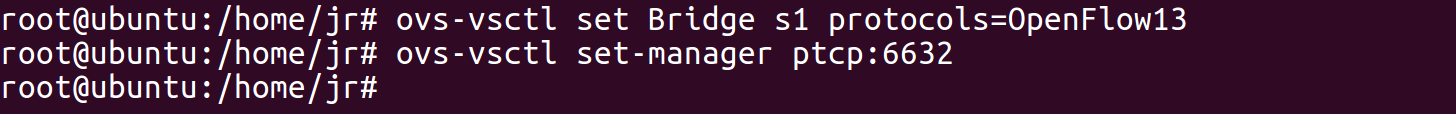
在h1终端，ping一下h2，此时是ping不通的，因为还没有连接控制器。



新启一个terminal，在交换机侧，设置openflow版本为1.3，在6632端口监听，以连接OVSDB。

*# ovs-vsctl set Bridge s1 protocols=OpenFlow13*

*# ovs-vsctl set-manager ptcp:6632*



然后，修改simple\_switch\_13.py为qos\_simple\_switch\_13.py，注册流表项到table id:1。

使用 sed 工具对文件 ryu/ryu/app/simple\_switch\_13.py 进行文本替换操作。具体来说，它在包含字符串 "OFPFlowMod(" 的行和下一个包含 ")" 的行之间进行替换操作。替换的内容是将 ")" 替换为 ", table\_id=1)"。替换后的结果被输出到文件 ryu/ryu/app/qos\_simple\_switch\_13.py 中。

然后，重新安装ryu

*# sed '/OFPFlowMod(/,/)/s/)/, table\_id=1)/' ryu/ryu/app/simple\_switch\_13.py > ryu/ryu/app/qos\_simple\_switch\_13.py*

*# cd ryu/*

*# python ./setup.py install*

2 然后，运行ryu控制器：rest\_qos, qos\_simple\_switch\_13 and rest\_conf\_switch。

*# ryu-manager rest\_qos qos\_simple\_switch\_13 rest\_conf\_switch*

现在ryu控制器已经起来了。



注意一定要先启ryu控制器，再启动mininet，否则双方连不上。

所以exit退出mininet。重新创建mininet网络拓扑。

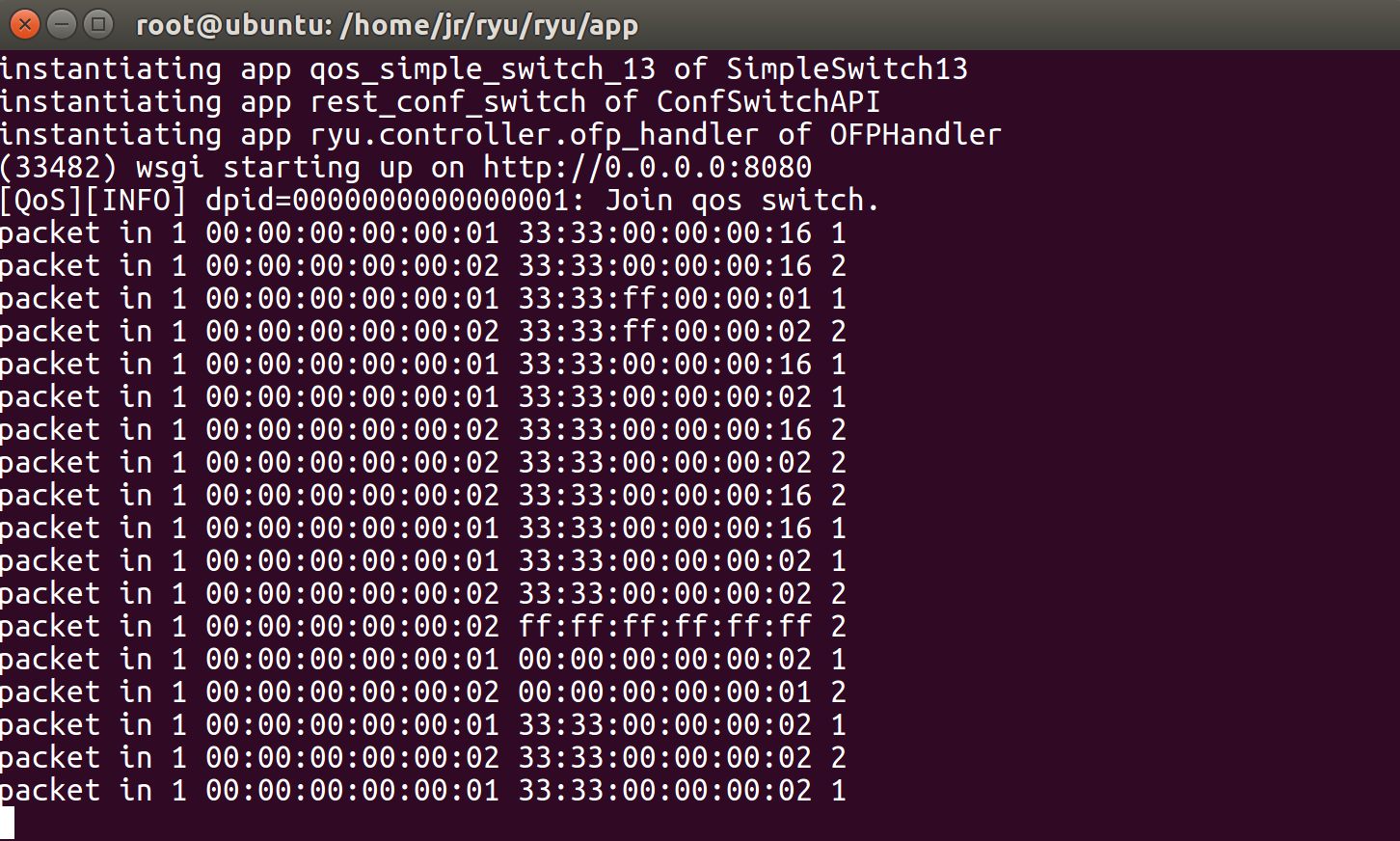
mn --mac --switch ovsk --controller remote -x

在h1终端，ping一下h2，此时可以Ping通。

可以在ryu侧查看验证ryu已经跟mininet连上了。

注意以下这句表示带qos功能的交换机s1已经连上ryu了。





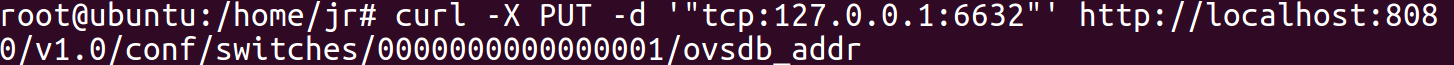
新启一个terminal，通过REST做QoS的设置。

因为后面要用curl命令，所以要先安装curl

*# apt install curl*

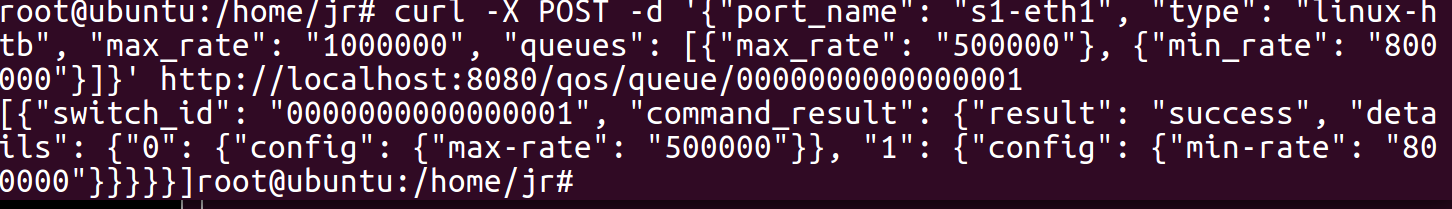
然后，设置ovsdb\_addr以连接OVSDB

*# curl -X PUT -d '"tcp:127.0.0.1:6632"' http://localhost:8080/v1.0/conf/switches/0000000000000001/ovsdb\_addr*



然后，设置队列

*# curl -X POST -d '{"port\_name": "s1-eth1", "type": "linux-htb", "max\_rate": "1000000", "queues": [{"max\_rate": "500000"}, {"min\_rate": "800000"}]}' http://localhost:8080/qos/queue/0000000000000001*



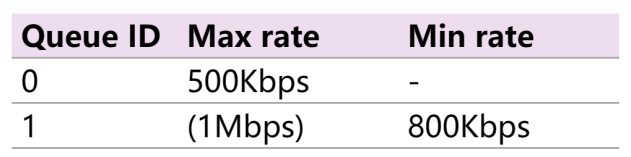
"port\_name": "s1-eth1"：要配置队列的端口名称为 s1-eth1。

"type": "linux-htb"：队列的类型为 linux-htb，表示使用 Linux HTB 调度算法。

"max\_rate": "1000000"：队列的最大速率为 1000000，表示最大带宽为 1 Mbps。

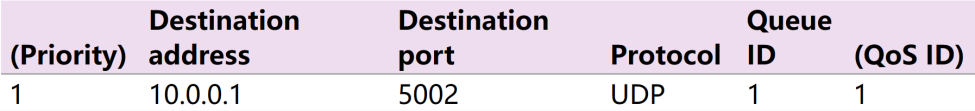
"queues": [{"max\_rate": "500000"}, {"min\_rate": "800000"}]：队列的配置包括两个子队列。第一个子队列的最大速率为 500000，第二个子队列的最小速率为 800000。

现在，交换机上设置了这样的队列：

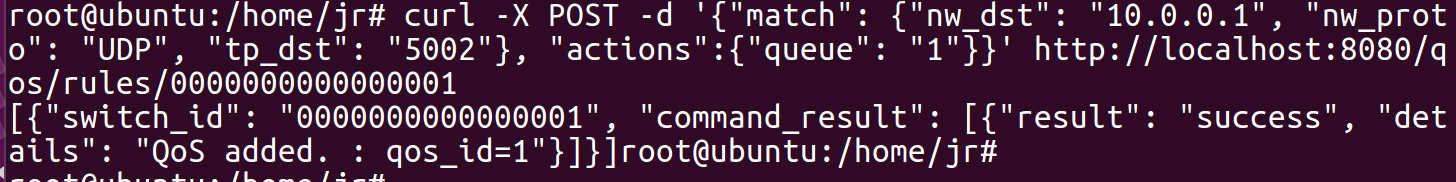


QoS设置

在交换机上添加流表项



*# curl -X POST -d '{"match": {"nw\_dst": "10.0.0.1", "nw\_proto": "UDP", "tp\_dst": "5002"}, "actions":{"queue": "1"}}' http://localhost:8080/qos/rules/0000000000000001*



"match": {"nw\_dst": "10.0.0.1", "nw\_proto": "UDP", "tp\_dst": "5002"}：流量匹配的条件，表示要匹配目的IP地址为10.0.0.1、传输层协议为UDP、目的端口为5002的数据包。

"actions": {"queue": "1"}：匹配成功后要执行的操作，将数据包发送到队列1。

验证查询一下交换机的设置：

*# curl -X GET http://localhost:8080/qos/rules/0000000000000001*



测试带宽

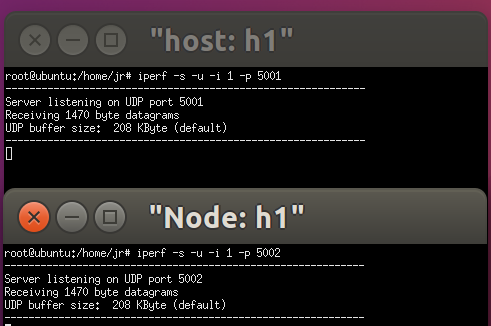
h1在UDP端口5001和5002监听，作为iperf的服务器。然后h2分别向h1的5001和5002端口发送1Mbps的流量。

在h1终端，在h1上开启iperf服务器

*# iperf -s -u -i 1 -p 5001*

mininet中用xterm h1, 再打开一个h1终端，开启iperf服务器

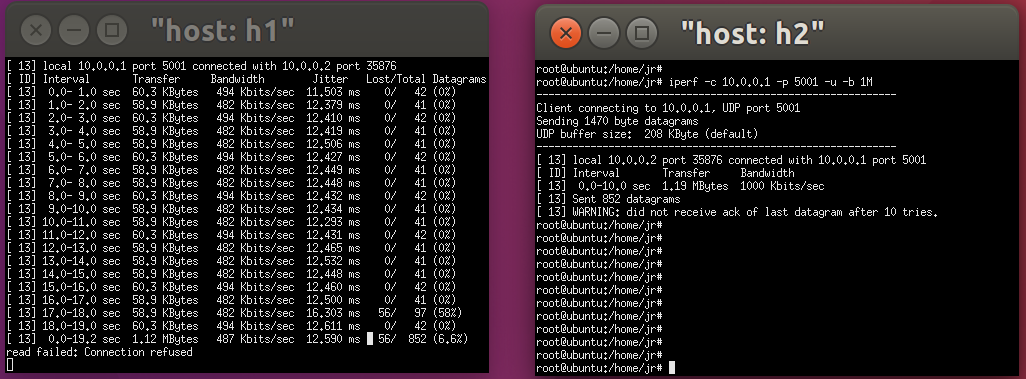
*# iperf -s -u -i 1 -p 5002*



打开一个h2终端，

*# iperf -c 10.0.0.1 -p 5001 -u -b 1M*

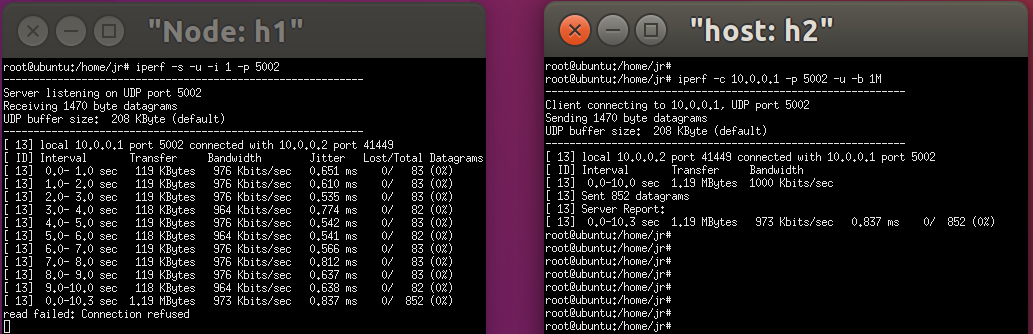
从h1那边的打印信息来看，实际带宽大约是500kbps，虽然命令是想用1M的速率，但实际只给500k，原因是这个流匹配默认的队列0，预留带宽为500kbps。



再向5002端口发送流

*# iperf -c 10.0.0.1 -p 5002 -u -b 1M*

从h1那边的打印信息来看，带宽大约是1M。因为这条流匹配流表项，继而匹配队列1，对于队列1，最低速率是800k，最高速率是1M。



**附录6：流量监控与QoS联调：**

先创建一个交换机两台主机的拓扑，其中交换机支持qos。

开启控制器，注意把流量监测的进程my2\_monitor\_13.py也跑起来。注意这里查询间隔设置为2秒了。否则流量计算不准确。

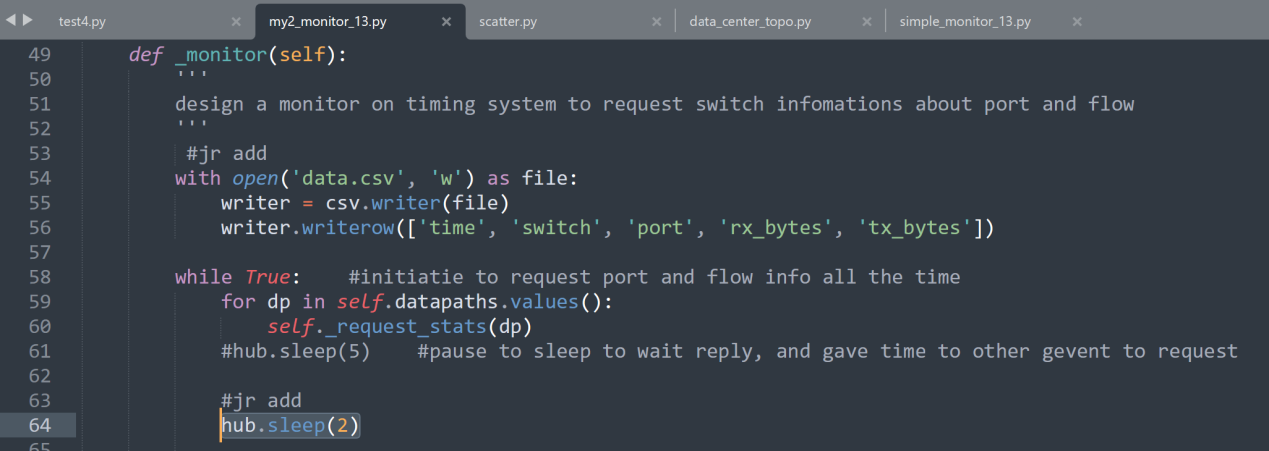
然后添加队列和规则，为h2与h1的5001UDP端口号通信的流，预留大于800k，小于1M的带宽。

通过iperf打流。生成一些流量。

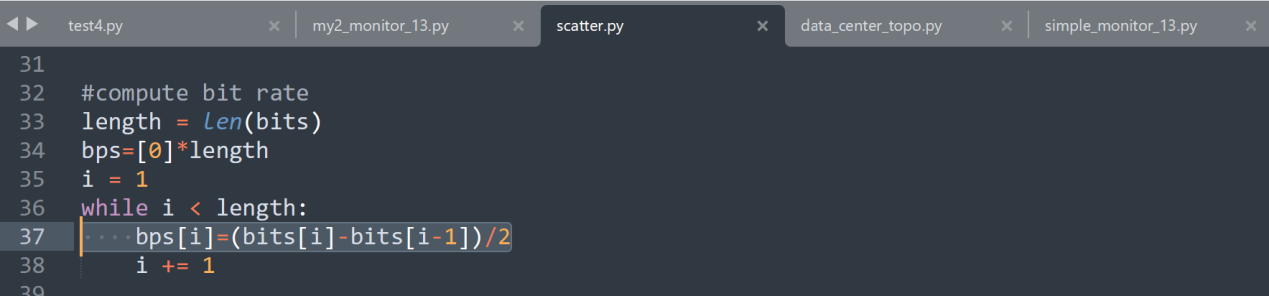
结束ryu，退出mininet。运行scatter.py生成流量曲线。

操作步骤：

1 修改my2\_monitor\_13.py的轮询间隔为2秒。如图。



2 修改scatter.py的计算速率的公式，时间间隔改为2。如图。



3 按照附录：QoS流程下来，

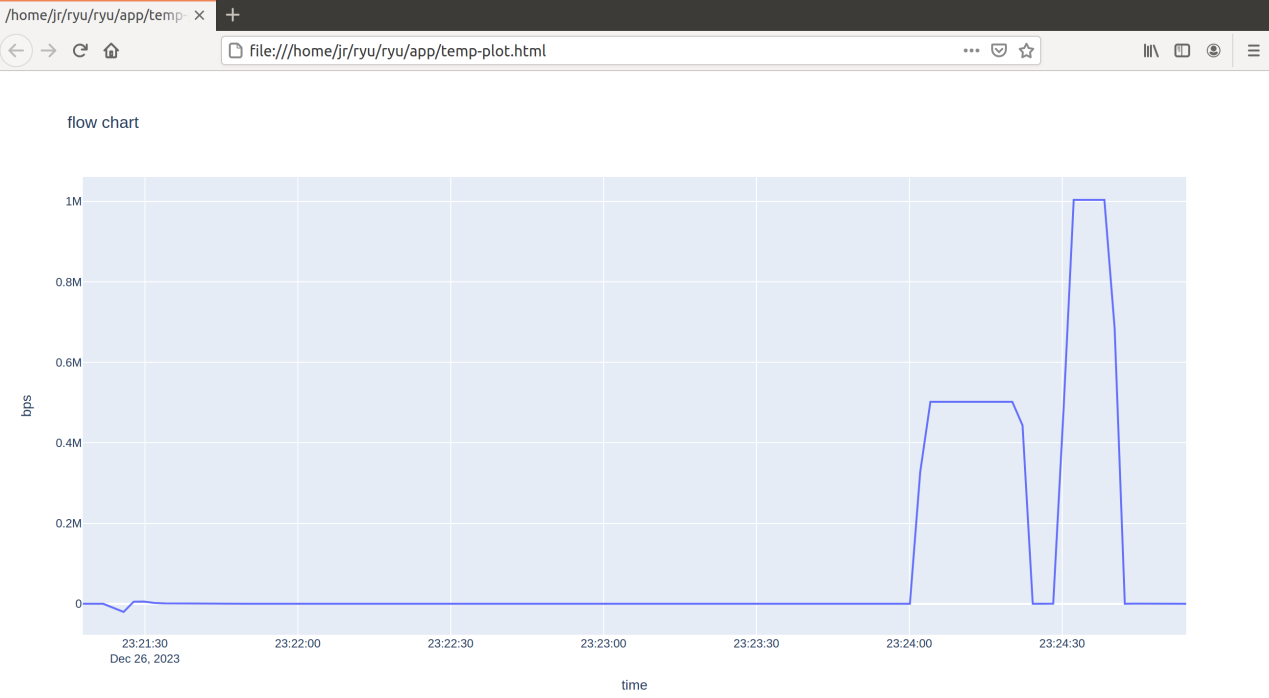
注意：第2步要多加一个my2\_monitor\_13.py

*# ryu-manager rest\_qos qos\_simple\_switch\_13 rest\_conf\_switch my2\_monitor\_13*

4 最后结束ryu，退出mininet。

运行scatter.py画流量曲线。如下图。





从流量曲线图可以看出，在h2 iperf h1的5001端口的那段时间，速率大约是500kbps，而h2 iperf h1的5002端口的那段时间，速率大约是1Mbps。监测到的流量数据，与实际情况相符。

**附录7：最短路径路由算法的实现short\_path.py**

[ryu实例---基于跳数的最短路径转发-CSDN博客](https://blog.csdn.net/weixin_40042248/article/details/116977397)

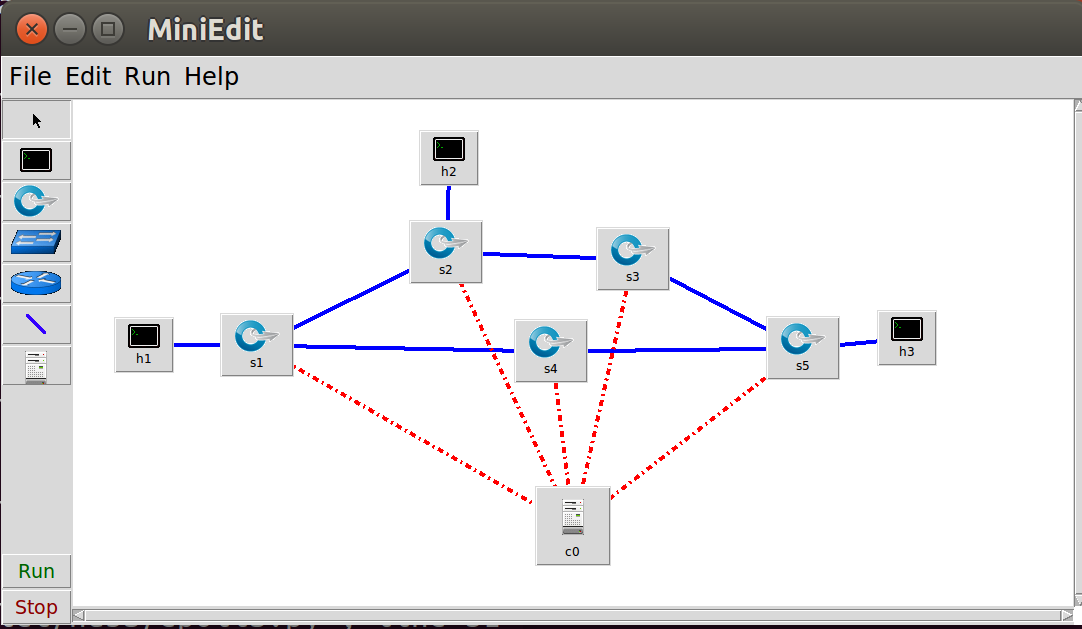
<https://blog.csdn.net/weixin_40042248/article/details/116977397>

一、mininet搭建拓扑

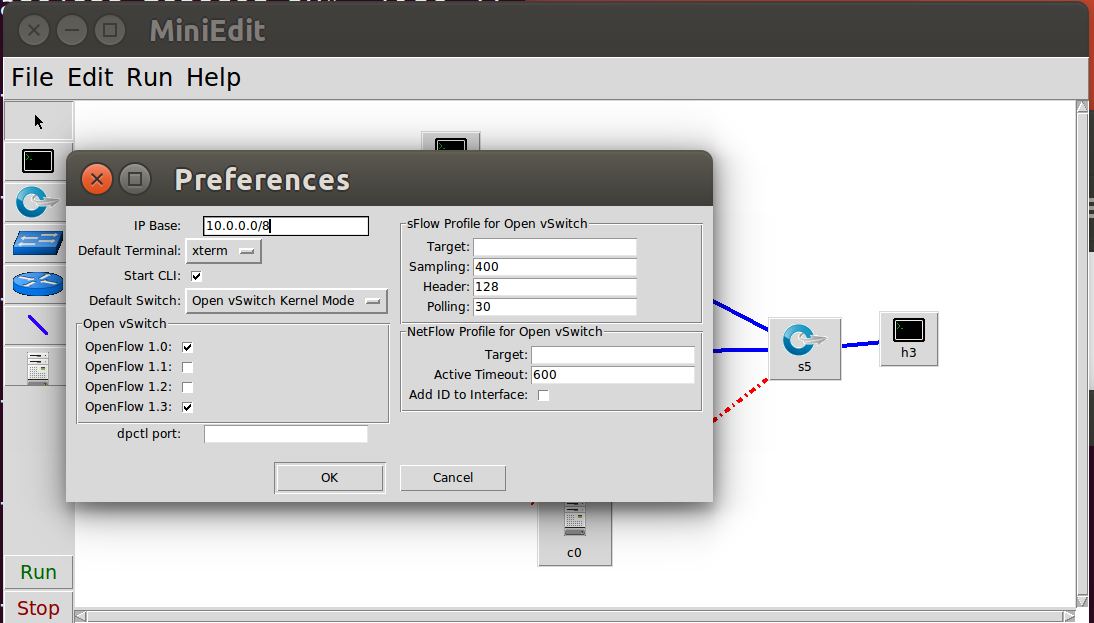
cd /home/jr/mininet/examples

./miniedit.py

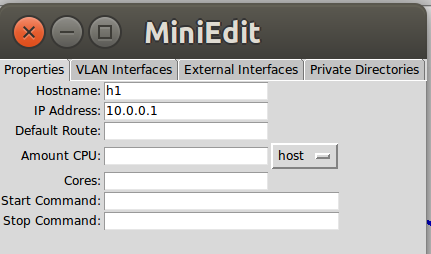
注意连线的时候要鼠标一直拖住连接。而不是点击。



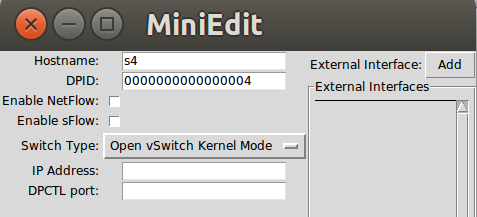
Edit->Preferences->IP Base，Start CLI，OpenFlow1.3如图。



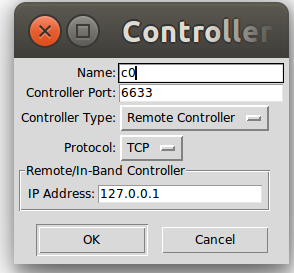
对于host右键，属性，Hostname，IP Address如图。



对于交换机，属性，DPID如图，switch type选择open vswitch kernel mode



对于控制器，属性，controller type选remote controller.



export l2 script，保存命名为topo\_shortpath.py，默认保存路径为/home/jr/mininet/examples

二、基于ryu开发最短路径应用

注意：

1 代码中的中文注释全部删除，否则会报错。

2 因为用到了模块networkx，所以先要安装：

pip install networkx

3 应用的名称为short\_path.py，运行的时候，

cd /home/jr/ryu/ryu/app

ryu-manager short\_path.py --observe-links

否则报错。

4 在windows下的文件拷贝到ubuntu下，开始的时候ryu文件夹是带锁的，表示只读。可以用chmod –R 777 ryu来设置文件夹是权限。就可以直接把short\_path.py拷贝到/home/jr/ryu/ryu/app

下了。

from ryu.base import app\_manager

from ryu.ofproto import ofproto\_v1\_3

from ryu.controller import ofp\_event

from ryu.controller.handler import MAIN\_DISPATCHER, CONFIG\_DISPATCHER

from ryu.controller.handler import set\_ev\_cls

from ryu.topology.api import get\_switch, get\_link

from ryu.topology import event

from ryu.lib.packet import packet

from ryu.lib.packet import ethernet

import networkx as nx

class PathForward(app\_manager.RyuApp):

OFP\_VERSIONS = [ofproto\_v1\_3.OFP\_VERSION]

def \_\_init\_\_(self, \*args, \*\*kwargs):

super(PathForward, self).\_\_init\_\_(\*args, \*\*kwargs)

self.G = nx.DiGraph()

self.topology\_api\_app = self

def add\_flow(self, datapath, priority, match, actions):

ofp = datapath.ofproto

ofp\_parser = datapath.ofproto\_parser

command = ofp.OFPFC\_ADD

inst = [ofp\_parser.OFPInstructionActions(ofp.OFPIT\_APPLY\_ACTIONS, actions)]

req = ofp\_parser.OFPFlowMod(datapath=datapath, command=command,

priority=priority, match=match, instructions=inst)

datapath.send\_msg(req)

@set\_ev\_cls(ofp\_event.EventOFPSwitchFeatures, CONFIG\_DISPATCHER)

def switch\_features\_handler(self, ev):

msg = ev.msg

datapath = msg.datapath

ofp = datapath.ofproto

ofp\_parser = datapath.ofproto\_parser

# add table-miss

match = ofp\_parser.OFPMatch()

actions = [ofp\_parser.OFPActionOutput(ofp.OFPP\_CONTROLLER, ofp.OFPCML\_NO\_BUFFER)]

self.add\_flow(datapath=datapath, priority=0, match=match, actions=actions)

@set\_ev\_cls(event.EventSwitchEnter)

def get\_topo(self, ev):

switch\_list = get\_switch(self.topology\_api\_app)

switches = []

for switch in switch\_list:

switches.append(switch.dp.id)

self.G.add\_nodes\_from(switches)

link\_list = get\_link(self.topology\_api\_app)

links = []

for link in link\_list:

links.append((link.src.dpid, link.dst.dpid, {'attr\_dict': {'port': link.src.port\_no}}))

self.G.add\_edges\_from(links)

for link in link\_list:

links.append((link.dst.dpid, link.src.dpid, {'attr\_dict': {'port': link.dst.port\_no}}))

self.G.add\_edges\_from(links)

def get\_out\_port(self, datapath, src, dst, in\_port):

dpid = datapath.id

if src not in self.G:

self.G.add\_node(src)

self.G.add\_edge(dpid, src, attr\_dict={'port': in\_port})

self.G.add\_edge(src, dpid)

if dst in self.G:

path = nx.shortest\_path(self.G, src, dst)

next\_hop = path[path.index(dpid) + 1]

out\_port = self.G[dpid][next\_hop]['attr\_dict']['port']

print(path)

else:

out\_port = datapath.ofproto.OFPP\_FLOOD

return out\_port

@set\_ev\_cls(ofp\_event.EventOFPPacketIn, MAIN\_DISPATCHER)

def packet\_in\_handler(self, ev):

msg = ev.msg

datapath = msg.datapath

ofp = datapath.ofproto

ofp\_parser = datapath.ofproto\_parser

dpid = datapath.id

in\_port = msg.match['in\_port']

pkt = packet.Packet(msg.data)

eth = pkt.get\_protocols(ethernet.ethernet)[0]

dst = eth.dst

src = eth.src

out\_port = self.get\_out\_port(datapath, src, dst, in\_port)

actions = [ofp\_parser.OFPActionOutput(out\_port)]

if out\_port != ofp.OFPP\_FLOOD:

match = ofp\_parser.OFPMatch(in\_port=in\_port, eth\_dst=dst, eth\_src=src)

self.add\_flow(datapath=datapath, priority=1, match=match, actions=actions)

data = None

if msg.buffer\_id == ofp.OFP\_NO\_BUFFER:

data = msg.data

out = ofp\_parser.OFPPacketOut(datapath=datapath, buffer\_id=msg.buffer\_id,

in\_port=in\_port, actions=actions, data=data)

datapath.send\_msg(out)

三、测试

1 运行mininet拓扑

2 ryu运行short\_path.py

3 在mininet中ping，查看ryu打印消息中查看ping消息是否走了最短路径。

**附录8：基于时延的最短路径算法**

[SDN实验（九）——基于链路质量的最短路径转发算法\_基于链路质量的最短路径切换-CSDN博客](https://blog.csdn.net/qq_41422448/article/details/127389619?spm=1001.2014.3001.5502)

这个代码可以跑。

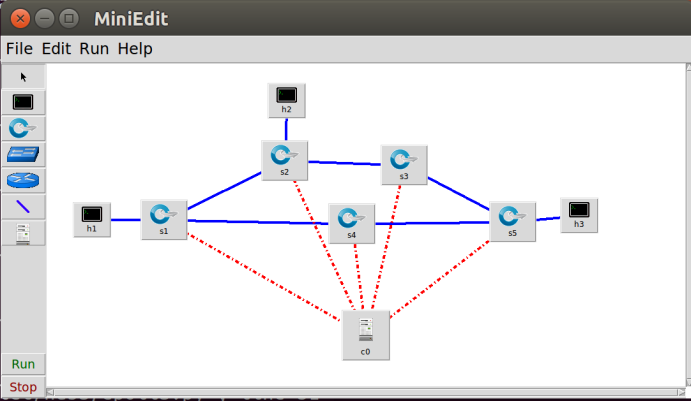
测试：

先运行ryu控制器：

注意：ryu-manager rate\_routing.py --observe-links，一定要加上--observe-links，否则报错。

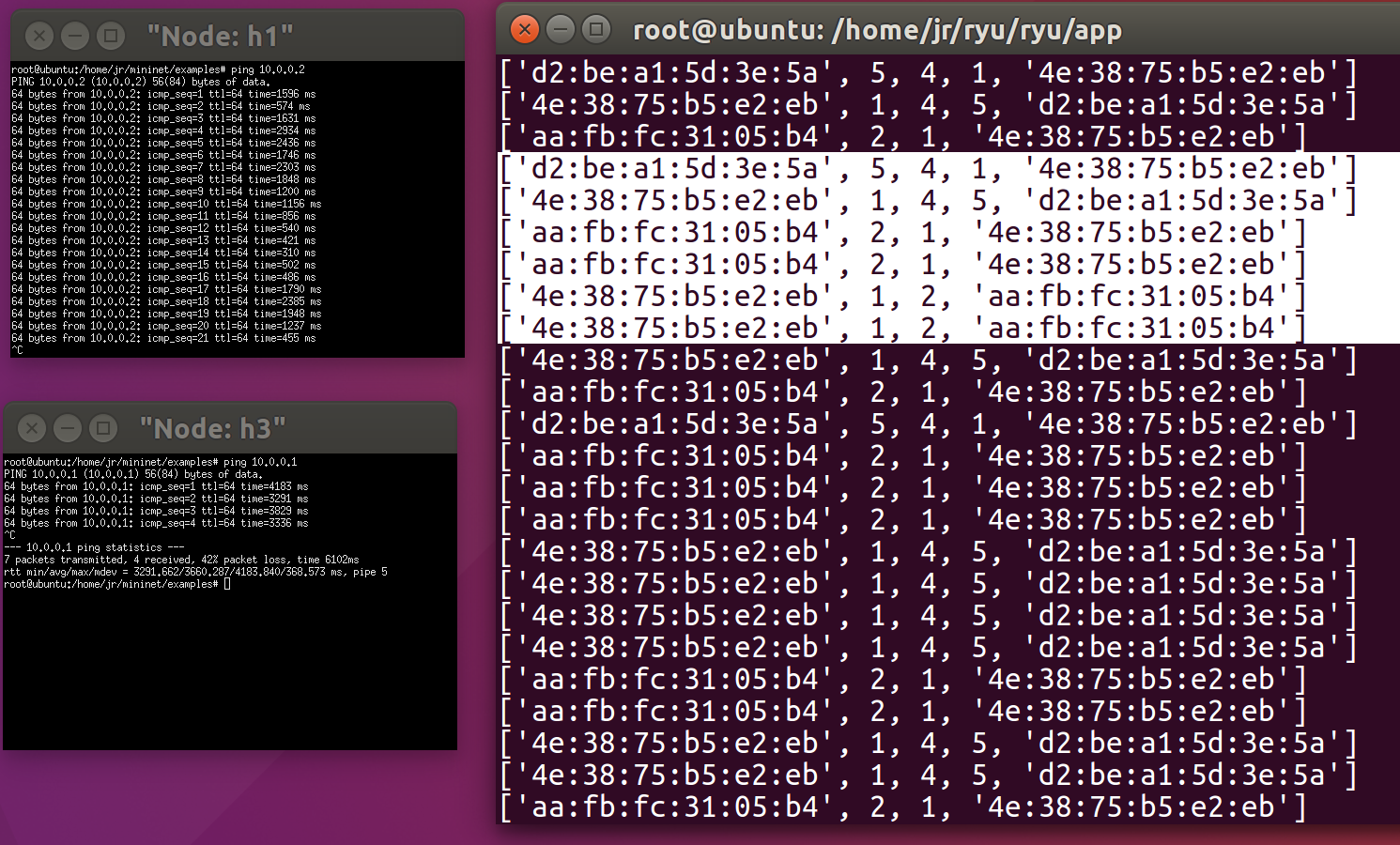
再运行mininet拓扑，还是用最短路径那个拓扑：

python topo\_rate\_routing.py



先让h1 ping h2。从打印信息看，走的是1,2两台交换机的路径。

在h1 ping h2的同时，让h3 ping h1。因为s1-s2的路径有在ping，所以此时选时延短的s1-s4-s5的路径。



注意：这个方案不太合理。重新换一个拓扑来测：

把h2连到s4上去。然后用iperf在h1和h2之间打流，再让h3 ping h1，看看会不会选择跳数更多的s5-s3-s2-s1?

或者考虑mininet里面是不是可以设置link的带宽。可以把h1和h3之间跳数少的链路带宽设小，让这条路的延迟变大，看看会不会选跳数多但延迟少的路走？？

或者考虑用qos之类去去限速，让跳数少的路速率慢，从而延迟大。从而选路的时候，选跳数多，但时延少的路。从而证明基于时延的路由选择算法。

**附录9：基于可用带宽的K-最短路径算法**

[SDN实验（六）——SDN流量监控-CSDN博客](https://blog.csdn.net/qq_41422448/article/details/127060313?spm=1001.2014.3001.5502)

<https://blog.csdn.net/qq_41422448/article/details/127060313?spm=1001.2014.3001.5502>

前面说了流量计算，基于流量的最短路径算法。但实际代码仅仅实现了端口信息的查询。

[基于网络流量的SDN最短路径转发应用 - 码农教程 (manongjc.com)](http://www.manongjc.com/detail/50-wczdpxnpzpwdxco.html)

<http://www.manongjc.com/detail/50-wczdpxnpzpwdxco.html>

这篇实现了基于可用带宽的K-最短路径算法。方案可以参考。有部分代码，没有完整代码。

github上有代码：[ryu/ryu/app/network\_awareness at master · muzixing/ryu (github.com)](https://github.com/muzixing/ryu/tree/master/ryu/app/network_awareness)

[基于跳数，时延，带宽的最短/优路径和负载均衡\_基于负载非跳频-CSDN博客](https://blog.csdn.net/Neo233/article/details/80988025?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522162436228016780261946326%2522%252C%2522scm%2522%253A%252220140713.130102334.pc%255Fall.%2522%257D&request_id=162436228016780261946326&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_v2~rank_v29-3-80988025.first_rank_v2_pc_rank_v29&utm_term=%E5%9F%BA%E4%BA%8E%E9%93%BE%E8%B7%AF%E5%B8%A6%E5%AE%BD%E7%9A%84%E6%9C%80%E7%9F%AD%E8%B7%AF%E5%BE%84&spm=1018.2226.3001.4187)

<https://blog.csdn.net/Neo233/article/details/80988025?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522162436228016780261946326%2522%252C%2522scm%2522%253A%252220140713.130102334.pc%255Fall.%2522%257D&request_id=162436228016780261946326&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_v2~rank_v29-3-80988025.first_rank_v2_pc_rank_v29&utm_term=%E5%9F%BA%E4%BA%8E%E9%93%BE%E8%B7%AF%E5%B8%A6%E5%AE%BD%E7%9A%84%E6%9C%80%E7%9F%AD%E8%B7%AF%E5%BE%84&spm=1018.2226.3001.4187>

为解析权重和最短K路径的参数，还需要在Ryu中注册全局的启动参数。注册参数的方法十分简单，只需要在Ryu顶级目录下的flags.py文件中添加如下的代码即可：

CONF.register\_cli\_opts([

# k\_shortest\_forwarding

cfg.IntOpt('k-paths', default=1, help='number for k shortest paths'),

cfg.StrOpt('weight', default='hop',

help='type of computing shortest path.')])

注意这段话加在原文后面。

完成以上修改后，将Github仓库中的代码下载到本地，然后放置到Ryu目录下合适的位置，比如Ryu/app目录下。

下载链接

<https://github.com/muzixing/ryu/blob/master/ryu/app/network_awareness/>

最后还需要重新安装Ryu：进入到ryu/的根目录，运行setup.py文件，并添加install参数。

sudo python setup.py install

重新安装完成之后，安装networkx

pip install networkx

启动shortest\_forwarding应用，并添加observe-links，链路权重和最短路径条数等重要参数,示例如下：

ryu-manager shortest\_forwarding --observe-links --k-paths=2 --weight=bw

启动Ryu之后，启动任意的SDN网络，如Mininet模拟的网络，并连接到Ryu控制器。最后可以在Mininet输入框中输入pingall进行测试。

sudo mn --controller=remote --topo=tree,3,3 –mac

注意用这个拓扑，ping不通。

为了方便使用，读者可以通过修改setting.py中的信息来修改应用的重要参数，比如获取链路信息的周期，是否打印网络信息等等。setting信息具体如下所示：

DISCOVERY\_PERIOD = 10

MONITOR\_PERIOD = 10

DELAY\_DETECTING\_PERIOD = 10

TOSHOW = True

MAX\_CAPACITY = 281474976710655L

读者可以通过修改对应的周期数值，来修改对应模块获取信息的周期，其单位为秒。TOSHOW是一个布尔值，用于设置是否在终端打印网络信息。MAX\_CAPACITY值为链路最大可用带宽值，可根据实际情况进行修改。

放弃。