软件定义网络实验课

—— Software Defined Networking

周晓波

xiaobo.zhou@tju.edu.cn

工具及环境部署

- **环境部署**

➢ 虚拟机环境：
  ✓ Virtual Box　https://www.virtualbox.org/wiki/Downloads
  ✓ ubuntu 16.0.4 LTS　http://www.ubuntu.org.cn/download/desktop

➢ 工具：mininet + Ryu

工具及环境部署

- **环境部署 - Mininet安装**

➢ 刚装好的虚拟机，需要安装git，python，vim 等

```
sudo apt-get install git
```

➢ 通过apt安装工具包安装mininet

```
sudo apt-get install -y mininet
```

➢ 通过源码安装minient

```
git clone git://github.com/mininet/mininet
cd mininet
git tag
git checkout -b 2.2.2
sudo util/install.sh -a
```

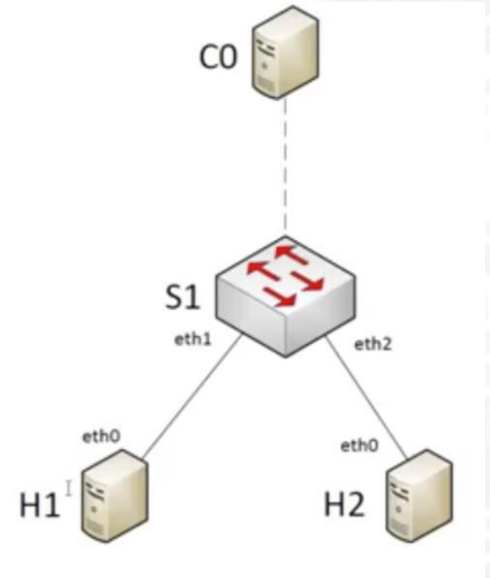
```
util/install.sh -a

-a 全部安装
-nfv 安装 mininet，Openflow 和 Open vSwitch
-s mydir 選擇之後想儲存的目錄，而並非主目錄
```

工具及环境部署

- **环境部署 - Mininet安装**

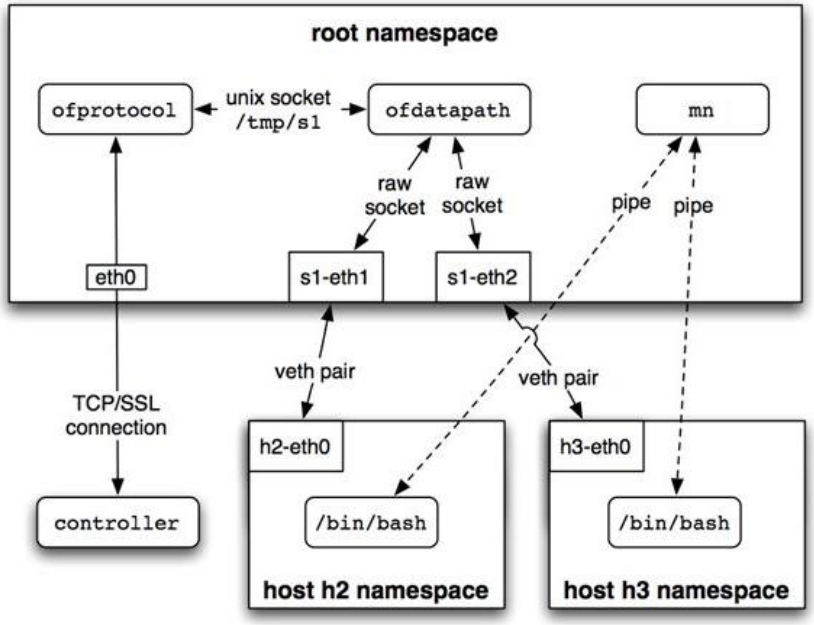
➢ Mininet安装成功

```
xiaobo@xiaobo-VirtualBox:~$ sudo mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```



工具及环境部署

- **环境部署 - Mininet安装**
  - Mininet的原理



- ✓ 基于Linux Container （LXC）
- ✓ 通过 namespace 的隔离，使得创建出来的每个 controller，switch 都在一个单独的 namespace 中，每个 namespace 中都可以模拟出网卡，并通过创建 veth pair 来连接不同的 namespace，使不同的 namespace 之间相互通信。
- ✓ 就像每个都在一个单独的虚拟机中一般，从而便可实现大规模网络的模拟。

工具及环境部署

- **环境部署 - Mininet安装**
  - Mininet的原理

```
xiaobo@VirtualBox: ~/mininet/bin

PLACEMENT = { 'block': SwitchBinPlacer, 'random': RandomPlacer }

# built in topologies, created only when run
TOPODEF = 'minimal'
TOPOS = { 'minimal': MinimalTopo,
          'linear': LinearTopo,
          'reversed': SingleSwitchReversedTopo,
          'single': SingleSwitchTopo,
          'tree': TreeTopo,
          'torus': TorusTopo }
```

mn

```
xiaobo@VirtualBox: ~/mininet/mininet

                        port1=0, port2=( k - h + 1 ) )


class MinimalTopo( SingleSwitchTopo ):
    "Minimal topology with two hosts and one switch"
    def build( self ):
        return SingleSwitchTopo.build( self, k=2 )
```

topo.py

```
xiaobo@VirtualBox: ~/mininet/mininet
# pylint: disable=arguments-differ

class SingleSwitchTopo( Topo ):
    "Single switch connected to k hosts."

    def build( self, k=2, **_opts ):
        "k: number of hosts"
        self.k = k
        switch = self.addSwitch( 's1' )
        for h in irange( 1, k ):
            host = self.addHost( 'h%s' % h )
            self.addLink( host, switch )
```

工具及环境部署

- **环境部署 - Mininet安装**
  - Mininet的命令

```
xiaobo@VirtualBox: ~/mininet/mininet
s1 ...
*** Starting CLI:
mininet> help

Documented commands (type help <topic>):
========================================
EOF    gterm  iperfudp  nodes        pingpair      py      switch
dpctl  help   link      noecho       pingpairfull  quit    time
dump   intfs  links     pingall      ports         sh      x
exit   iperf  net       pingallfull  px            source  xterm

You may also send a command to a node using:
  <node> command {args}
For example:
  mininet> h1 ifconfig

The interpreter automatically substitutes IP addresses
for node names when a node is the first arg, so commands
like
  mininet> h2 ping h3
should work.

Some character-oriented interactive commands require
noecho:
  mininet> noecho h2 vi foo.py
However, starting up an xterm/gterm is generally better:
  mininet> xterm h2

mininet>
```

工具及环境部署

- **环境部署 - Mininet安装**
  - Mininet的命令 – dump

```
xiaobo@VirtualBox: ~/mininet/mininet
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=31733>
<Host h2: h2-eth0:10.0.0.2 pid=31736>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=31742>
<Controller c0: 127.0.0.1:6653 pid=31726>
mininet>
```

✓ 查看所有节点相关信息
✓ 节点类别、名字、IP信息、pid

工具及环境部署

- **环境部署 - Mininet安装**

➢ Mininet的命令 – nodes

```
mininet> nodes
available nodes are:
c0 h1 h2 s1
mininet>
```

✓ 只想知道有哪些节点

➢ Mininet的命令 – net

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
mininet>
```

✓ 网络连接

工具及环境部署

- **环境部署 - Mininet安装**
  - Mininet的命令 – 节点执行命令

```
mininet> h1 ifconfig
h1-eth0   Link encap:Ethernet  HWaddr 16:f6:cd:e4:35:29
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::14f6:cdff:fee4:3529/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:32 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3629 (3.6 KB)  TX bytes:648 (648.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

mininet>
```

✓ 查看所有节点相关信息
✓ 节点类别、名字、IP信息、pid

工具及环境部署

- **环境部署 - Mininet安装**
  - Mininet的命令 – ping，pingall

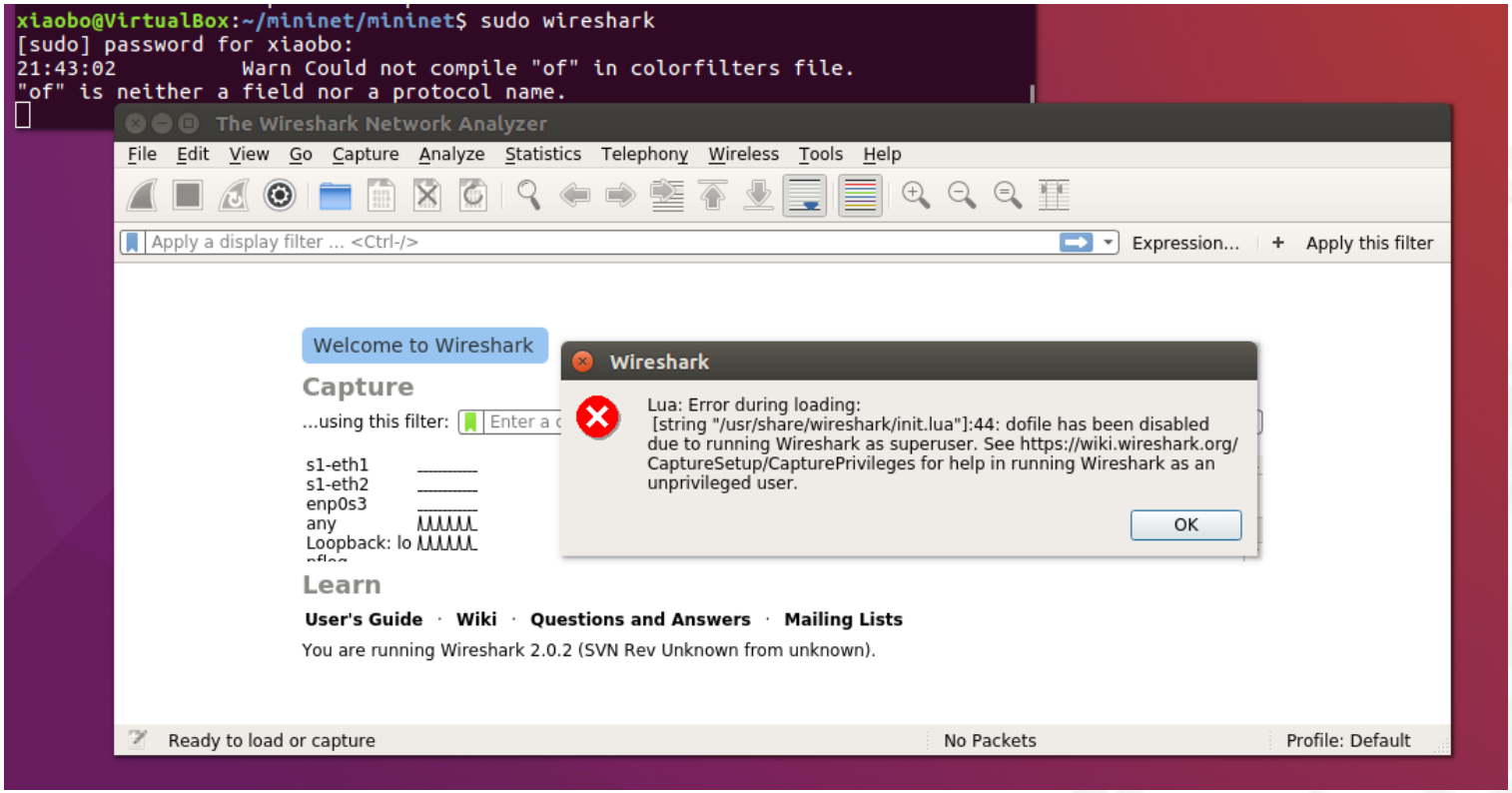
```
mininet> h1 ping -c 4 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=2.95 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.244 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.048 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.054 ms

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.048/0.825/2.955/1.232 ms
mininet>
```

✓ 连通性测试
✓ h1能否和h2通信

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

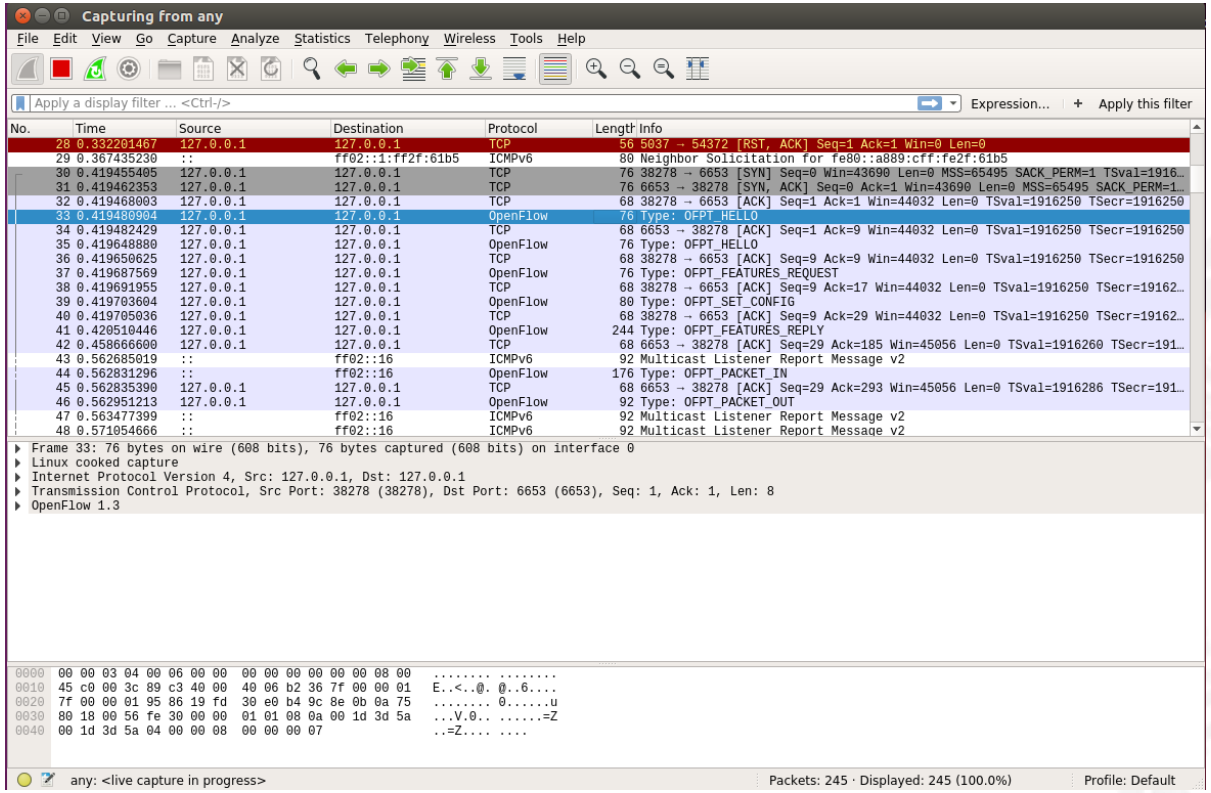
工具及环境部署

- **环境部署 - OpenFlow协议分析**
  - 借助Wireshark（完整安装mininet后自带）



工具及环境部署

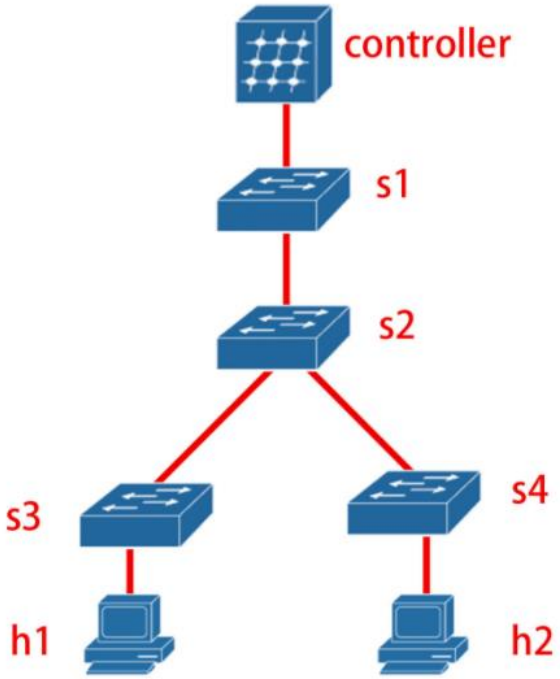
- **环境部署 - OpenFlow协议分析**
  - 借助Wireshark



另开一个终端，输入 sudo mn 打开 mininet

工具及环境部署

- **环境部署 - Mininet安装**
  - 建立自己的网络拓扑结构



- ✓ 不使用mininet自带的topo，如何建立自己的网络拓扑结构？
- ✓ 通过python脚本来随心所欲的创建！

工具及环境部署

- **环境部署 - Mininet安装**
  - 建立自己的网络拓扑结构

首先因为我们只是写拓扑结构，该脚本并不是为了直接在命令行中运行，所以开头便可不添加 #!/usr/bin/env python

注意代码缩进

```
xiaobo@VirtualBox: ~
from mininet.topo import Topo

class Testtopo(Topo):
        def __init__(self):

                Topo.__init__(self)

                S1=self.addSwitch('s1')
                S2=self.addSwitch('s2')
                S3=self.addSwitch('s3')
                S4=self.addSwitch('s4')
                H1=self.addHost('h1')
                H2=self.addHost('h2')

                self.addLink(S1,S2)
                self.addLink(S2,S3)
                self.addLink(S2,S4)
                self.addLink(S3,H1)
                self.addLink(S4,H2)

topos={
        'firsttopo':(lambda: Testtopo())
}
~
~
~
```

```
xiaobo@VirtualBox:~$ sudo mn --custom=my_topo.py --topo firsttopo
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(s1, s2) (s2, s3) (s2, s4) (s3, h1) (s4, h2)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
mininet>
```

工具及环境部署

- **环境部署 - Ryu的安装**

➢ 需要的python套件：python-eventlet python-routes python-webob python-paramiko python-pip

```
sudo apt-get install python-eventlet
```

➢ 通过pip安装Ryu

```
sudo pip install ryu
```

➢ 通过源码安装minient

```
git clone git://github.com/osrg/ryu.git
cd ryu
sudo pip install -r tools/pip-requires
sudo python setup.py install
```

工具及环境部署

- **环境部署 - Ryu的安装**

➢ 安装成功

```
xiaobo@VirtualBox:~$ ryu-manager
lzma module is not available
Registered VCS backend: git
Registered VCS backend: hg
Registered VCS backend: svn
Registered VCS backend: bzr
loading app ryu.controller.ofp_handler
instantiating app ryu.controller.ofp_handler of OFPHandler
```

Terminal 1

工具及环境部署

- **环境部署 - Ryu和mininet的连接**

1. 外部启动ryu，通过指定ip地址来连接

```
xiaobo@VirtualBox:~$ sudo mn --controller=remote,ip=127.0.0.1
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

Terminal 2

```
xiaobo@VirtualBox: ~
xiaobo@VirtualBox:~$ sudo ovs-vsctl show
[sudo] password for xiaobo:
ac46f5db-c383-46d0-be5c-028e51c380e1
    Bridge "s1"
        Controller "ptcp:6654"
        Controller "tcp:127.0.0.1:6653"
            is_connected: true
        fail_mode: secure
        Port "s1-eth1"
            Interface "s1-eth1"
        Port "s1-eth2"
            Interface "s1-eth2"
        Port "s1"
            Interface "s1"
                type: internal
    ovs_version: "2.5.2"
xiaobo@VirtualBox:~$
```

Terminal 3

工具及环境部署

- **环境部署 - Ryu和mininet的连接**

2. 直接指定ryu控制器

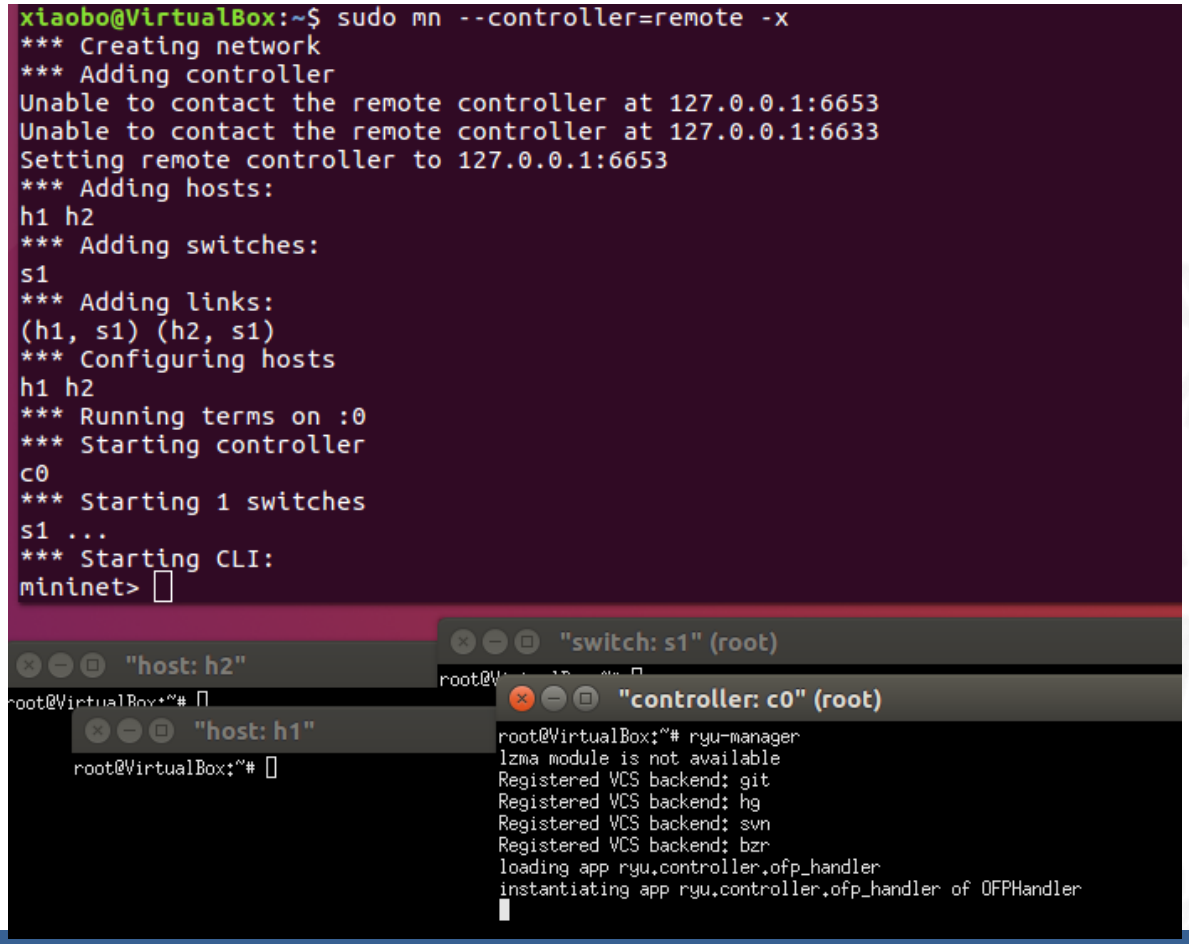
```
xiaobo@VirtualBox:~$ sudo mn --controller=ryu
*** Creating network
*** Adding controller
warning: no Ryu modules specified; running simple_switch only
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=3832>
<Host h2: h2-eth0:10.0.0.2 pid=3835>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=3841>
<Ryu c0: 127.0.0.1:6653 pid=3825>
mininet>
```

工具及环境部署

- **环境部署 - Ryu和mininet的连接**

3. 在xterm中连接控制器

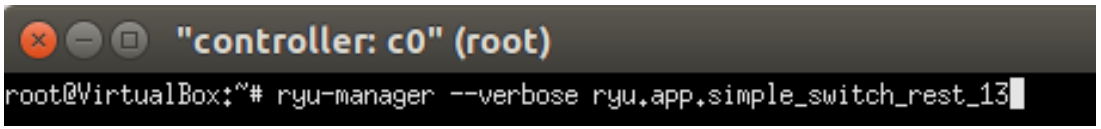
X参数可以再启动时同时开启
每个节点的xterm



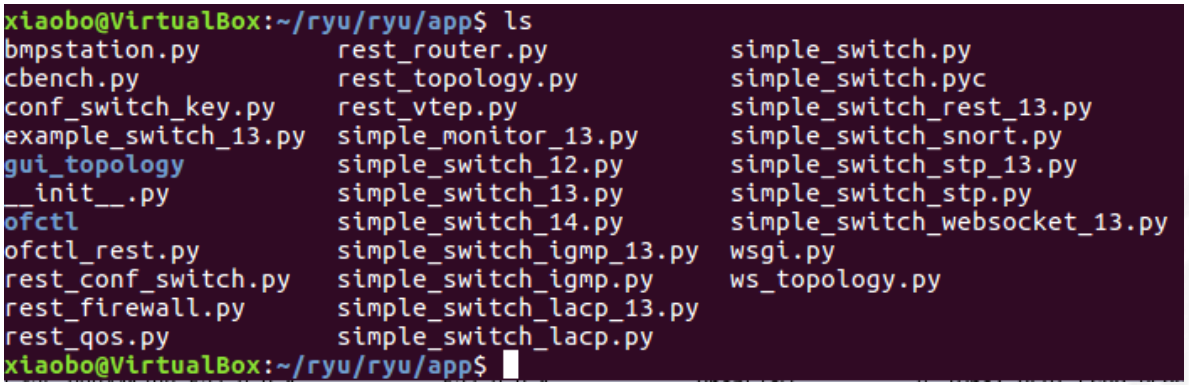
工具及环境部署

- **环境部署 - Ryu的APP**

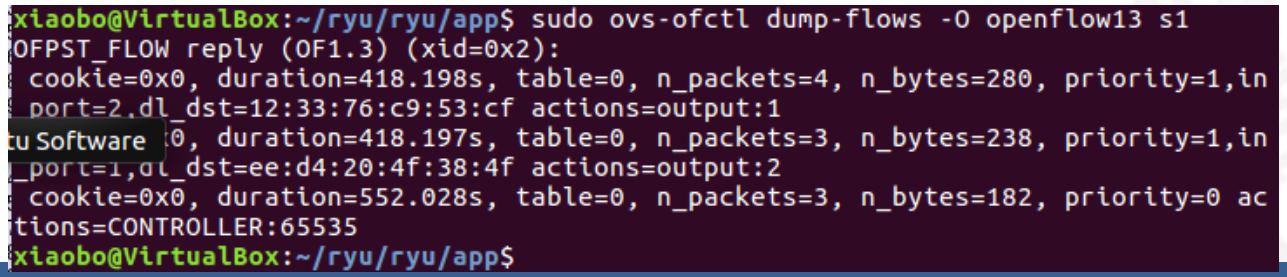
➢ 在控制器上运行app



控制器窗口运行app



自带app例子



运行pingall后查看S1流表

工具及环境部署

- **环境部署 - Ryu的RESTAPI简介**

➢ Rest API简介

✓ REST即表述性状态传递（RepreSentational State Transfer），是一种针对网络应用的设计和开发方式，可以降低开发的复杂性，提高系统的可伸缩性。

✓ 表述性状态转移是一组构架约束条件和原则，满足这些约束和原则的应用程序或设计就是RESTful，REST是设计风格而不是标准，它通常基于使用HTTP，URI，XML以及HTML这些现有的广泛流行的协议和标准。

✓ REST定义了一组体系构架原则，可以根据这些原则设计以系统资源为中心的Web服务，包括使用不同语言编写的客户端如何通过HTTP处理和传输资源状态。

工具及环境部署

- **环境部署 - Ryu的RESTAPI简介**

➢ Ryu中的Rest API简介

✓ ryu已经提供了一些RESTAPI的定义，在ryu/app目录下可以找到如下相关的文件：

```
ofctl_rest.py rest_topology.py rest_firewall.py rest_qos.py rest_router.py
```

✓ 打开这些文件简单浏览下可以发现他们分别提供了和OpenFlow协议，拓扑等相关的信息查询和配置，查询的结果以json格式返回给浏览器，而配置会调用相关模块的相关函数，可以简单的看下获取SDN网络中的交换机的代码。获取switches的指令为http://ip:port/stats/switches

```
path = '/stats'
uri = path + '/switches'
mapper.connect('stats', uri,
                       controller=StatsController, action='get_dpids',
                       conditions=dict(method=['GET']))

def get_dpids(self, req, **_kwargs):
    dps = list(self.dpset.dps.keys())
    body = json.dumps(dps)
    return Response(content_type='application/json', body=body)
```

工具及环境部署

- **环境部署 - Ryu的RESTAPI简介**

➢ 启动Ryu相关组件

```
xiaobo@VirtualBox:~/ryu/ryu/app$ ryu-manager ofctl_rest.py simple_switch_13.py
lzma module is not available
Registered VCS backend: git
Registered VCS backend: hg
Registered VCS backend: svn
Registered VCS backend: bzr
loading app ofctl_rest.py
loading app simple_switch_13.py
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app simple_switch_13.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ofctl_rest.py of RestStatsApi
(7223) wsgi starting up on http://0.0.0.0:8080
```

运行Ryu之后，可以查看到wsgi启动，监听端口为8080

工具及环境部署

- **环境部署 - Ryu的RESTAPI简介**

➢ 在ofctl_rest.py源码的前面部分，我们可以查看到写成注释形式的RESTAPI的使用方法，节选如下：

```
# REST API
#
# Retrieve the switch stats
#
# get the list of all switches
# GET /stats/switches
#
# get the desc stats of the switch
# GET /stats/desc/
#
# get flows stats of the switch
# GET /stats/flow/
#
# get flows stats of the switch filtered by the fields
# POST /stats/flow/
#
# get aggregate flows stats of the switch
# GET /stats/aggregateflow/
#
# get aggregate flows stats of the switch filtered by the fields
# POST /stats/aggregateflow/
#
# get ports stats of the switch
# GET /stats/port/
```

工具及环境部署

- **环境部署 - Ryu的RESTAPI简介**

➢ Mininet连接控制器

```
xiaobo@VirtualBox:~$ sudo mn --controller=remote,ip=127.0.0.1 --mac
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
```

打开mininet，运行任意拓扑，连接控制器RYU。并执行pingall，检测网络联通性。

工具及环境部署

- **环境部署 - 使用REST API**

➢ 三种方式

✓ 在浏览器中输入类似http://ip:port/stats/switches命令来发送GET请求，获取信息，ip为控制器的IP，ryu提供的port为8080

✓ 用curl（curl是利用URL语法在命令行方式下工作的开源文件传输工具）代替浏览器，在终端输入curl http://ip:port/stats/switches来传输内容

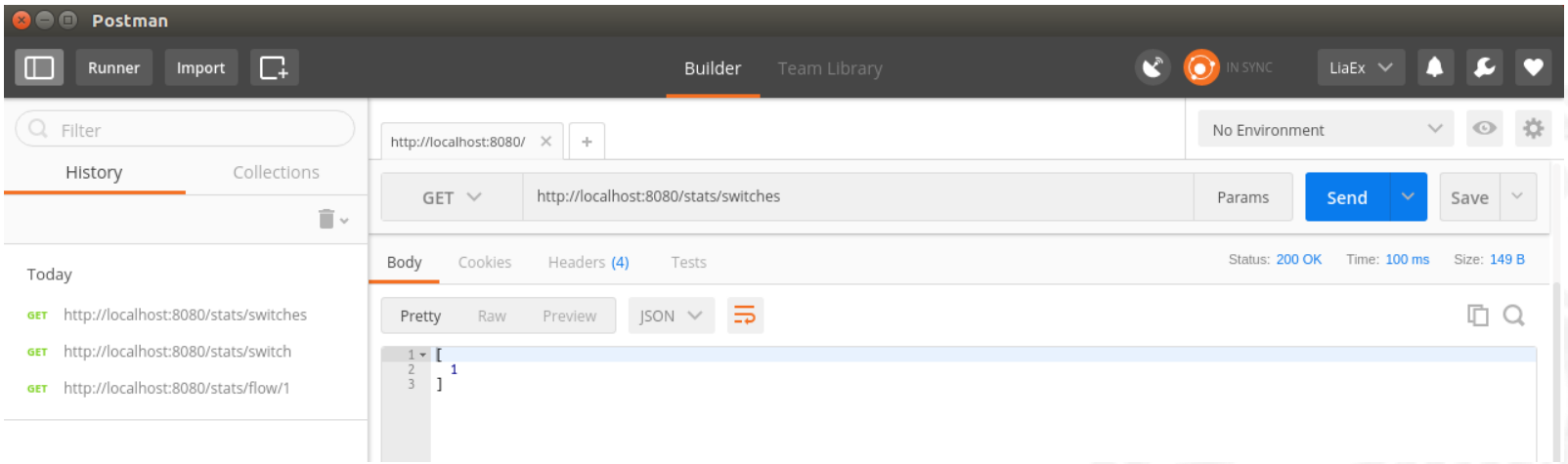
✓ 使用Postman，非常推荐这种方法，原因如下：

- 提供了Pretty和Raw两种结果展示方法，Raw和前面两种方法的返回格式一样，但是Pretty的格式的可视性和便读性远远好于Raw
- 提供了JSON和XML两种格式展示结果
- 下发流表同样简单，可视化比较好，只需要在body里面按照python字典的格式书写流表，然后下发即可
- 可以显示对请求的response，比如成功的话STATUS会显示200 OK
- 下载网址：https://www.getpostman.com/apps

流表的灵活性是SDN网络的优势之一，利用上述方法在查看流表，验证网络功能，开发APP具有非常重要的作用。

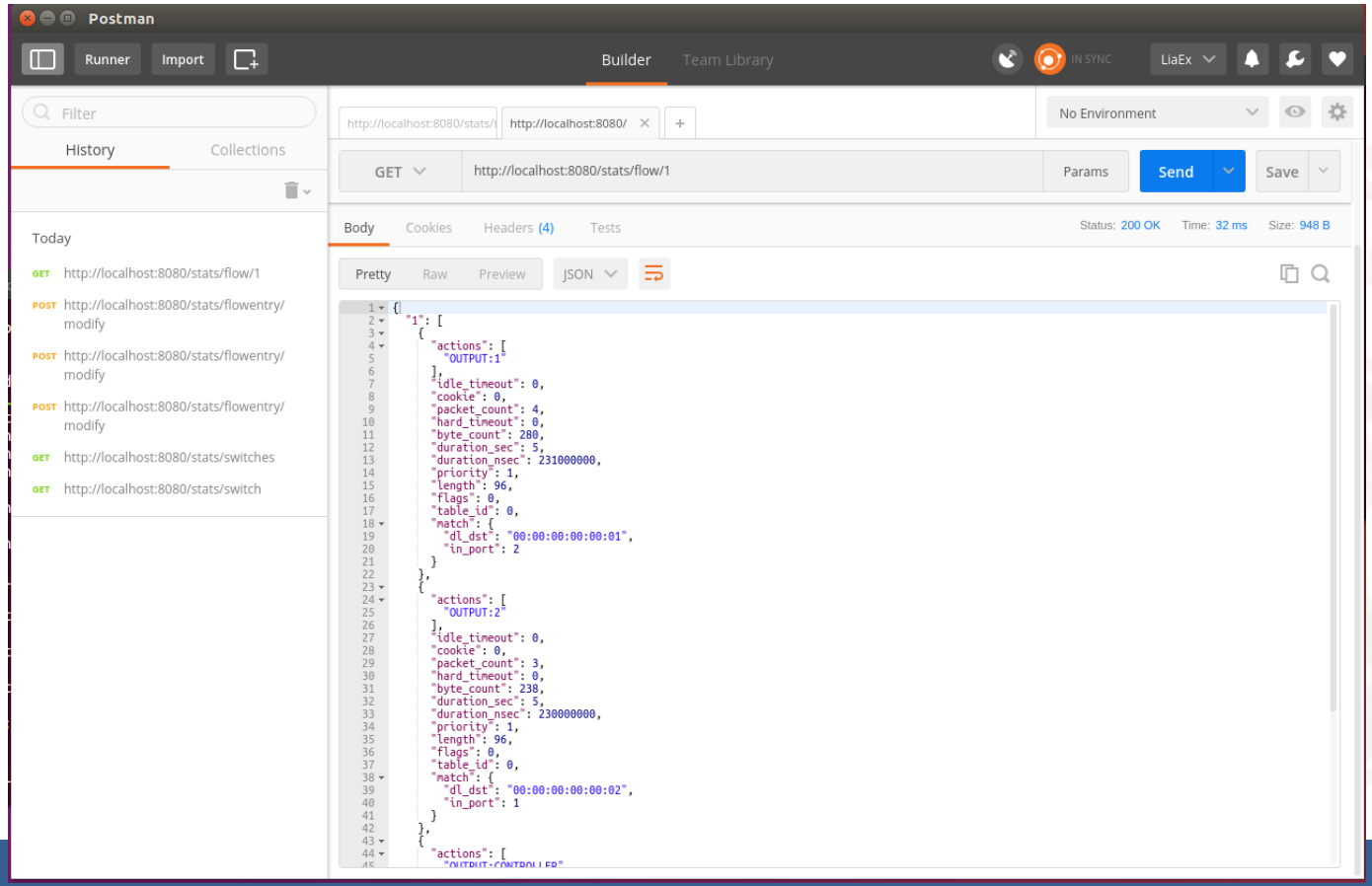
工具及环境部署

- **环境部署 - 使用REST API**
  - 查看网络中的交换机



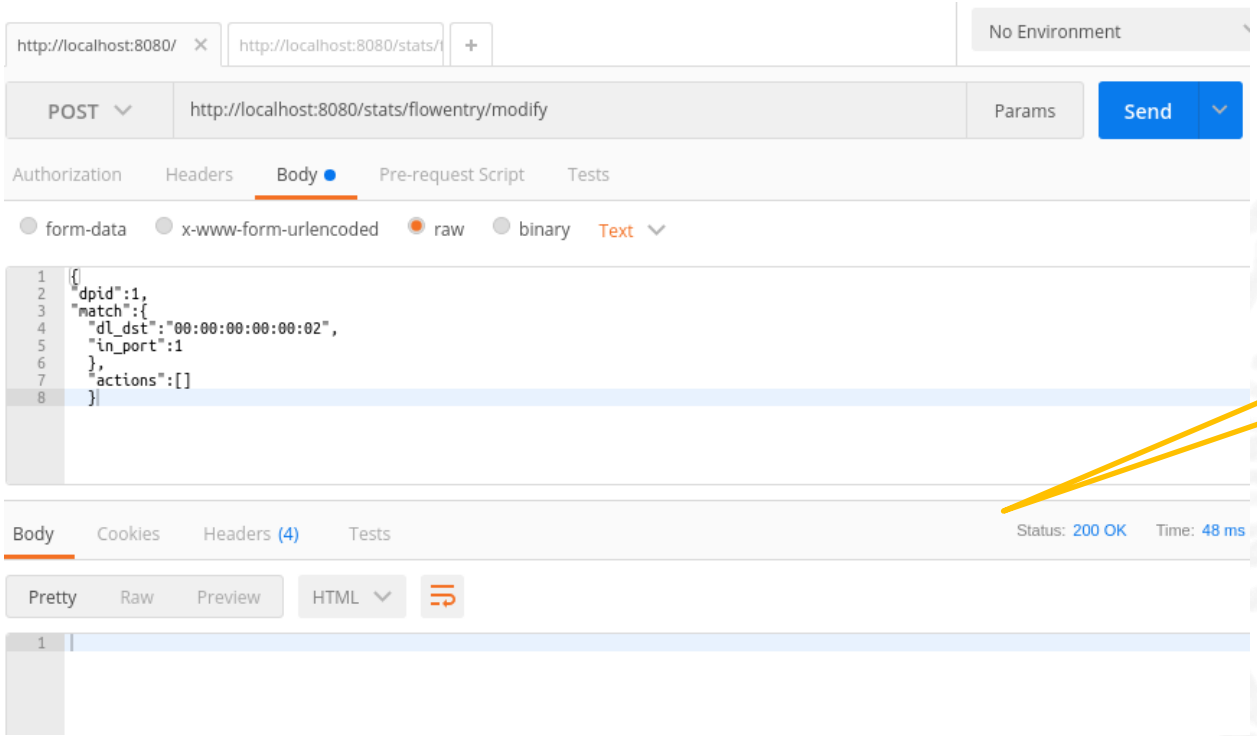
工具及环境部署

- **环境部署 - 使用REST API**
  - 请求pid为1的交换机上的流表信息



工具及环境部署

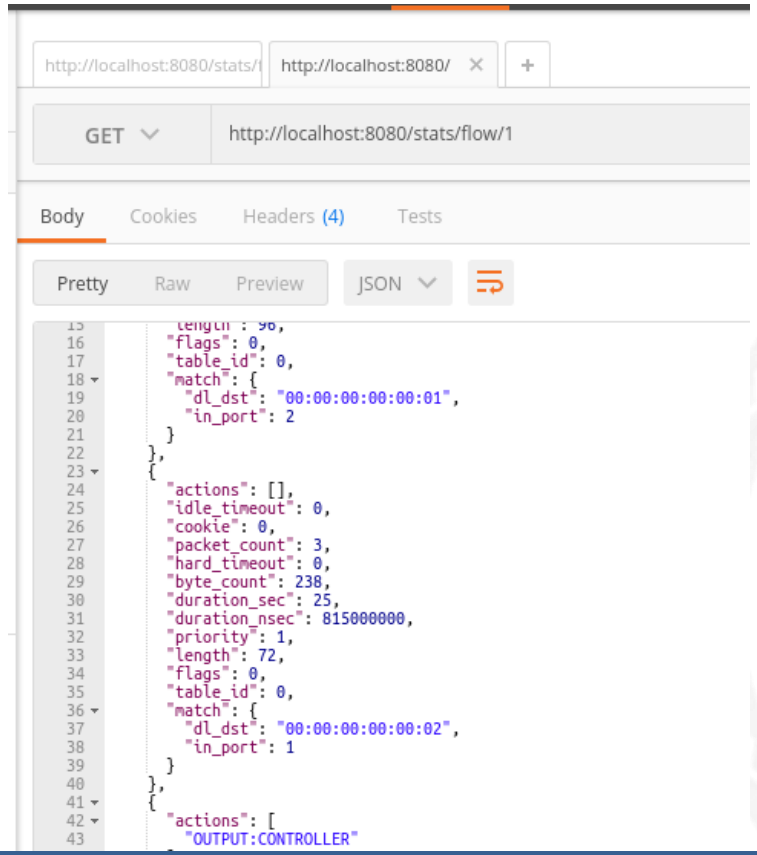
- **环境部署 - 使用REST API**
  - 流表修改 -可以使用POST动作类型，下发一个flow_mod消息，对现有流表进行操作



send之后，返回200状态码，提示成功。RYU返回消息内容为1。

工具及环境部署

- **环境部署 - 使用REST API**
  - 此时重新获取交换机上的刘表，可以观察到流表修改已经成功。



工具及环境部署

- **环境部署 - 使用REST API**

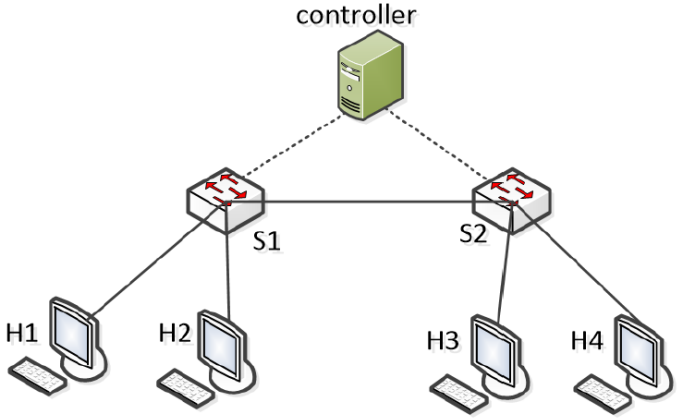
➢ 在Mininet中重新pingall测试联通性，果然不通，修改流表结果正确。

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X
h2 -> X
*** Results: 100% dropped (0/2 received)
mininet>
```

✓ 其他RESTAPI的示例不再赘述，可自行尝试。
✓ 基于Ryu的APP编程入门请参考
http://www.sdnlab.com/1785.html

工具及环境部署

- SDN编程实践



要求实现功能：

1. 参照上图所示拓扑部署一个基于SDN的简单网络环境，南向接口采用OpenFlow 1.3协议。（描述生成步骤，截屏相应命令，贴出生成topo的python代码）
2. H1、H2、H3、H4任意两两可互通。（给出4个节点两两互ping测试截图，每对只需ping二次即可）
3. 下发流表项实现H1和H2，H1和H4不能互通，并使得H1和H3可互通。（给出具体操作，查询到的流表信息和ping测试结果）
4. 结合捕获到的某些OpenFlow协议报文，分析其报文结构，并简要描述该类报文作用。（一种即可，限300字）
5. 思考在同样网络拓扑下如何在传统网络中实现要求3，并分析与SDN实现之间的差别。（限500字）

工具及环境部署

- SDN编程实践
  - ✓ 每人提交一份实验报告，实验过程可分小组讨论。
  - ✓ 报告截止日期：2017.06.05
  - ✓ 报告应为PDF格式文档，报告应详细列明实验步骤，截图。报告中所有图片必须在图片下方表明题注。
  - ✓ 提交文件以“SDN实验_学号_姓名.pdf”格式命名。报告以邮件形式发至 tanklab@163.com 邮件标题和文件标题相同。
  - ✓ 实验所需材料及课件可在百度云下载（有效期至2017.05.26）
    http://pan.baidu.com/s/1jHEsOmy
    提取密码 2cjj

工具及环境部署

- 参考资料

- ✓ Mininet源码分析 http://hwchiu.logdown.com/posts/221370-mininet-parsing

- ✓ OpenFlow tutorial https://github.com/mininet/openflow-tutorial/wiki

- ✓ OpenFlow 1.3流表项 http://www.brocade.com/content/html/en/configuration-guide/netiron-05900-sdnguide/GUID-B26EC8DB-D5A7-422E-94A0-94CC981595B3.html

- ✓ Ryu RESTAPI的使用 http://www.sdnlab.com/11552.html

谢谢！

Q&A