

一、实验目的

- 1.通过 Mininet 仿真平台熟悉 SDN 仿真基本环境;
- 2.熟悉 Opendaylight 控制器的基本操作;
- 3.实现简单的 OpenFlow 协议管理功能。

二、实验环境

2.1 仿真器：Mininet

Mininet 是一个轻量级软件定义网络和测试平台;它采用轻量级的虚拟化技术使一个单一的系统看起来像一个完整的网络,也可简单理解为 SDN 网络系统中的一种基于进程虚拟化平台,它支持 OpenFlow、OpenvSwitch 等各种协议,Mininet 也可以模拟一个完整的网络主机、链接和交换机在同一台计算机上且有助于互动开发、测试和演示。

Mininet 实现的特性:

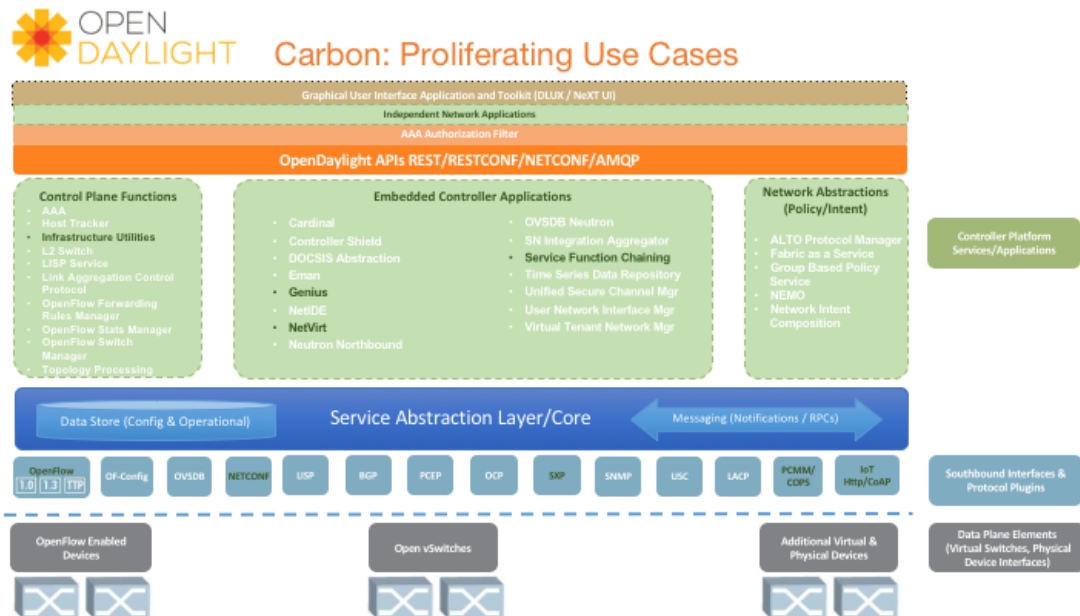
- 支持 OpenFlow、OpenvSwitch 等软定义网路部件
- 支持系统级的还原测试,支持复杂拓扑,自定义拓扑等
- 提供 Python API,方便多人协作开发
- 很好的硬件移植性与高扩展性
- 支持数千台主机的网络结构

2.2 开源 SDN 控制器：Opendaylight

就像操作系统为计算机的硬件提供接口一样,Opendaylight 也提供接口,允许使用者通过接口快速和智能化地连接网络设备,从而来提供最佳的网络性能。

OpenDaylight 不是一个单一的软件，当按步骤安装 OpenDaylight 后，还可以安装额外的 feature 来满足使用需求。

以 Carbon 版本为例，ODL 控制器系统整体架构如下图所示：（图片源自 ODL 官网）



从架构图可以看到，控制器主要由以下部分组成：

- 1、开放的北向 API：包括 REST/RESTCONF/NETCONF/AMQP
- 2、控制器平面：包括 ODL root Parent、Controller 等几十个工程
- 3、南向接口和协议插件：包括 OpenFlow、NETCONF 等

2.3 Ubuntu in VMware

```
root@sdnhubvm:/home/ubuntu# lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:   Ubuntu 14.04.3 LTS
Release:      14.04
Codename:     trusty
root@sdnhubvm:/home/ubuntu#
```

三、实验要求

比较 SDN 和经典路由的性能，其中经典路由方案是基于目的地址的路由选择，SDN 方案则是基于目的地址+源地址/应用类型/传输协议等的路由选择。具体要求如下：

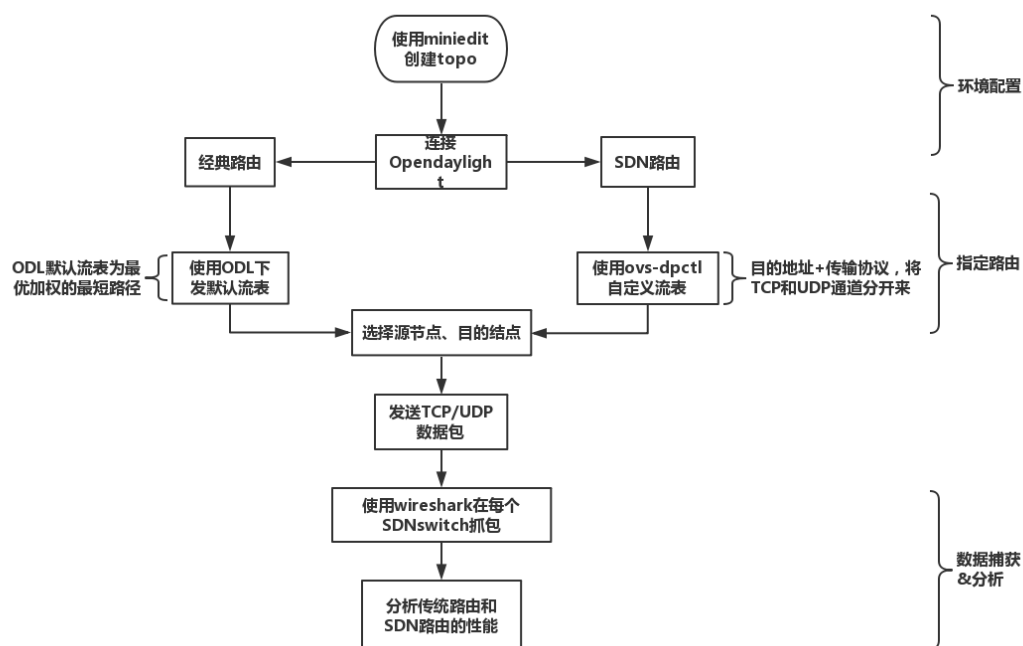
- 1.基于 mininet+controller 搭建 SDN 仿真网络；
- 2.选择部分节点作为源节点、目的节点；
- 3.从源节点发送数据给目的节点；
- 4.由 controller 根据策略决定路由；
- 5.在传输过程中，测量链路节点的时延、丢包、时延抖动、路径跳数、成本等 QoS 性能指标。

传统 TCP/IP 仅能根据目的地址决定下一跳节点，所以只要目的地址相同，不管是什么数据包，不管是什么源主机发的，都走相同路径；而 SDN 可以通过流表定制数据流的路径，使发往相同目的节点，不同协议的数据分别走不同的路径。我们要做的就是制定转发策略，和传统路由进行比较。

四、实验思路和基本原理

流程图：

主要为三个部分（环境配置、指定路由、数据捕获&分析，详细过程将在实验过程中叙述）



对于传统路由的仿真：

基于 Opendaylight 下发的初始流表。所有协议的数据包都根据目的地址，以最优权值最短路径算法获得下一跳，实现路由选择。

对于 SDN 路由的仿真：

基于 OVS 中含有匹配项的特殊流表。在 Opendaylight 控制器 C0 的控制下，根据每个 SDN 交换机中设置的流表，从各个主机发出的 TCP 和 UDP 协议的数据包各自拥有不同的传输路径，从而实现基于“目的地址+传输层协议通道”、“源地址+目的地址”双重约束的 SDN 路由选择。

数据捕获和分析：

1. 使用网络性能测试工具 iperf。iperf 可以测试最大 TCP 和 UDP 带宽性能，具有多种参数和 UDP 特性，可以根据需要调整，可以报告带宽、时延、延迟抖动和数据包丢失率。
2. 利用 wireshark 抓捕数据包进行分析。

五、实验过程（包含详细步骤）

1. OpenDaylight 的安装与运行

1.1 下载SDN Hub 网站提供的已经预装了一系列软件的Ubuntu 14.04.3 LTS 虚拟机文件，并使用 VMware 虚拟机打开。

64 位: http://yuba.stanford.edu/~srini/tutorial/SDN_tutorial_VM_64bit.ova

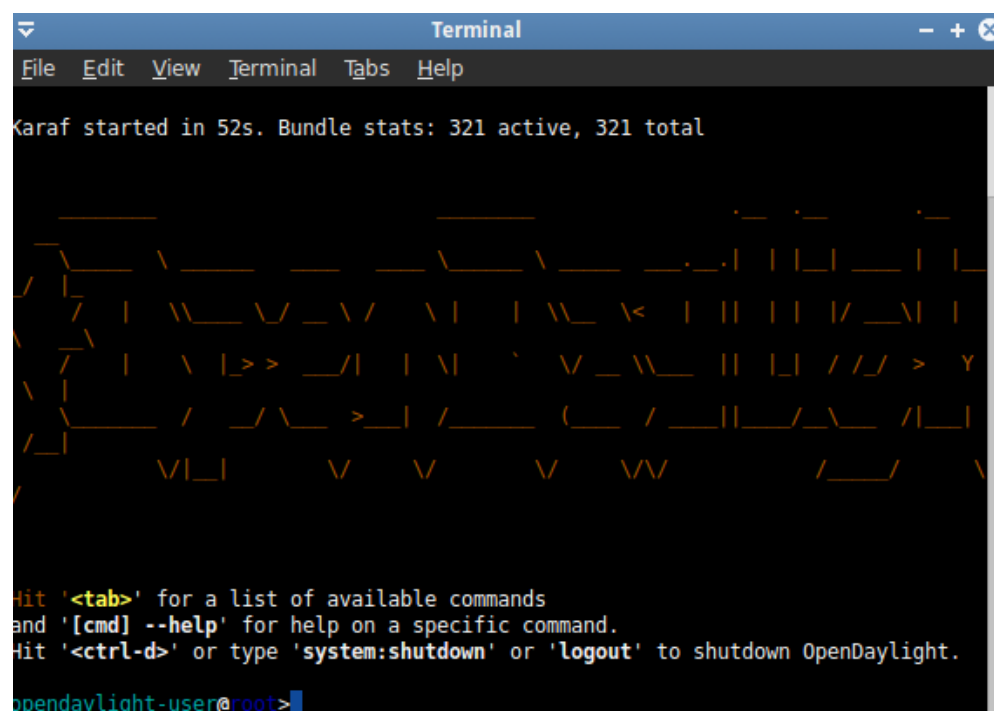
32 位: http://yuba.stanford.edu/~srini/tutorial/SDN_tutorial_VM_32bit.ova

1.2 下载 OpenDaylight 官方网站提供的预编译的压缩包，拷贝到虚拟机中，解压，运行。

实验中使用的版本为 0.5.2-Boron-SR2.zip，操作命令为：

1. unzip distribution-karaf-0.5.2-Boron-SR2.zip
2. cd /home/ubuntu/Desktop/distribution-karaf-0.5.2-Boron-SR2/bin/
3. sudo ./karaf

如下图所示：



```
Terminal
File Edit View Terminal Tabs Help

Karaf started in 52s. Bundle stats: 321 active, 321 total

      _____
     /         \
    /           \
   /             \
  /               \
 /                 \
/                   \
 \                   /
  \                 /
   \               /
    \             /
     \           /
      \         /
       \       /
        \     /
         \   /
          \ /
           /
          /
         /
        /
       /
      /
     /
    /
   /
  /
 /
/

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.

opendaylight-user@root>
```

第一次启动后需要安装相关的组件，Installation Guide 中列出了所有组件，并给出了对应的名称，如图所示：

OpenDaylight Documentation

Boron

Site

Page

« API

Release Notes »

Source

Search

Karaf OpenDaylight Features

Karaf OpenDaylight features

Feature Name	Feature Description	Karaf feature name	Compatibility
Authentication	Enables authentication with support for federation using Apache Shiro	odl-aaa-shiro	all
BGP	Provides support for Border Gateway Protocol (including Link-State Distribution) as a source of L3 topology information	odl-bgpcep-bgp	all
BMP	Provides support for BGP Monitoring Protocol as a monitoring station	odl-bgpcep-bmp	all
DIDM	Device Identification and Driver Management	odl-didm-all	all
Centinel	Provides interfaces for streaming analytics	odl-centinel-all	all
DLUX	Provides an intuitive graphical user interface for OpenDaylight	odl-dlux-all	all
Fabric as a Service (Faas)	Creates a common abstraction layer on top of a physical network so northbound APIs or services can be more easily mapped onto the physical network as a concrete device configuration	odl-faas-all	all
Group Based Policy	Enables Endpoint Registry and Policy Repository REST APIs and associated functionality for Group Based Policy with the default renderer for OpenFlow renderers	odl-groupbasedpolicy-ofoverlay	all
GBP User Interface	Enables a web-based user interface for Group Based Policy	odl-groupbasedpolicy-ui	all
GBP Faas renderer	Enables the Fabric as a Service renderer for Group Based Policy	odl-groupbasedpolicy-faas	self+all

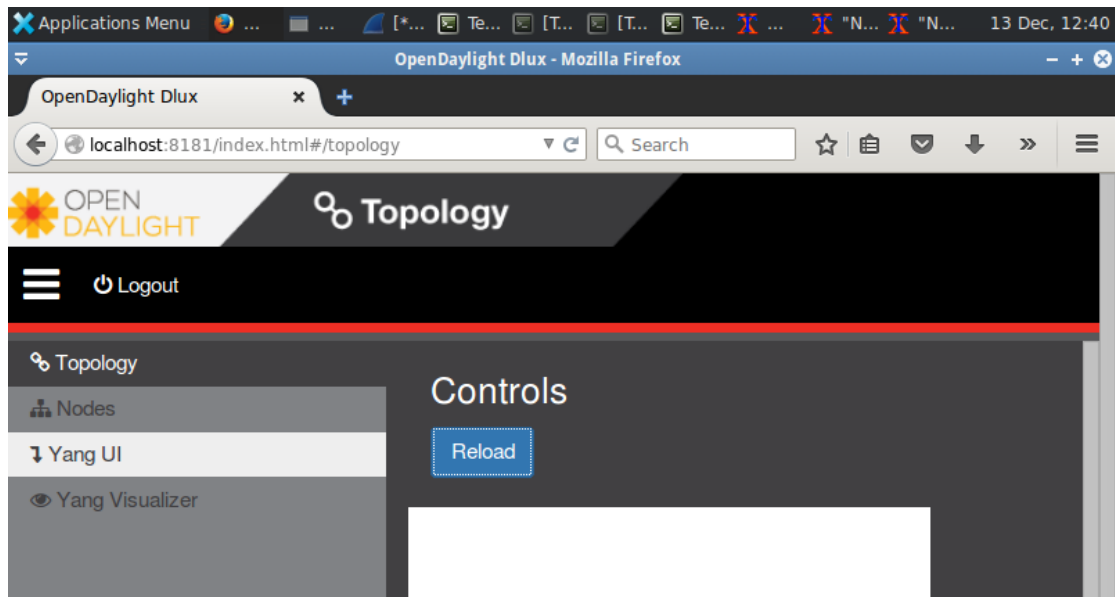
通过命令行安装组件：

- ```
1. opendaylight-user@root>feature:install odl-restconf
2. opendaylight-user@root>feature:install odl-l2switch-switch
3. opendaylight-user@root>feature:install odl-mdsal-apidocs
4. opendaylight-user@root>feature:install odl-l2switch-switch-ui
5. opendaylight-user@root>feature:install odl-dlux-all
6. opendaylight-user@root>feature:install odl-mdsal-all
7. opendaylight-user@root>feature:install odl-openflowplugin-flow-services-ui
```

安装完成后，即可打开浏览器，进入 ODL 的控制台：

- ```
1. http://localhost:8181/index.html
```

如图：



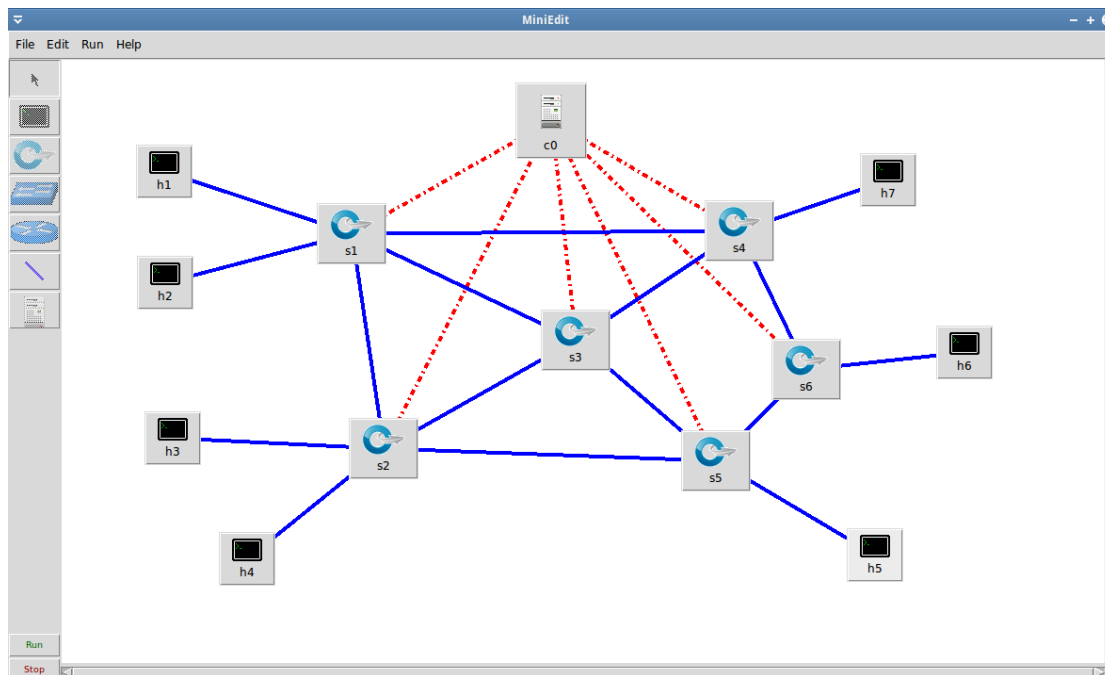
2. Mininet 的安装与运行

由于安装的 Ubuntu 虚拟机中预装了相关软件，故可直接找到所在目录启动。

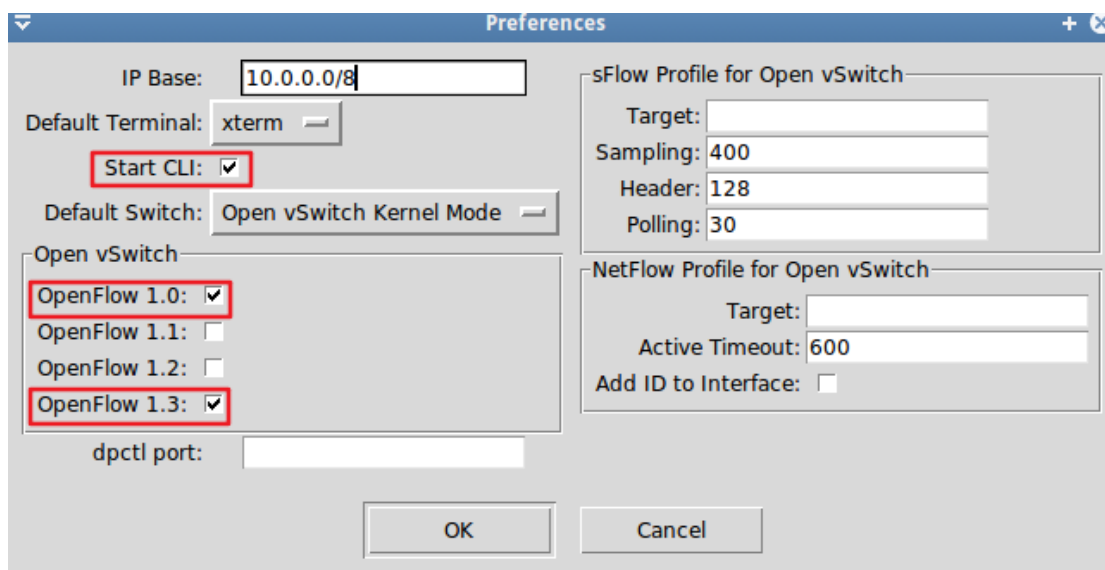
启动 Mininet 的图形界面：

```
1. cd /home/ubuntu/mininet/examples/  
2. sudo ./miniedit.py
```

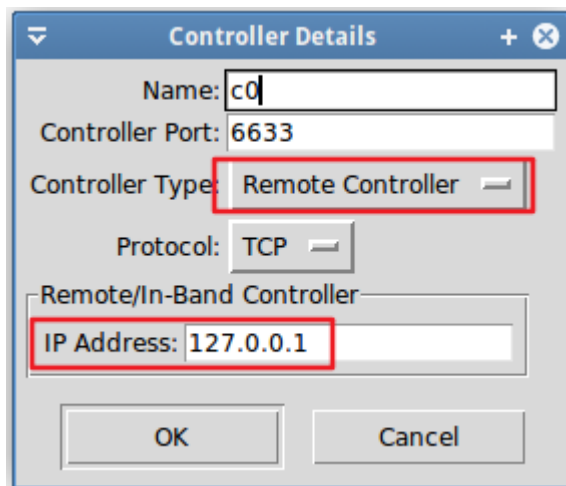
在界面界面中可添加控件，如图：



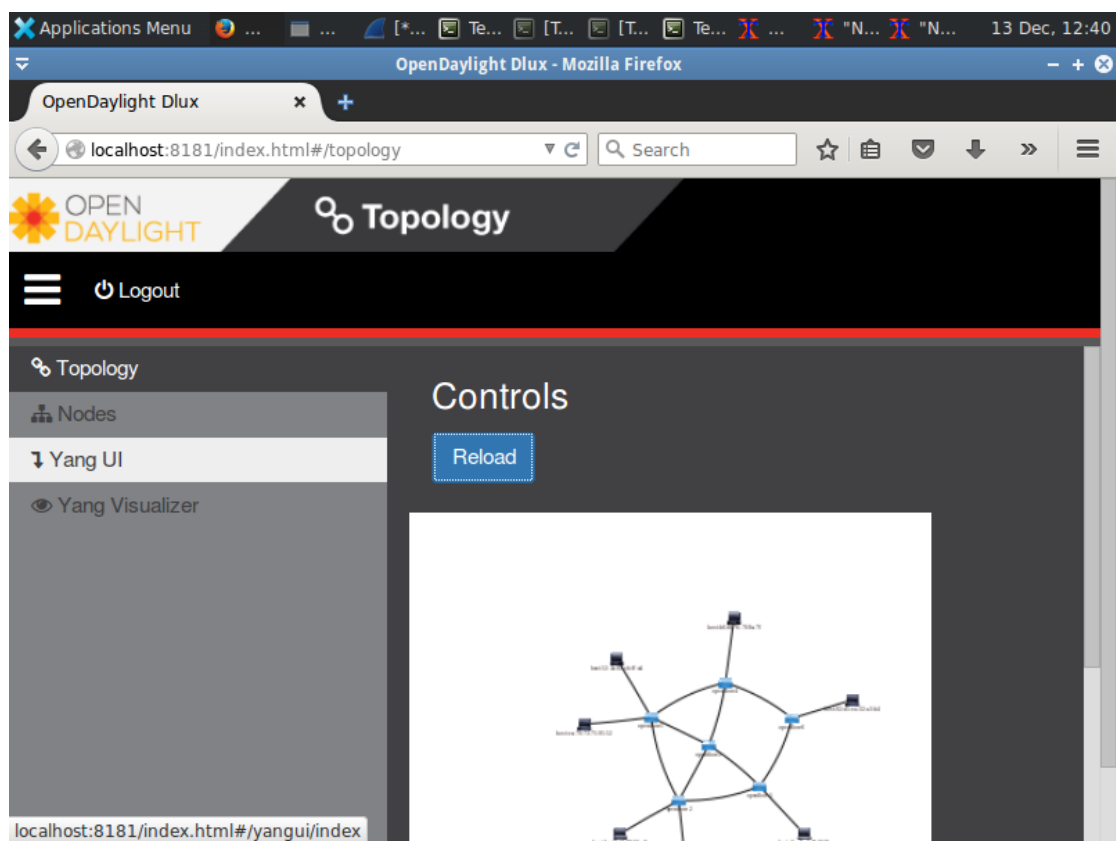
单击左上角的 Edit，在弹出的对话框中，勾选 Start CLI 和 OpenFlow 1.3，如图：



在控制器图标上，右击鼠标，选择 properties 后，弹出配置界面，将 Controller Type 改为 Remote Controller，将 IP Address 改为 ODL 所在的主机的 IP 地址，或者设置为 127.0.0.1，如图：



单机 Miniedit 界面中的 Run，然后刷新浏览器，即可看到交换机的图标：



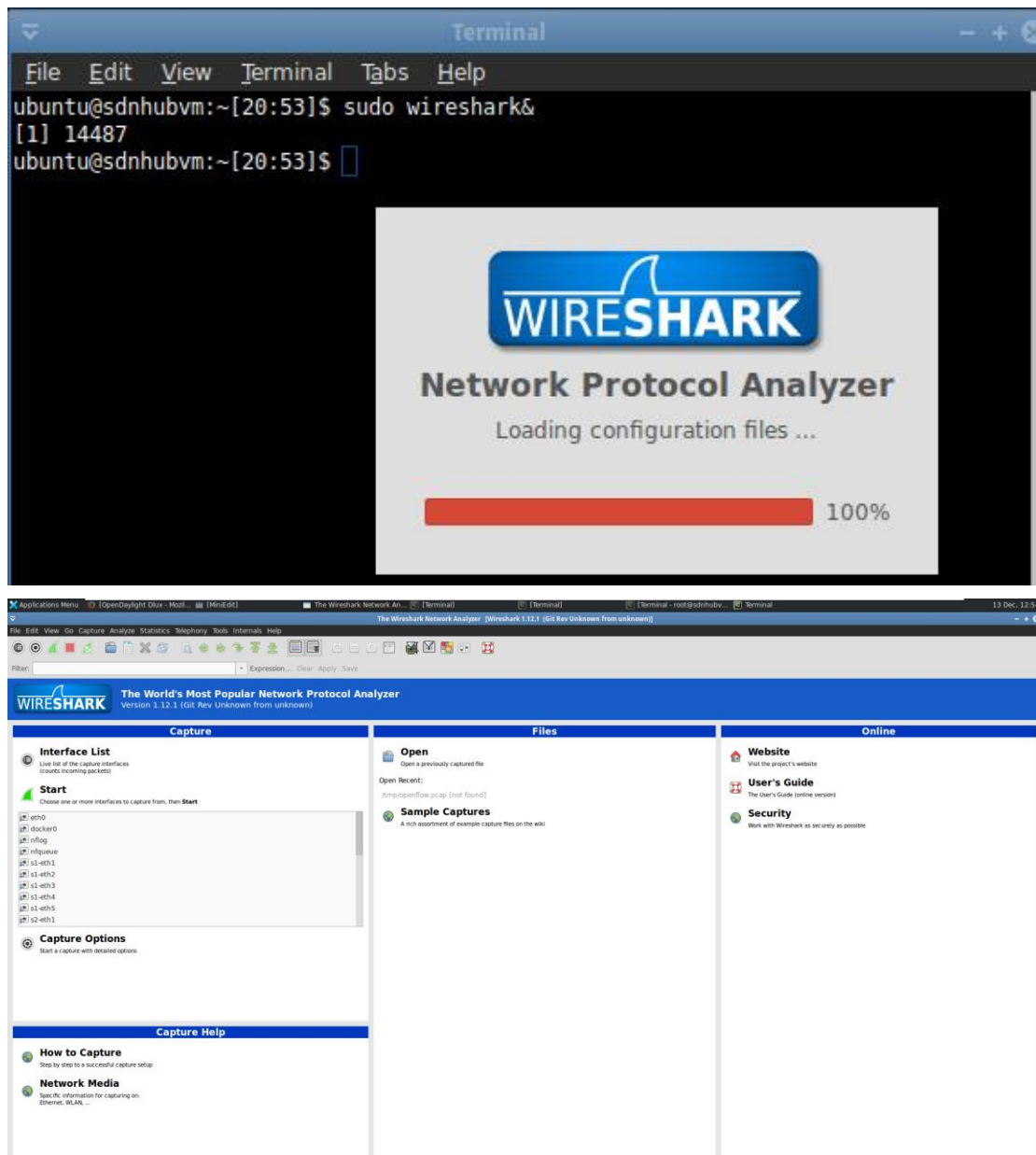
3. Wireshark 的安装与运行

由于 Ubuntu 14.04.3 LTS 中已经预装了 wireshark 软件，我们只需要打开即可，

命令行指令：

1. `sudo wireshark&`

如图：



4. 设置路由

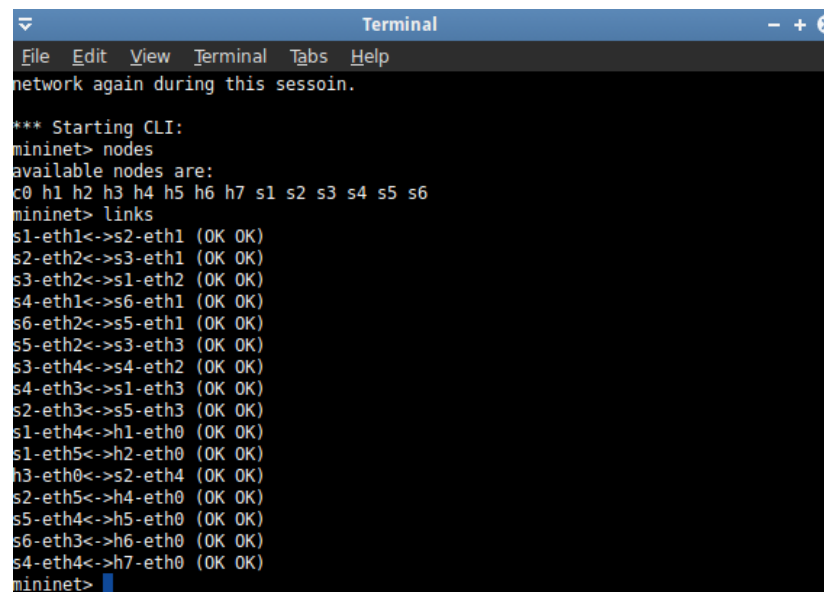
【说明】环境配置完成后，进行路由的设置和网络性能测试，包括“传统路由”和“SDN 路由”。下面分为两大部分进行：（一）传统路由以及（二）SDN 路由。

(一) 传统路由

4.1.1 在 mininet 窗口中输入指令查看结点以及链路状态，指令如下：

1. nodes
2. links

结果如下图所示：



```
Terminal
File Edit View Terminal Tabs Help
network again during this session.

*** Starting CLI:
mininet> nodes
available nodes are:
c0 h1 h2 h3 h4 h5 h6 h7 s1 s2 s3 s4 s5 s6
mininet> links
s1-eth1<->s2-eth1 (OK OK)
s2-eth2<->s3-eth1 (OK OK)
s3-eth2<->s1-eth2 (OK OK)
s4-eth1<->s6-eth1 (OK OK)
s6-eth2<->s5-eth1 (OK OK)
s5-eth2<->s3-eth3 (OK OK)
s3-eth4<->s4-eth2 (OK OK)
s4-eth3<->s1-eth3 (OK OK)
s2-eth3<->s5-eth3 (OK OK)
s1-eth4<->h1-eth0 (OK OK)
s1-eth5<->h2-eth0 (OK OK)
h3-eth0<->s2-eth4 (OK OK)
s2-eth5<->h4-eth0 (OK OK)
s5-eth4<->h5-eth0 (OK OK)
s6-eth3<->h6-eth0 (OK OK)
s4-eth4<->h7-eth0 (OK OK)
mininet>
```

4.1.2. 输入 pingall 发现能够 ping 通，连接成功，说明 OpenDaylight 控制器在 pingall 执行时下发了流表，我们视为传统路由，因为这是纯粹根据目的地址进行路由的，与发的数据包的数据包协议类型无关。指令如下：

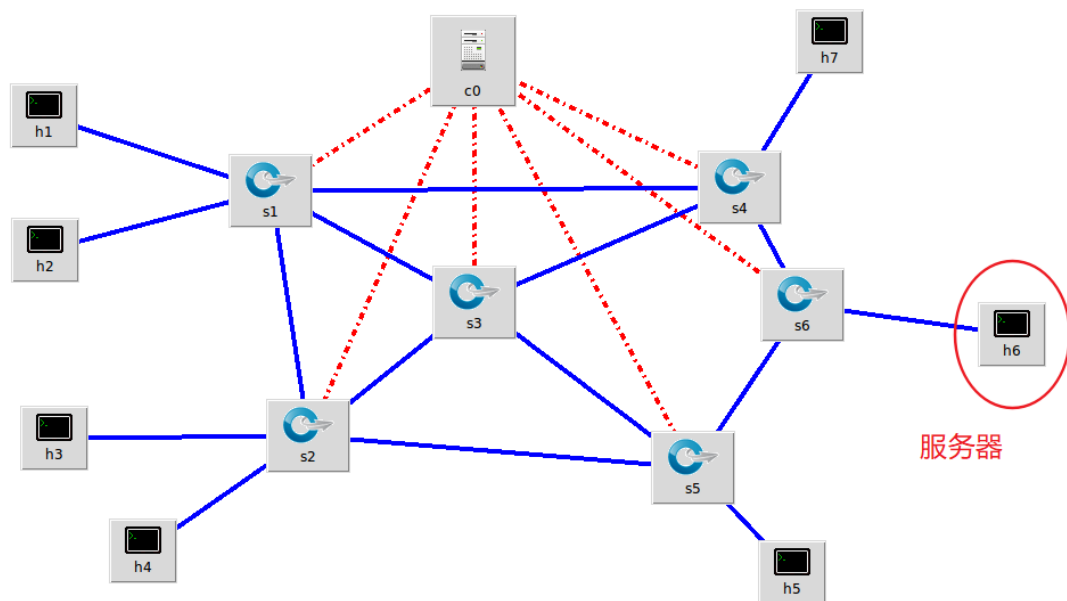
1. >pingall
2. >dpctl dump-flows

指令执行结果如图所示：

```
mininet> pingall
*** Ping: testing ping reachability
h5 -> h6 h3 h7 h1 h2 h4
h6 -> h5 h3 h7 h1 h2 h4
h3 -> h5 h6 h7 h1 h2 h4
h7 -> h5 h6 h3 h1 h2 h4
h1 -> h5 h6 h3 h7 h2 h4
h2 -> h5 h6 h3 h7 h1 h4
h4 -> h5 h6 h3 h7 h1 h2
*** Results: 0% dropped (42/42 received)
mininet>
```

```
mininet> dpctl dump-flows
*** s4 -----
NXST_FLOW reply (xid=0x4):
  cookie=0x2b00000000000006, duration=489.095s, table=0, n_packets=294, n_bytes=24990, idle_age=4, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
  cookie=0x2b00000000000001a, duration=484.895s, table=0, n_packets=157, n_bytes=10990, idle_age=338, priority=2,in_port=2 actions=output:4
  cookie=0x2b00000000000001b, duration=484.895s, table=0, n_packets=26, n_bytes=1820, idle_age=338, priority=2,in_port=4 actions=output:2,CONTROLLER:65535
  cookie=0x2b00000000000006, duration=489.157s, table=0, n_packets=16, n_bytes=1288, idle_age=480, priority=0 actions=drop
*** s5 -----
NXST_FLOW reply (xid=0x4):
  cookie=0x2b00000000000008, duration=489.076s, table=0, n_packets=293, n_bytes=24905, idle_age=4, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
  cookie=0x2b00000000000001c, duration=484.898s, table=0, n_packets=80, n_bytes=5600, idle_age=338, priority=2,in_port=2 actions=output:1,output:4,output:3
  cookie=0x2b00000000000001d, duration=484.895s, table=0, n_packets=28, n_bytes=1960, idle_age=338, priority=2,in_port=1 actions=output:2,output:4,output:3
  cookie=0x2b00000000000001e, duration=484.894s, table=0, n_packets=26, n_bytes=1820, idle_age=338, priority=2,in_port=4 actions=output:2,output:1,output:3,CONTROLLER:65535
  cookie=0x2b00000000000001f, duration=484.893s, table=0, n_packets=49, n_bytes=3430, idle_age=338, priority=2,in_port=3 actions=output:2,output:1,output:4
  cookie=0x2b000000000000008, duration=489.076s, table=0, n_packets=12, n_bytes=1000, idle_age=487, priority=0 actions=drop
*** s6 -----
NXST_FLOW reply (xid=0x4):
  cookie=0x2b00000000000007, duration=489.101s, table=0, n_packets=195, n_bytes=16575, idle_age=4, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
  cookie=0x2b000000000000020, duration=484.894s, table=0, n_packets=26, n_bytes=1820, idle_age=338, priority=2,in_port=3 actions=output:2,CONTROLLER:65535
  cookie=0x2b000000000000021, duration=484.893s, table=0, n_packets=157, n_bytes=10990, idle_age=338, priority=2,in_port=2 actions=output:3
```

4.1.3 利用 iperf 工具自行产生相应的测试数据。回顾我们刚刚创建好的 topo 图：



我们以 h6 (10.0.0.6) 作为服务器 (server)，以 h1 (10.0.0.1) 和 h3 (10.0.0.3) 作为两个测试的客户端 (client)，分别进行 UDP 和 TCP 的网络性能测试。

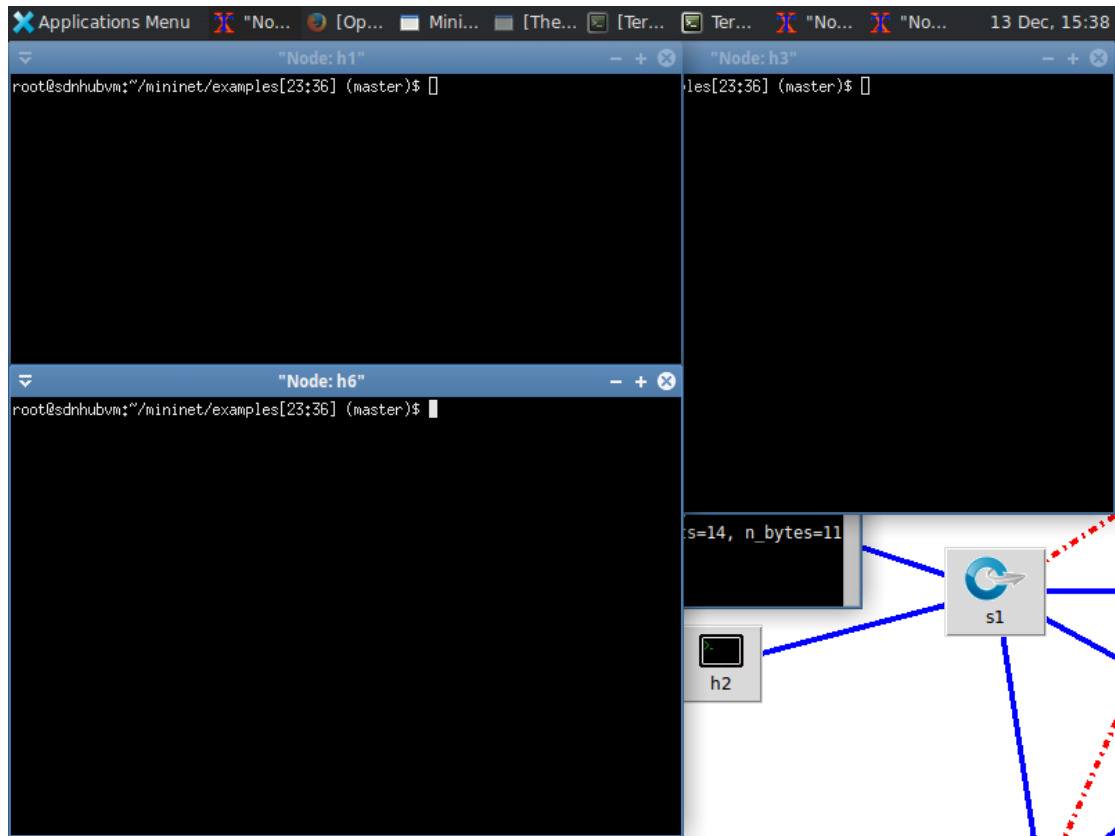
UDP 模式

带宽测试通常都会采用 UDP 模式，因为能测出极限带宽、时延抖动和丢包率。

① 在 Mininet 窗口使用 xterm 指令打开 h1,h3,h6 的窗口：

```
1. xterm h1 h3 h6
```

指令执行结果如图：



② 输入 iperf 指令

服务器端：

1. `iperf -u -s`

客户端：

1. `/*在 udp 模式下, 以 100Mbps 为数据发送速率, 客户端到服务器 192.168.1.1 上传带宽测试, 测试时间为 60 秒*/`
2. `iperf -u -c 192.168.1.1 -b 100M -t 60`
- 3.
4. `/*客户端以 5Mbps 为数据发送速率, 同时向服务器端发起 30 个连接线程*/`
5. `iperf -u -c 192.168.1.1 -b 5M -P 30 -t 60`
- 6.
7. `/*以 100M 为数据发送速率, 进行上下行带宽测试, -L 参数指定本端双测试监听的端口*/`
8. `iperf -u -c 192.168.1.1 -b 100M -d -t 60 -L 30000`

在 h6 窗口输入服务端的指令, 在 h1 和 h3 窗口输入客户端的指令, 如下图所示: h6 成为

服务器，并在 5001 端口监听 UDP 数据；

```
"Node: h6"
root@sdnhubvm:~/mininet/examples[23:36] (master)$ iperf -u -s
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
```

h1 成为客户端，并用 58935 端口连接上服务器，在 UDP 模式下，输入指令，以 100Mbps 为数据发送率，进行上传带宽测试，测试时间为 60s；

```
"Node: h1"
root@sdnhubvm:~/mininet/examples[23:36] (master)$ iperf -u -c 10.0.0.6 -b 100M
-t 60
-----
Client connecting to 10.0.0.6, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[  5] local 10.0.0.1 port 58935 connected with 10.0.0.6 port 5001
```

h6 服务器连接成功；

```
"Node: h6"
root@sdnhubvm:~/mininet/examples[23:36] (master)$ iperf -u -s
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[  5] local 10.0.0.6 port 5001 connected with 10.0.0.1 port 58935
```

等待 60s，获得结果；

```
-----
[  5] local 10.0.0.6 port 5001 connected with 10.0.0.1 port 58935
[ ID] Interval      Transfer    Bandwidth    Jitter    Lost/Total Datagrams
[  5] 0.0-60.0 sec   608 MBytes  85.0 Mbits/sec  0.033 ms 18559/452325 (4.1%)
[  5] 0.0-60.0 sec   1 datagrams received out-of-order
```

结果分析：

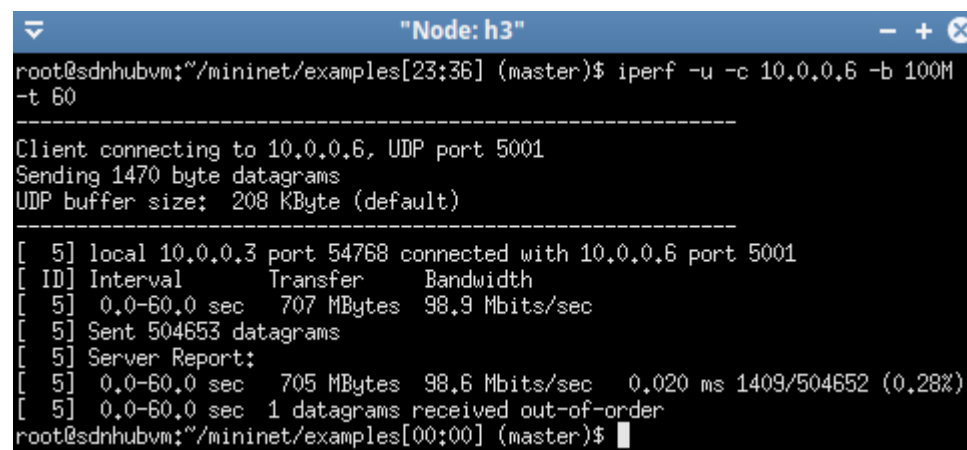
由此我们可以知道，在本次测试中，100Mbps 的测试带宽下，h1->h6 的时延抖动为 0.033ms，丢包率为 4.1%。接下来使用实际测出的 85Mbps 作为测试带宽，重新进行测试，

得到结果：

```
[ 6] local 10.0.0.6 port 5001 connected with 10.0.0.1 port 50714
[ 6] 0.0-60.0 sec  580 MBytes  81.1 Mbits/sec  0.024 ms 4409/418171 (1.1%)
[ 6] 0.0-60.0 sec  1 datagrams received out-of-order
```

发现时延抖动为 0.024ms，丢包率为 1.1%，比 100Mbps 时好很多。

同理获得 h3 发送的 UDP 数据包的结果如下图所示：



```
root@sdnhubvm:~/mininet/examples[23:36] (master)$ iperf -u -c 10.0.0.6 -b 100M -t 60
-----
Client connecting to 10.0.0.6, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[ 5] local 10.0.0.3 port 54768 connected with 10.0.0.6 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 5] 0.0-60.0 sec  707 MBytes  98.9 Mbits/sec
[ 5] Sent 504653 datagrams
[ 5] Server Report:
[ 5] 0.0-60.0 sec  705 MBytes  98.6 Mbits/sec  0.020 ms 1409/504652 (0.28%)
[ 5] 0.0-60.0 sec  1 datagrams received out-of-order
root@sdnhubvm:~/mininet/examples[00:00] (master)$
```

服务器信息：

```
[ 5] local 10.0.0.6 port 5001 connected with 10.0.0.3 port 54768
[ 5] 0.0-60.0 sec  705 MBytes  98.6 Mbits/sec  0.021 ms 1409/504652 (0.28%)
[ 5] 0.0-60.0 sec  1 datagrams received out-of-order
```

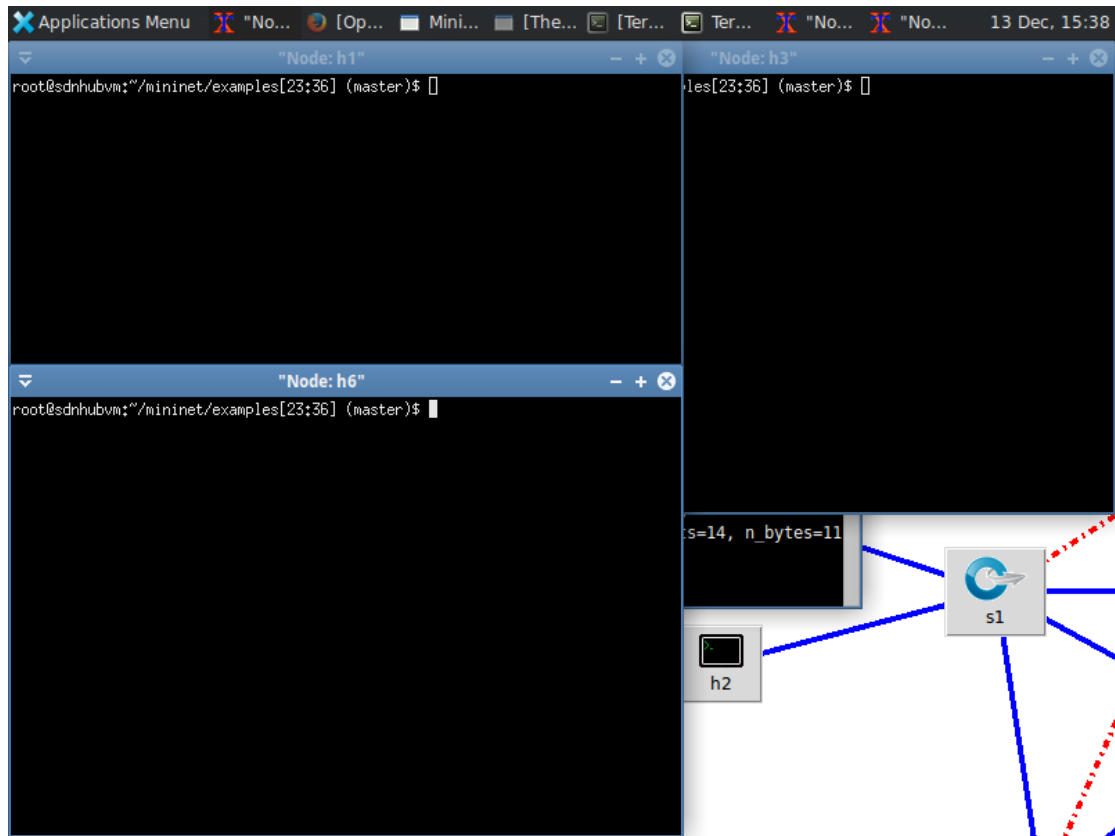
由此我们可以知道，在 100Mbps 的测试带宽下，h3->h6 的时延抖动为 0.021ms，丢包率为 0.28%。测出的实际带宽为 98.6Mbps，丢包率和实际带宽都十分可观，因此我们可以认为 h3 到 h6 的实际带宽为 100Mbps 左右。

TCP 模式

① 在 Mininet 窗口使用 xterm 指令打开 h1,h3,h6 的窗口：

```
1. xterm h1 h3 h6
```

指令执行结果如图：



② 输入 iperf 指令

服务器端：

1. `iperf -s`

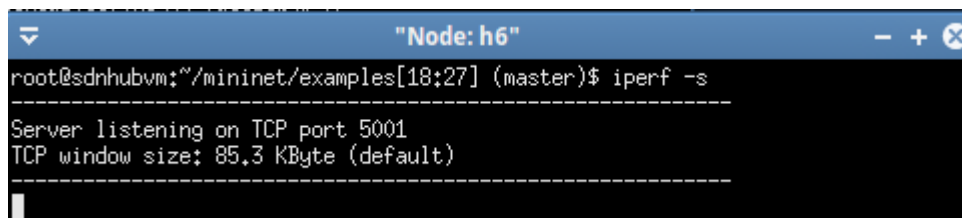
客户端：

1. /*在 tcp 模式下，客户端到服务器 192.168.1.1 上传带宽测试，测试时间为 60 秒*/
2. `iperf -c 192.168.1.1 -t 60`
- 3.
4. /*进行上下行带宽测试*/
5. `iperf -c 192.168.1.1 -d -t 60`
- 6.
7. /*测试多线程 TCP*/
8. `iperf -c 192.168.1.1 -p 12345 -i 1 -t 10 -w 20K`
- 9.
10. -c: 客户端模式，后接服务器 ip
11. -p: 后接服务端监听的端口
12. -i: 设置带宽报告的时间间隔，单位为秒
13. -t: 设置测试的时长，单位为秒

```
14. -w: 设置 tcp 窗口大小, 一般可以不用设置, 默认即可
15.
16. 对应服务器端:
17. iperf -s -p 12345 -i 1 -t 10 -m -y
18.
19. /*测试多线程 TCP: 客户端同时向服务器端发起 30 个连接线程*/
20. iperf -c 192.168.1.1 -P 30 -t 60
```

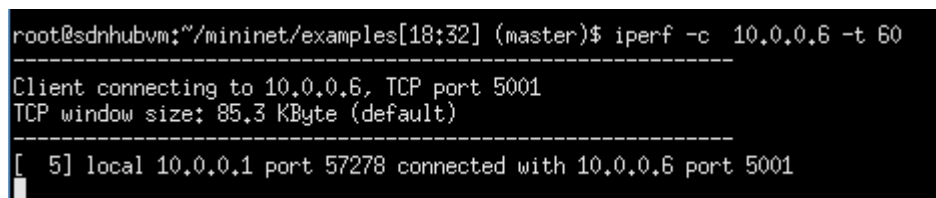
发包完成后, 可以通过 `ifconfig ethx` 和 `ethtool -S ethx` 查看对应收发包情况, 确定发包数、包场、是否丢包等。

在服务器端 h6 输入对应的指令, 如下图所示:



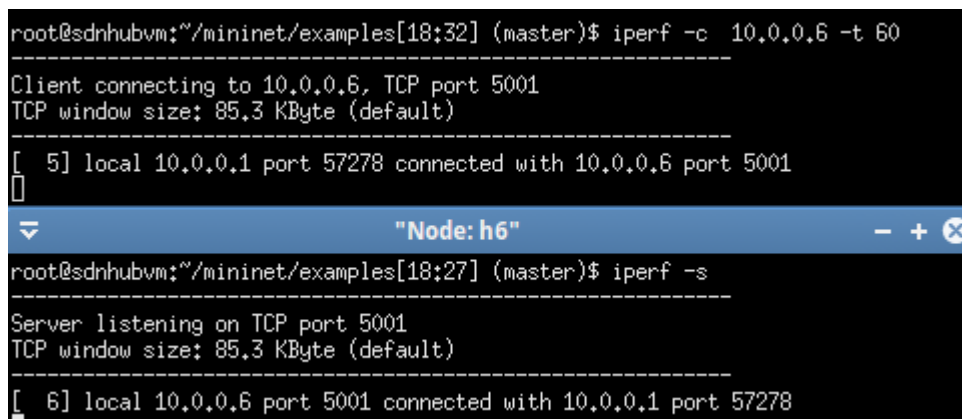
```
root@sdnhubvm:~/mininet/examples[18;27] (master)$ iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
```

在客户端 h1 输入对应的指令, 如下图所示:



```
root@sdnhubvm:~/mininet/examples[18;32] (master)$ iperf -c 10.0.0.6 -t 60
-----
Client connecting to 10.0.0.6, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 5] local 10.0.0.1 port 57278 connected with 10.0.0.6 port 5001
-----
```

在 h6 和 h1 的窗口均有显示连接成功:



```
root@sdnhubvm:~/mininet/examples[18;32] (master)$ iperf -c 10.0.0.6 -t 60
-----
Client connecting to 10.0.0.6, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 5] local 10.0.0.1 port 57278 connected with 10.0.0.6 port 5001
-----

root@sdnhubvm:~/mininet/examples[18;27] (master)$ iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 6] local 10.0.0.6 port 5001 connected with 10.0.0.1 port 57278
-----
```

等待 60s, 获得结果:

```
[ 6] local 10.0.0.6 port 5001 connected with 10.0.0.1 port 57278
[ ID] Interval      Transfer      Bandwidth
[ 6] 0.0-60.2 sec  1.15 GBytes  165 Mbits/sec
```

结果分析:

可以看到 interval 选项中的 “0-60.2sec”, h1->h6 的 TCP 传输的时延约为 200ms, 比 UDP 时延大许多, 这是正常的, 因为 TCP 在处理交互数据流时, 采用了 Delayed Ack 机制以及 Nagle 算法来减少小分组数目, 故而导致了 TCP 延时。除此之外, 还能知道: 在 60s 的时间内, h1->h6 传输了 1.15GB 的数据, 带宽为 165Mbps, 比 UDP 高不少。

关于丢包情况, 我们利用 ifconfig 指令, 在 h1 窗口中输入 “ifconfig”, 获得如下所示:

```
root@sdnhubvm:~/mininet/examples[18:35] (master)$ ifconfig
h1-eth0  Link encap:Ethernet  HWaddr ee:58:df:db:47:58
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::ec58:dfdf:fedb:4758/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:407833 errors:0 dropped:223 overruns:0 frame:0
          TX packets:92945 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:27008751 (27.0 MB)  TX bytes:1251878298 (1.2 GB)

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:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

其中 RX 为接收, TX 为发送数据包情况统计。可知 h1 在接收时接收了 407833 个数据包, 丢失了 223 个数据包, 丢包率为 0.055%。

同理获得 h3->h6 的 TCP 传输结果:

```
[ 34] local 10.0.0.6 port 5001 connected with 10.0.0.3 port 53609
[ 34] 0.0-60.3 sec  1.18 GBytes  168 Mbits/sec
```

同时在 h3 窗口输入 “ifconfig” 获得数据包情况信息:

```

root@sdnhubvm:~/mininet/examples[18:50] (master)$ ifconfig
h3-eth0  Link encap:Ethernet  HWaddr 76:84:24:36:bf:06
          inet addr:10.0.0.3  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::7484:24ff:fe36:bf06/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1683983 errors:0 dropped:334 overruns:0 frame:0
          TX packets:89488 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1350076412 (1.3 GB)  TX bytes:1275393968 (1.2 GB)

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:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

```

可以知道, h3->h6 的时延约为 300ms, 在 60s 时间内, 传输了 1.18GB, 带宽为 168Mbps,

h3 接口接收了 1683983 个数据包, 其中丢失了 334 个, 丢包率为 0.02%。

(二) SDN 路由

我的设计思路是：作为源节点的 h1 和 h3, 他们分别发送 TCP 和 UDP 数据包, 在传统路由中无论是任何协议类型的包都会从同一条路径传送, 而我则将它们区分开来。具体做法就是采取“源地址+目的地址”、“目的地址+传输协议类型”的双重约束, 最终效果是 TCP 和 UDP 的传输线路是不同的, h1 和 h3 的传输路径也是不同的。

具体步骤如下所示。

4.2.1 首先将 ODL 下发的流表删除掉, 指令:

```
1. dpctl del-flows
```

如下图所示:

```

mininet> dpctl del-flows
*** s4 -----
*** s5 -----
*** s6 -----
*** s2 -----
*** s1 -----
*** s3 -----
mininet>

```

4.2.2 另开一个控制台，进入 root 权限，并查看网桥和端口情况，验证与 topo 图是否相

符：

1. `sudo su`
2. `ovs-vsctl show`

如下图所示：

```
ubuntu@sdnhubvm:~[19:01]$ sudo su
root@sdnhubvm:/home/ubuntu# ovs-vsctl show
873c293e-912d-4067-82ad-d1116d2ad39f
    Bridge "s3"
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        Port "s3-eth4"
            Interface "s3-eth4"
        Port "s3-eth3"
            Interface "s3-eth3"
        Port "s3"
            Interface "s3"
            type: internal
        Port "s3-eth2"
            Interface "s3-eth2"
        Port "s3-eth1"
            Interface "s3-eth1"
    Bridge "s4"
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        Port "s4-eth4"
            Interface "s4-eth4"
```

查看各交换机的流表，由于之前删除了流表，应为空，指令：

1. `ovs-ofctl dump-flows br0`

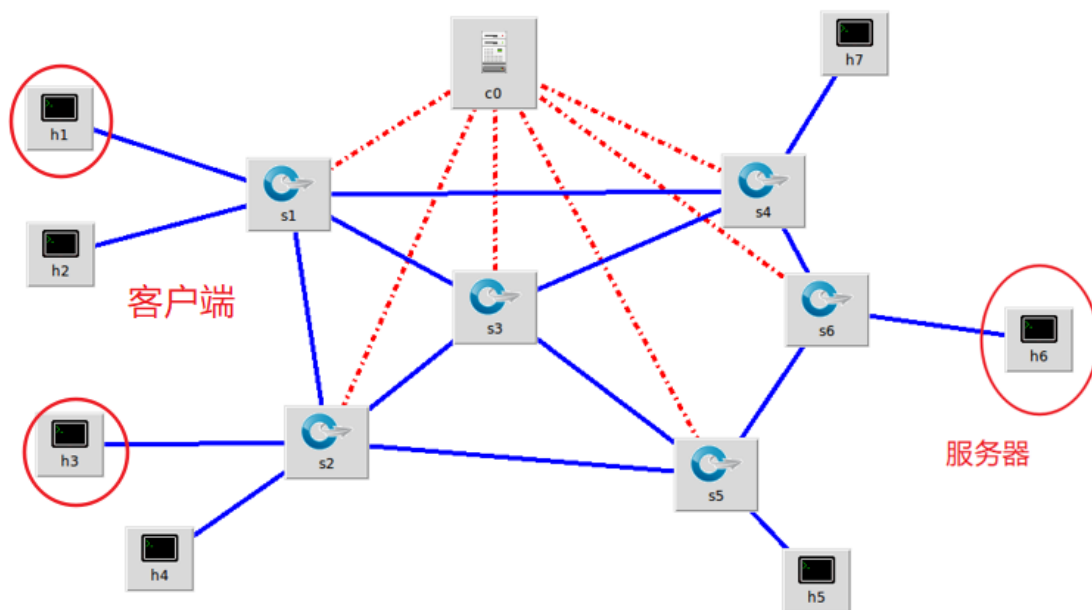
如下图所示：

```

root@sdnhubvm:/home/ubuntu# ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
root@sdnhubvm:/home/ubuntu# ovs-ofctl dump-flows s2
NXST_FLOW reply (xid=0x4):
root@sdnhubvm:/home/ubuntu# ovs-ofctl dump-flows s3
NXST_FLOW reply (xid=0x4):
root@sdnhubvm:/home/ubuntu# ovs-ofctl dump-flows s4
NXST_FLOW reply (xid=0x4):
root@sdnhubvm:/home/ubuntu# ovs-ofctl dump-flows s5
NXST_FLOW reply (xid=0x4):
root@sdnhubvm:/home/ubuntu# ovs-ofctl dump-flows s6
NXST_FLOW reply (xid=0x4):

```

4.2.3 通过 OVS 指令下发流表，回顾我们的 topo 图：



我们设置的思路是将 TCP 线路和 UDP 线路分开，为了方便我们下发流表，我们需要知道各个交换机的各个端口的连接情况，通过在 Mininet 窗口输入指令：

```
1. net
```

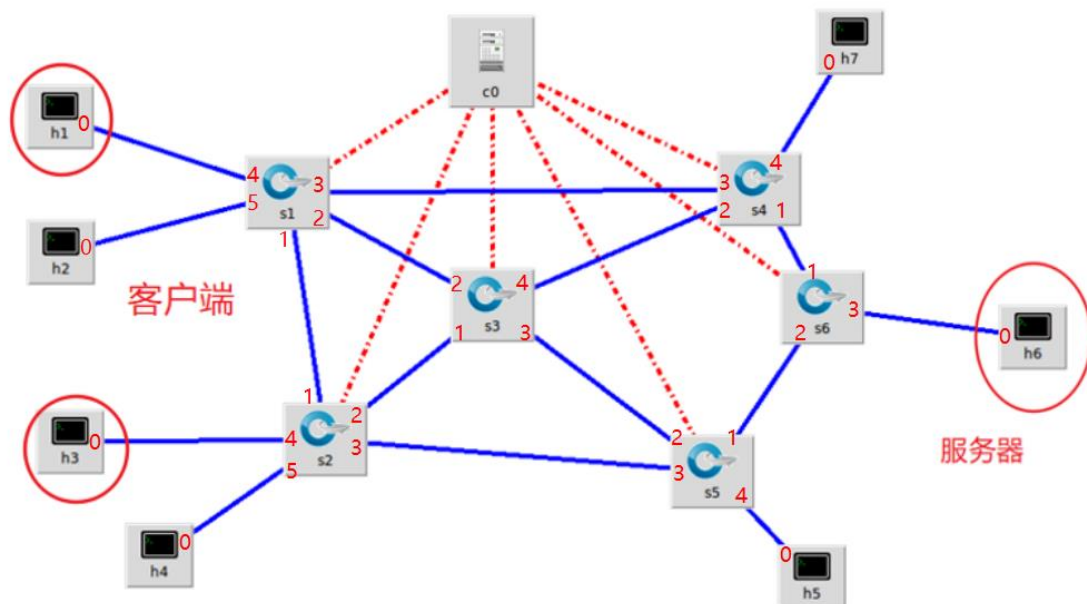
如下图所示：

```

mininet> net
h5 h5-eth0:s5-eth4
h6 h6-eth0:s6-eth3
h3 h3-eth0:s2-eth4
h7 h7-eth0:s4-eth4
h1 h1-eth0:s1-eth4
h2 h2-eth0:s1-eth5
h4 h4-eth0:s2-eth5
s4 lo: s4-eth1:s6-eth1 s4-eth2:s3-eth4 s4-eth3:s1-eth3 s4-eth4:h7-eth0
s5 lo: s5-eth1:s6-eth2 s5-eth2:s3-eth3 s5-eth3:s2-eth3 s5-eth4:h5-eth0
s6 lo: s6-eth1:s4-eth1 s6-eth2:s5-eth1 s6-eth3:h6-eth0
s2 lo: s2-eth1:s1-eth1 s2-eth2:s3-eth1 s2-eth3:s5-eth3 s2-eth4:h3-eth0 s2-eth5:
h4-eth0
s1 lo: s1-eth1:s2-eth1 s1-eth2:s3-eth2 s1-eth3:s4-eth3 s1-eth4:h1-eth0 s1-eth5:
h2-eth0
s3 lo: s3-eth1:s2-eth2 s3-eth2:s1-eth2 s3-eth3:s5-eth2 s3-eth4:s4-eth2
c0

```

就可以十分清楚的知道每个端口的连接情况，通过标记，我们获得以下 topo 图：



设计 h1 和 h3 的 TCP/UDP 线路为：

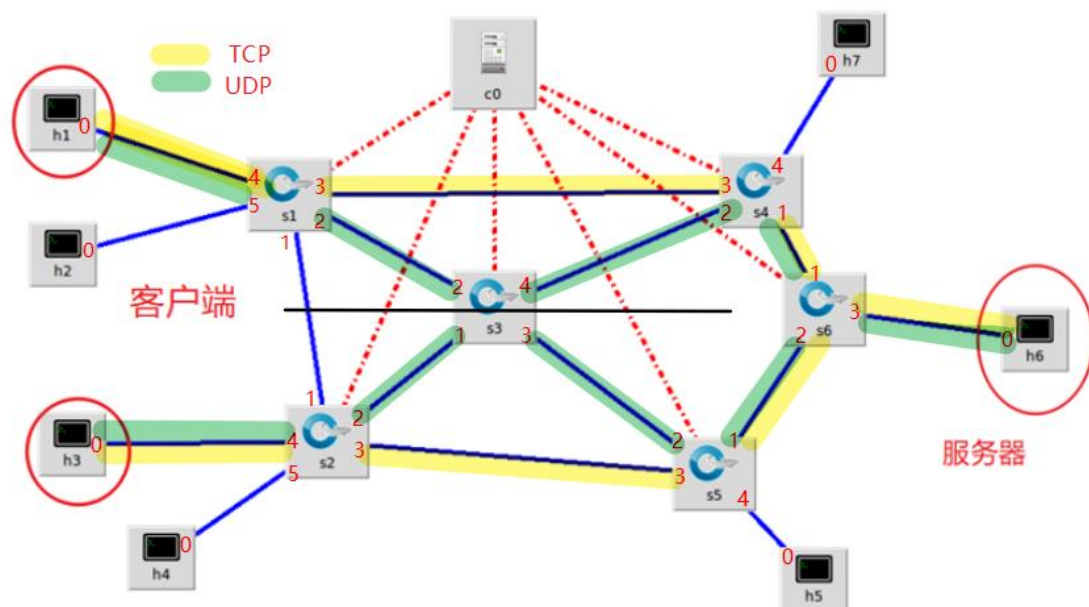
【h1-TCP】 h1->s1->s4->s6->h6

【h1-UDP】 h1->s1->s3->s4->s6->h6

【h3-TCP】 h3->s2->s5->s6->h6

【h3-UDP】 h3->s2->s3->s5->s6->h6

改善后的 topo 图如下所示：



关于 ping

首先我们需要确保 $h1 \leftrightarrow h6$ 、 $h3 \leftrightarrow h6$ 是畅通的，也就是能够 ping 通。

通过指令：

```
1. ovs-ofctl add-flow s1 dl_type=0x0800,nw_proto=1,in_port=3,actions=output:4
```

在 s1 中添加一个关于 icmp 协议的流表，其中 $dl_type=0x0800, nw_proto=1$ 是 icmp 的标识。这里要注意需要添加双向的流表，以上指令为例，还需要输入：

```
1. ovs-ofctl add-flow s1 dl_type=0x0800,nw_proto=1,in_port=4,actions=output:3
```

才能实现 icmp 协议数据包的在 s1 的传输。同时回顾 ping 过程中有转发 ARP 包，因此我们也需要设置 ARP 协议的流表，指令和上面 icmp 类似，具体区别可查下表。

速记符	匹配项
ip	dl_type=0x800
ipv6	dl_type=0x86dd
icmp	dl_type=0x0800,nw_proto=1
icmp6	dl_type=0x86dd,nw_proto=58
tcp	dl_type=0x0800,nw_proto=6
tcp6	dl_type=0x86dd,nw_proto=6
udp	dl_type=0x0800,nw_proto=17
udp6	dl_type=0x86dd,nw_proto=17
sctp	dl_type=0x0800,nw_proto=132
sctp6	dl_type=0x86dd,nw_proto=132
arp	dl_type=0x0806
rarp	dl_type=0x8035
mpls	dl_type=0x8847
mplsm	dl_type=0x8848

表-1

```

Terminal - root@sdnhubvm: /home/ubuntu
File Edit View Terminal Tabs Help
ort=4,actions=output:3
root@sdnhubvm:/home/ubuntu# ovs-ofctl add-flow s1 dl_type=0x0800,nw_proto=1,in_p
ort=3,actions=output:4
root@sdnhubvm:/home/ubuntu# ovs-ofctl add-flow s4 dl_type=0x0800,nw_proto=1,in_p
ort=3,actions=output:1
root@sdnhubvm:/home/ubuntu# ovs-ofctl add-flow s4 dl_type=0x0800,nw_proto=1,in_p
ort=1,actions=output:3
root@sdnhubvm:/home/ubuntu# ovs-ofctl add-flow s6 dl_type=0x0800,nw_proto=1,in_p
ort=1,actions=output:3
root@sdnhubvm:/home/ubuntu# ovs-ofctl add-flow s6 dl_type=0x0800,nw_proto=1,in_p
ort=3,actions=output:1
root@sdnhubvm:/home/ubuntu# ovs-ofctl add-flow s1 dl_type=0x0806,in_port=3,actio
ns=output:4
root@sdnhubvm:/home/ubuntu# ovs-ofctl add-flow s4 dl_type=0x0806,in_port=1,actio
ns=output:3
root@sdnhubvm:/home/ubuntu# ovs-ofctl add-flow s6 dl_type=0x0806,in_port=3,actio
ns=output:1
root@sdnhubvm:/home/ubuntu# ovs-ofctl add-flow s6 dl_type=0x0806,in_port=1,actio
ns=output:3
root@sdnhubvm:/home/ubuntu# ovs-ofctl add-flow s4 dl_type=0x0806,in_port=3,actio
ns=output:1
root@sdnhubvm:/home/ubuntu# ovs-ofctl add-flow s1 dl_type=0x0806,in_port=4,actio
ns=output:3
root@sdnhubvm:/home/ubuntu#

```

最后我们成功 ping 通 h1 和 h6! 如下图:

```
mininet> h1 ping h6
PING 10.0.0.6 (10.0.0.6) 56(84) bytes of data:
64 bytes from 10.0.0.6: icmp_seq=1 ttl=64 time=0.049 ms
64 bytes from 10.0.0.6: icmp_seq=2 ttl=64 time=0.061 ms
64 bytes from 10.0.0.6: icmp_seq=3 ttl=64 time=0.061 ms
^C
--- 10.0.0.6 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.049/0.207/0.513/0.216 ms
```

接下来是四条线路分配流表的过程，我们按路线分为四个部分叙述：

h1-TCP

回顾 h1 传输 TCP 数据包到服务器 h6 的路线为 “h1->s1->s4->s6->h6”。

【实现思路】

在线路上的交换机设置匹配项流表，匹配项要求为 TCP 协议，查表-1 可知为 “dl_type=0x0800,nw_proto=6”，注意双向设置流表。

【注意事项】

由于本次实验 topo 图的特殊性，在连接服务器端 h6 的交换机 s6 是两条 TCP 测试通路的交汇处，我们需要添加 nw_src 和 nw_dst 约束区分开这两条 TCP 流表项。

【结果】

用 “xterm h1 h6” 指令打开 h1、h6 终端，利用 4.1.3 中描述过的 iperf 命令进行 TCP 连接传输，如下图所示：

```
"Node: h6"
root@sdnhubvm:~/mininet/examples[07:19] (master)$ iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 6] local 10.0.0.6 port 5001 connected with 10.0.0.1 port 51146
[ ID] Interval      Transfer    Bandwidth
[ 6] 0.0-10.0 sec  29.7 GBytes 25.5 Gbits/sec

"Node: h1"
root@sdnhubvm:~/mininet/examples[07:19] (master)$ iperf -c 10.0.0.6 -t 10
-----
Client connecting to 10.0.0.6, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 5] local 10.0.0.1 port 51146 connected with 10.0.0.6 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 5] 0.0-10.0 sec  29.7 GBytes 25.5 Gbits/sec
root@sdnhubvm:~/mininet/examples[07:20] (master)$
```

在 h1->h6 的 TCP 测试传输 10s 时间中，传输 29.7GB 数据，带宽为 25.5Gbps。

在 h1 中输入 “ifconfig” 查看：

```
root@sdnhubvm:~/mininet/examples[07:20] (master)$ ifconfig
h1-eth0  Link encap:Ethernet HWaddr 8e:ee:61:5a:f5:63
          inet addr:10.0.0.1 Bcast:10.255.255.255 Mask:255.0.0.0
          inet6 addr: fe80::8cee:61ff:fe5a:f563/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1072877 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2404575 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:70903242 (70.9 MB) TX bytes:143511900310 (143.5 GB)
```

可以知道，这次 TCP 传输中，h1 客户端接收 1072877 个数据包，发送 2404575 个数据包，丢包率为 0%。

h3-TCP

回顾 h3 传输 TCP 数据包到服务器 h6 的路线为 “h3->s2->s5->s6->h6”。

【实现思路】

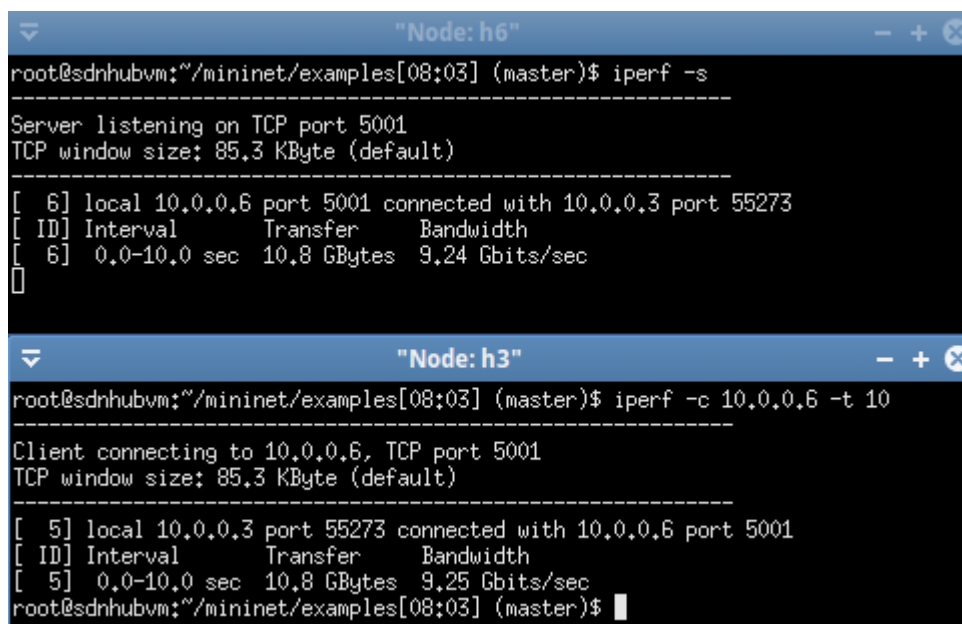
【注意事项】

实现思路以及注意事项和 h1-TCP 是一样的，这里不赘述。

【结果】

用 “xterm h3 h6” 指令打开 h3、h6 终端，利用 4.1.3 中描述过的 iperf 命令进行 TCP 连

接传输，如下图所示：

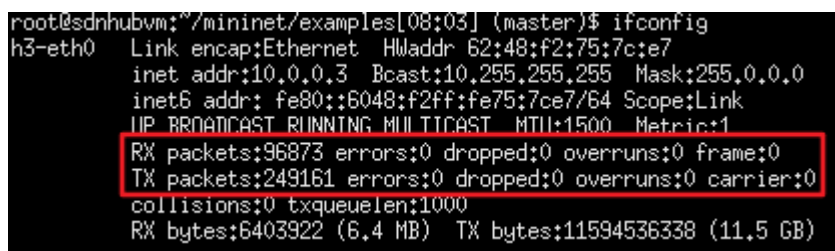


```
root@sdnhubvm:~/mininet/examples[08:03] (master)$ iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[  6] local 10.0.0.6 port 5001 connected with 10.0.0.3 port 55273
[ ID] Interval      Transfer    Bandwidth
[  6]  0.0-10.0 sec  10.8 GBytes  9.24 Gbits/sec
[ ]

root@sdnhubvm:~/mininet/examples[08:03] (master)$ iperf -c 10.0.0.6 -t 10
-----
Client connecting to 10.0.0.6, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[  5] local 10.0.0.3 port 55273 connected with 10.0.0.6 port 5001
[ ID] Interval      Transfer    Bandwidth
[  5]  0.0-10.0 sec  10.8 GBytes  9.25 Gbits/sec
root@sdnhubvm:~/mininet/examples[08:03] (master)$
```

在 h3->h6 的 TCP 测试传输 10s 时间中，传输 10.8GB 数据，带宽为 9.25Gbps。

在 h3 中输入 “ifconfig” 查看：



```
root@sdnhubvm:~/mininet/examples[08:03] (master)$ ifconfig
h3-eth0  Link encap:Ethernet  HWaddr 62:48:f2:75:7c:e7
          inet addr:10.0.0.3  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::6048:f2ff:fe75:7ce7/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:96873 errors:0 dropped:0 overruns:0 frame:0
          TX packets:249161 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:6403922 (6.4 MB)  TX bytes:11594536338 (11.5 GB)
```

可以知道，这次 TCP 传输中，h3 客户端接收 96873 个数据包，发送 249161 个数据包，丢包率为 0%。

为了验证是否如实按照 topo 图中的 TCP 路径进行传输，我们使用 wireshark 在 h1->h6 时对 s6-eth2、s6-eth3 进行抓包分析，如下图所示：



Interface List

Live list of the capture interfaces
(counts incoming packets)



Start

Choose one or more interfaces to capture from, then **Start**



s6-eth2 没有任何 TCP 包:

```
root@sdnhubvm:~/mininet/examples[08:03] (master)$ iperf -s
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)

[ 6] local 10.0.0.6 port 5001 connected with 10.0.0.3 port 55273
[ ID] Interval      Transfer    Bandwidth
[ 6] 0.0-10.0 sec  10.8 GBytes  9.24 Gbits/sec
[ 84] local 10.0.0.6 port 5001 connected with 10.0.0.1 port 32772
[ 84] 0.0-10.0 sec  23.7 GBytes  20.4 Gbits/sec
[  ]
```

```
root@sdnhubvm:~/mininet/examples[08:13] (master)$ iperf -c 10.0.0.6 -t 10
Client connecting to 10.0.0.6, TCP port 5001
TCP window size: 85.3 KByte (default)

[ 5] local 10.0.0.1 port 32772 connected with 10.0.0.6 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 5] 0.0-10.0 sec  23.7 GBytes  20.4 Gbits/sec
root@sdnhubvm:~/mininet/examples[08:14] (master)$
```

Capturing from s6-eth2

Filter: Expression

No.	Time	Source	Destination
-----	------	--------	-------------

而 s6-eth3 成功捕获 TCP 包:

```
root@sdnhubvm:~/mininet/examples[08:03] (master)$ iperf -s
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)

[ 6] local 10.0.0.6 port 5001 connected with 10.0.0.3 port 55273
[ ID] Interval      Transfer    Bandwidth
[ 6] 0.0-10.0 sec  10.8 GBytes  9.24 Gbits/sec
[ 84] local 10.0.0.6 port 5001 connected with 10.0.0.1 port 32772
[ 84] 0.0-10.0 sec  23.7 GBytes  20.4 Gbits/sec
[ 61] local 10.0.0.6 port 5001 connected with 10.0.0.1 port 33061
[  ]
```

```
root@sdnhubvm:~/mininet/examples[08:13] (master)$ iperf -c 10.0.0.6 -t 10
Client connecting to 10.0.0.6, TCP port 5001
TCP window size: 85.3 KByte (default)

[ 5] local 10.0.0.1 port 32772 connected with 10.0.0.6 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 5] 0.0-10.0 sec  23.7 GBytes  20.4 Gbits/sec
root@sdnhubvm:~/mininet/examples[08:14] (master)$ iperf -c 10.0.0.6 -t 10
Client connecting to 10.0.0.6, TCP port 5001
TCP window size: 85.3 KByte (default)

[ 5] local 10.0.0.1 port 33061 connected with 10.0.0.6 port 5001
[  ]
```

Capturing from s6-eth3

Filter: Expression

No.	Time	Source	Destination
5713	2.956088000	10.0.0.6	10.0.0.1
5714	3.076269000	10.0.0.1	10.0.0.6
5715	3.076316000	10.0.0.1	10.0.0.6
5716	3.076323000	10.0.0.1	10.0.0.6
5717	3.076912000	10.0.0.6	10.0.0.1
5718	3.139759000	10.0.0.6	10.0.0.1
5719	3.139806000	10.0.0.1	10.0.0.6
5720	3.139832000	10.0.0.1	10.0.0.6
5721	3.139926000	10.0.0.6	10.0.0.1
5722	3.238871000	10.0.0.6	10.0.0.1
5723	3.239018000	10.0.0.1	10.0.0.6
5724	3.239091000	10.0.0.1	10.0.0.6

由此可以验证 TCP 通路设置的正确性。

h1-UDP

回顾 h1 传输 UDP 包到服务器端 h6 的路径为 “h1->s1->s3->s4->s6->h6”。

【实现思路】

同样在线路上的交换机设置匹配项流表，匹配项要求为 UDP 协议，查表-1 可知为 “dl_type=0x0800,nw_proto=17”，注意双向设置流表。

【注意事项】

由于本次实验 topo 图的特殊性，在交换机 s3 以及连接服务器端 h6 的交换机 s6 是两条 UDP 测试通路的交汇处，我们需要添加 nw_src 和 nw_dst 约束区分开这两条 UDP 流表项。

【结果】

用 “xterm h1 h6” 指令打开 h1、h6 终端，利用 4.1.3 中描述过的 iperf 命令进行 UDP 连接传输，如下图所示：

```
"Node: h6"
root@sdnhubvm:~/mininet/examples[08:26] (master)$ iperf -u -s
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[  5] local 10.0.0.6 port 5001 connected with 10.0.0.1 port 51149
[ ID] Interval      Transfer    Bandwidth    Jitter    Lost/TOTAL Datagrams
[  5] 0.0-10.0 sec   105 MBytes  88.0 Mbits/sec  0.023 ms 6996/81865 (8.5%)
[  5] 0.0-10.0 sec   1 datagrams received out-of-order
[]

"Node: h1"
root@sdnhubvm:~/mininet/examples[08:26] (master)$ iperf -u -c 10.0.0.6 -b 100M -t 10
-----
Client connecting to 10.0.0.6, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[  5] local 10.0.0.1 port 51149 connected with 10.0.0.6 port 5001
[ ID] Interval      Transfer    Bandwidth
[  5] 0.0-10.0 sec   115 MBytes  96.3 Mbits/sec
[  5] Sent 81866 datagrams
[  5] Server Report:
[  5] 0.0-10.0 sec   105 MBytes  88.0 Mbits/sec  0.022 ms 6996/81865 (8.5%)
[  5] 0.0-10.0 sec   1 datagrams received out-of-order
root@sdnhubvm:~/mininet/examples[08:27] (master)$
```

在 h1->h6 的 UDP 测试传输 10s 时间中，传输 105MB 数据，带宽为 88Mbps，时延抖动为 0.022ms，丢包率为 8.5%。

h3-UDP

回顾 h3 传输 UDP 包到服务器端 h6 的路径为 “h3->s2->s3->s5->s6->h6”。

【实现思路】

【注意事项】

与 h1-UDP 相同，不赘述。

【结果】

用 “xterm h3 h6” 指令打开 h3、h6 终端，利用 4.1.3 中描述过的 iperf 命令进行 UDP 连接传输，如下图所示：

```
"Node: h6"
root@sdnhubvm:~/mininet/examples[08:34] (master)$ iperf -u -s
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[ 5] local 10.0.0.6 port 5001 connected with 10.0.0.3 port 42027
[ ID] Interval      Transfer    Bandwidth      Jitter    Lost/Total Datagrams
[ 5] 0.0-10.0 sec   108 MBytes  90.5 Mbits/sec  0.020 ms  7365/84324 (8.7%)
[ 5] 0.0-10.0 sec   1 datagrams received out-of-order
[

"Node: h3"
root@sdnhubvm:~/mininet/examples[08:34] (master)$ iperf -u -c 10.0.0.6 -b 100M -t 10
-----
Client connecting to 10.0.0.6, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[ 5] local 10.0.0.3 port 42027 connected with 10.0.0.6 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 5] 0.0-10.0 sec   118 MBytes  99.2 Mbits/sec
[ 5] Sent 84325 datagrams
[ 5] Server Report:
[ 5] 0.0-10.0 sec   108 MBytes  90.5 Mbits/sec  0.019 ms  7365/84324 (8.7%)
[ 5] 0.0-10.0 sec   1 datagrams received out-of-order
root@sdnhubvm:~/mininet/examples[08:34] (master)$
```

在 h3->h6 的 UDP 测试传输 10s 时间中，传输 108MB 数据，带宽为 90.5Mbps，时延抖动为 0.019ms，丢包率为 8.7%。

同样为了验证是否如实按照 topo 图中的 UDP 路径进行传输,我们使用 wireshark 在 h3->h6 时对 s6-eth1、s6-eth3 进行抓包分析，如下图所示：



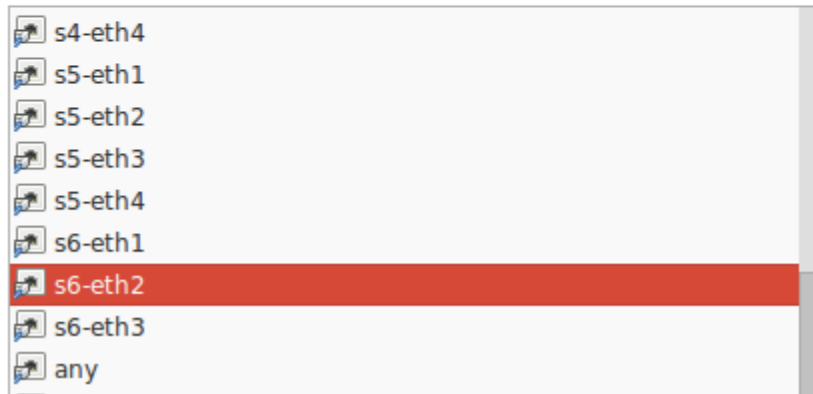
Interface List

Live list of the capture interfaces
(counts incoming packets)



Start

Choose one or more interfaces to capture from, then **Start**



s6-eth1 没有任何 TCP 包:

```
root@sdnhubvm:~/mininet/examples[08:34] (master)$ iperf -u -s
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)

[ 5] local 10.0.0.6 port 5001 connected with 10.0.0.3 port 42027
[ ID] Interval      Transfer    Bandwidth    Jitter    Lost/Totl  Datagrams
[ 5] 0.0-10.0 sec  108 MBytes  90.5 Mbits/sec  0.020 ms  7365/84324 (8.7%)
[ 5] 0.0-10.0 sec  1 datagrams received out-of-order
[ 6] local 10.0.0.6 port 5001 connected with 10.0.0.3 port 54610
[ 6] 0.0- 9.7 sec  117 MBytes  101 Mbits/sec  0.014 ms  1464/85027 (1.7%)
[ 6] 0.0- 9.7 sec  611 datagrams received out-of-order

root@sdnhubvm:~/mininet/examples[08:34] (master)$ iperf -u -c 10.0.0.6 -b 100M -t 10
Client connecting to 10.0.0.6, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 208 KByte (default)

[ 5] local 10.0.0.3 port 54610 connected with 10.0.0.6 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 5] 0.0-10.0 sec  119 MBytes  100 Mbits/sec
[ 5] Sent 85028 datagrams
[ 5] Server Report:
[ 5] 0.0-10.0 sec  108 MBytes  90.5 Mbits/sec  0.019 ms  7365/84324 (8.7%)
[ 5] 0.0-10.0 sec  1 datagrams received out-of-order
root@sdnhubvm:~/mininet/examples[08:36] (master)$
```

而 s6-eth3 成功捕获 TCP 包:

Node: h6

root@sdnhubvm:~/mininet/examples[08:34] (master)\$ iperf -u -s
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)

[5] local 10.0.0.6 port 5001 connected with 10.0.0.3 port 42027
[ID] Interval Transfer Bandwidth Jitter Lost/Total Datagrams
[5] 0.0-10.0 sec 108 MBytes 90.5 Mbits/sec 0.020 ms 7365/84324 (8.7%)
[5] 0.0-10.0 sec 1 datagrams received out-of-order
[6] local 10.0.0.6 port 5001 connected with 10.0.0.3 port 54610
[6] 0.0- 9.7 sec 117 MBytes 101 Mbits/sec 0.014 ms 1464/85027 (1.7%)
[6] 0.0- 9.7 sec 611 datagrams received out-of-order
[5] local 10.0.0.6 port 5001 connected with 10.0.0.3 port 38931

Node: h3

UDP buffer size: 208 KByte (default)

[5] local 10.0.0.3 port 54610 connected with 10.0.0.6 port 5001
[ID] Interval Transfer Bandwidth
[5] 0.0-10.0 sec 119 MBytes 100 Mbits/sec
[5] Sent 85028 datagrams
[5] Server Report:
[5] 0.0- 9.7 sec 117 MBytes 101 Mbits/sec 0.013 ms 1464/85027 (1.7%)
[5] 0.0- 9.7 sec 611 datagrams received out-of-order
root@sdnhubvm:~/mininet/examples[08:36] (master)\$ iperf -u -c 10.0.0.6 -b 100M -t 10

Client connecting to 10.0.0.6, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 208 KByte (default)

[5] local 10.0.0.3 port 38931 connected with 10.0.0.6 port 5001
[ID] Interval Transfer Bandwidth
[5] 0.0-10.0 sec 113 MBytes 94.4 Mbits/sec
[5] Sent 80263 datagrams
[5] Server Report:
[5] 0.0-10.0 sec 111 MBytes 93.0 Mbits/sec 0.573 ms 1158/80262 (1.4%)
[5] 0.0-10.0 sec 1 datagrams received out-of-order
root@sdnhubvm:~/mininet/examples[08:37] (master)\$

Capturing from s6-eth3

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: Expression

No.	Time	Source	Destination
73518	9.987617000	10.0.0.3	10.0.0.6
73519	9.988033000	10.0.0.3	10.0.0.6
73520	9.988130000	10.0.0.3	10.0.0.6
73521	9.988310000	10.0.0.3	10.0.0.6
73522	9.988471000	10.0.0.3	10.0.0.6
73523	9.988512000	10.0.0.3	10.0.0.6
73524	9.991071000	10.0.0.3	10.0.0.6
73525	9.991165000	10.0.0.3	10.0.0.6
73526	9.991186000	10.0.0.3	10.0.0.6
73527	9.991200000	10.0.0.3	10.0.0.6
73528	9.991347000	10.0.0.3	10.0.0.6
73529	9.991412000	10.0.0.3	10.0.0.6
73530	9.991427000	10.0.0.3	10.0.0.6
73531	9.996475000	10.0.0.3	10.0.0.6
73532	10.001219000	10.0.0.3	10.0.0.6
73533	10.001296000	10.0.0.3	10.0.0.6
73534	10.015037000	10.0.0.6	10.0.0.3
73535	15.023757000	22:62:39:b4:89:18	62:48:f2:75:7c:e7
73536	15.023806000	62:48:f2:75:7c:e7	22:62:39:b4:89:18

Frame 1: 1512 bytes on wire (12096 bits). 1512 bytes captured

由此可以验证 UDP 通路设置的正确性，同时到此为止，可以验证该 SDN 路由中的约束策略包括了“目的地址+传输协议类型”以及“目的地址+源地址”。