

实验四——SDN 环境搭建与协议仿真

一、实验目的

- 学习了解软定义网络 SDN
- 了解 SDN 封包传输的流程
- 通过观察 OpenFlow 跟传统网络协议的沟通过程
- 希望能在实体及其的环境下对 SDN 跟传统网络有更好的理解

二、实验前的准备

- 熟悉了解软定义网络 SDN
- 了解 Mininet 的使用
- 环境准备：linux 系统【可以安装虚拟机在虚拟机上完成实验，最好安装 Ubuntu16.x 及以上版本进行实验】

三、实验内容

- 安装 Mininet，熟悉其命令
- 利用 Mininet 创建网络拓扑
- 简易环境测试
- 利用 Wireshark 在 Controller 端所截取的包进行分析

四、实验指导

A. Mininet 安装与使用

1) Mininet 安装

- 首先打开 terminal 界面，输入指令：`sudo -i`，输入自己的密码进入 root 权限
- 输入指令：`git clone https://github.com/mininet/mininet.git`
- 安装 Mininet，需要安装涉及安装 Mininet、user 交换机及 OVS 软件，可根据 `mininet/util/install.sh -h` 命令选择参数进行安装，如：
`mininet/util/install.sh -n3V 2.5.0`
- **注：**`-n` 指 mininet 的核心文件和依赖；`3` 指的是安装 openflow 的 1.3 版本协议，同时也支持 1.0 版本的协议；`V` 指的是 open vSwitch 的版本，后接 open vSwitch 的版本

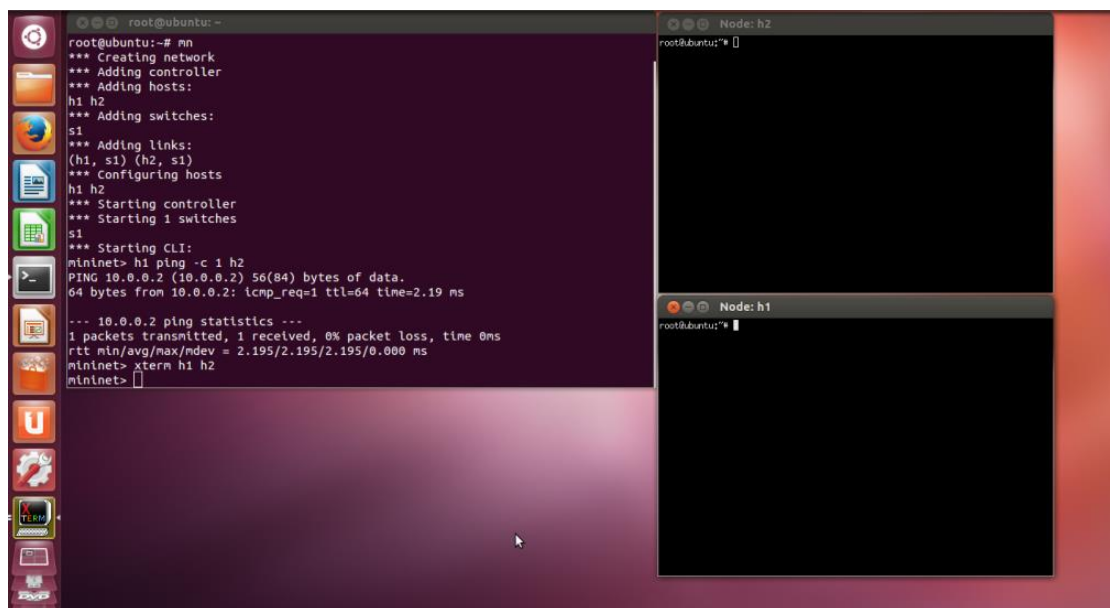
2) Mininet 命令

- 输入指令：`mn` 可以创建出 Mininet 最基本的虚拟拓扑

```
root@ubuntu: ~
peter@ubuntu:~$ sudo -i
[sudo] password for peter:
root@ubuntu:~# mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

从上图可以看出此基本网络拓扑中包含虚拟 host(h1,h2)、一个 switch(s1)和一个 controller(c0)

- 输入指令: `nodes` 查看各个节点
- 输入命令: `net` 可以看到各个连接的讯息
- 输入指令: `dump` 可以看到各个节点的讯息
- 输入指令: `h1 ping -c 1 h2`, 可以用 `h1 ping h2`, 反之也可以从 h2ping 过去
- 输入指令: `ping all`, 所有主机与其他主机进行 ping 测试
- 除了直接使用 Mininet 也可以只用命令: `xterm h1 h2` 来叫出两个 host 的命令视窗



Mininet 基础指令主要分为三大类: 网络构建启动参数, 进入网络内部后常用的交互命令以及外部运行参数

- 网络构建启动参数:

✧ `-topo`:

- ✓ 单一 (single) 拓扑: 整个网络拓扑中交换机有且只有一个, 其可以下挂一个或多个主机: `sudo mn -topo=single,3`
- ✓ 线形 (linear) 拓扑: 交换机连接呈线性排列, 且每个交换机所连

接主机数目只有一个: `sudo mn -topo=linear,4`

- ✓ 树形 (tree) 拓扑: 交换机连接成树形排列, 且每个交换机所连接主机一般有多个: `sudo mn -topo=tree, depth=2, fanout=2`
- ✓ 自定义 (custom) 拓扑: Python 编写文件 `file.py`, 执行此脚本即可创建定义的拓扑, `-custom` 与 `-topo` 联用: `sudo mn -custom file.py -topo mytopo`。注: `file.py` 最好使用绝对路径

✧ `-switch:`

- ✓ 定义 Mininet 要使用的交换机 (默认使用 OVS, 即 OpenVSwitch 交换机)
- ✓ 注: 弄明白内核态和用户态交换机

✧ `-controller:`

- ✓ 定义要使用的控制器, 如果没有指定则使用 Mininet 中默认的控制器。连接远程控制器, 可以指定存在于本机或者与之相连通设备上的控制器, 指定远程控制器的方法: `sudo mn -controller=remote,--ip=[controller IP],--port=[port]`
- ✓ 注: `port` 指的是控制器的监听端口, IP 和 `port` 可以省略, 如果省略的话, 默认使用的是本地的 IP 地址和 6653 端口或者 6633 端口, 视情况而定。

✧ `-mac:`

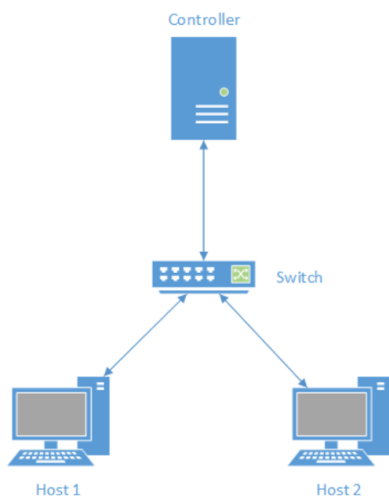
- ✓ 自动设置设备的 MAC 地址让 MAC 地址易读, 即设置交换机的 MAC、主机 MAC 及 IP 地址从小到大排序, 且设置简单唯一, 不仅让机器容易读取, 也容易让肉眼很容易识别其 ID。使用方法: `sudo mn -topo=tree, depth=2, fanout=2, --mac`

➤ `help` 可以显示一系列命令帮助信息

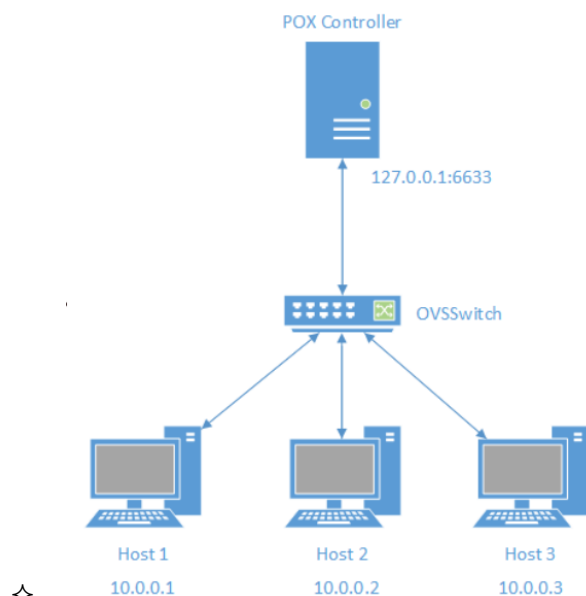
B. 利用 mininet 创建网络拓扑

➤ 输入指令, 建立预设 mininet 网络拓扑:

- ✧ 2 台 Host
- ✧ 1 台 Switch
- ✧ 1 台 Controller



✧ 但是, 此次实验需要建立的网络拓扑图为:



- ✧
- ✧ 可以利用四. A. 2)中提到的命令进行创建

C. 简易环境测试

- 这部分实验主要是希望利用 h1 ping h2 这一过程了解 SDN 封包传输的流程
- 首先先将 h1、h2 的 xterm 启动
- 接着观察 h1 和 h2 的网络属性，分别在 h1 和 h2 的视窗中输入 `ipconfig`

```
"Node: h1"
root@ubuntu:~# ifconfig
h1-eth0  Link encap:Ethernet  HWaddr 00:00:00:00:00:01
         inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
         inet6 addr: fe80::200:ff:fe00:1/64 Scope:Link
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:39 errors:0 dropped:0 overruns:0 frame:0
         TX packets:9 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:1000
         RX bytes:5027 (5.0 KB)  TX bytes:738 (738.0 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)

root@ubuntu:~#

"Node: h2"
root@ubuntu:~# ifconfig
h2-eth0  Link encap:Ethernet  HWaddr 00:00:00:00:00:02
         inet addr:10.0.0.2  Bcast:10.255.255.255  Mask:255.0.0.0
         inet6 addr: fe80::200:ff:fe00:2/64 Scope:Link
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:39 errors:0 dropped:0 overruns:0 frame:0
         TX packets:9 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:1000
         RX bytes:5027 (5.0 KB)  TX bytes:738 (738.0 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)

root@ubuntu:~#
```

- ✧
- ✧ 由上面的两张图可以得知，h1 的 ip 为 10.0.0.1，h2 的 ip 为 10.0.0.2

- 查看流表: `ovs-ofctl dump-flows s1`

```

root@ubuntu:~# ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=5.071s, table=0, n_packets=6, n_bytes=588, idle_timeout=10,
 , hard_timeout=30, idle_age=0, priority=65535,icmp,in_port=1,vlan_tci=0x0000,dl_
 src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,nw_src=10.0.0.1,nw_dst=10.0.0.2,n
 w_tos=0,icmp_type=8,icmp_code=0 actions=output:2
 cookie=0x0, duration=5.068s, table=0, n_packets=6, n_bytes=588, idle_timeout=10,
 , hard_timeout=30, idle_age=0, priority=65535,icmp,in_port=2,vlan_tci=0x0000,dl_
 src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,nw_src=10.0.0.2,nw_dst=10.0.0.1,n
 w_tos=0,icmp_type=0,icmp_code=0 actions=output:1
 cookie=0x0, duration=0.011s, table=0, n_packets=1, n_bytes=42, idle_timeout=10,
 , hard_timeout=30, idle_age=0, priority=65535,arp,in_port=2,vlan_tci=0x0000,dl_sr
 c=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,arp_spa=10.0.0.2,arp_tpa=10.0.0.1,a
 rp_op=1 actions=output:1
 cookie=0x0, duration=0.009s, table=0, n_packets=1, n_bytes=42, idle_timeout=10,
 , hard_timeout=30, idle_age=0, priority=65535,arp,in_port=1,vlan_tci=0x0000,dl_sr
 c=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,arp_spa=10.0.0.1,arp_tpa=10.0.0.2,a
 rp_op=2 actions=output:2
 root@ubuntu:~#
  
```

- ✧
- ✧ 由上图可知，一个 ping 包会有 4 条 flow 到流表中
- ✧ 第一个与第二个 flow 主要是 h1 到 h2 的 icmp 包与 h2 到 h1 的 icmp 包匹配

D. Wireshark 抓包

- 当安装完 Mininet 之后，Wireshark 也会一并安装进去，于是可以打开 Wireshark 来监看包的状态
- Switch connect Controller (在 controller 上观察)

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	SuperMic_cb:77:28	Broadcast	ARP	60	Who has 10.0.0.254? Tell 10.0.0.2
2	0.999307187	SuperMic_cb:77:28	Broadcast	ARP	60	Who has 10.0.0.254? Tell 10.0.0.2
3	1.355813642	10.0.0.1	10.0.0.3	TCP	74	36005→6653 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=61301 TSecr=0
4	1.355851142	10.0.0.3	10.0.0.1	TCP	54	6653→36005 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5	1.999011146	SuperMic_cb:77:28	Broadcast	ARP	60	Who has 10.0.0.254? Tell 10.0.0.2
6	2.045591608	0.0.0.0	255.255.255.255	DHCP	342	DHCP Discover - Transaction ID 0xc636415f
7	5.135880403	0.0.0.0	255.255.255.255	DHCP	342	DHCP Discover - Transaction ID 0xc636415f
8	9.365834868	10.0.0.1	10.0.0.3	TCP	74	36006→6653 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=62102 TSecr=0
9	9.365853395	10.0.0.3	10.0.0.1	TCP	74	6653→36006 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=62102 TSecr=0
10	9.366123301	10.0.0.1	10.0.0.3	TCP	66	36006→6653 [ACK] Seq=1 Ack=1 Win=14720 Len=0 TSval=62102 TSecr=91628
11	9.366341941	10.0.0.1	10.0.0.3	OpenFlow	74	Type: OFPT_HELLO 1
12	9.366348455	10.0.0.3	10.0.0.1	TCP	66	6653→36006 [ACK] Seq=1 Ack=9 Win=29056 Len=0 TSval=91628 TSecr=62102
13	9.400389792	10.0.0.3	10.0.0.1	OpenFlow	82	Type: OFPT_HELLO 2
14	9.401020862	10.0.0.1	10.0.0.3	TCP	66	36006→6653 [ACK] Seq=9 Ack=17 Win=14720 Len=0 TSval=62105 TSecr=91637
15	9.406167956	10.0.0.3	10.0.0.1	OpenFlow	74	Type: OFPT_FEATURES_REQUEST
16	9.406509538	10.0.0.1	10.0.0.3	TCP	66	36006→6653 [ACK] Seq=9 Ack=25 Win=14720 Len=0 TSval=62106 TSecr=91638
17	9.407310698	10.0.0.1	10.0.0.3	OpenFlow	98	Type: OFPT_FEATURES_REPLY
18	9.428373806	10.0.0.3	10.0.0.1	OpenFlow	82	Type: OFPT_MULTIPART_REQUEST
19	9.428777570	10.0.0.1	10.0.0.3	OpenFlow	514	Type: OFPT_MULTIPART_REPLY
20	9.43369205	10.0.0.3	10.0.0.1	OpenFlow	94	Type: OFPT_GET_CONFIG_REQUEST
21	9.434385302	10.0.0.1	10.0.0.3	OpenFlow	74	Type: OFPT_BARRIER_REPLY
22	9.434407631	10.0.0.1	10.0.0.3	OpenFlow	78	Type: OFPT_GET_CONFIG_REPLY
23	9.434547388	10.0.0.3	10.0.0.1	TCP	66	6653→36006 [ACK] Seq=69 Ack=509 Win=30880 Len=0 TSval=91645 TSecr=62108
24	9.438858117	10.0.0.3	10.0.0.1	OpenFlow	82	Type: OFPT_MULTIPART_REQUEST
25	9.439156398	10.0.0.1	10.0.0.3	OpenFlow	1138	Type: OFPT_MULTIPART_REPLY
26	9.447543796	10.0.0.3	10.0.0.1	OpenFlow	82	Type: OFPT_MULTIPART_REQUEST
27	9.463567373	10.0.0.1	10.0.0.3	TCP	4410	[TCP segment of a reassembled PDU]
28	9.463599795	10.0.0.1	10.0.0.3	TCP	66	6653→36006 [ACK] Seq=101 Ack=5925 Win=40960 Len=0 TSval=91652 TSecr=62111

- ✧
- ✧ 以上是包的截图
- ✧ 首先看第一个 OpenFlow 包，可以看到这个封包的讯息是 HELLO，称为 Hello message，是交换机和控制器用来建立连接的封包。里面包含目前可支持的最新 OF 版本协议。从这张图可以看到首个 Hello message 是从 10.0.0.1 发出的，也就是从交换机发出的。我们可以点开来看更详细的信息：

✓

```
Frame 13: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface 0
Ethernet II, Src: Vmware_40:6d:f7 (00:0c:29:40:6d:f7), Dst: Edgecore_6b:15:20 (cc:37:ab:6b:15:20)
Internet Protocol Version 4, Src: 10.0.0.3, Dst: 10.0.0.1
Transmission Control Protocol, Src Port: 6653, Dst Port: 36006, Seq: 1, Ack: 9, Len: 16
OpenFlow 1.4
  Version: 1.4 (0x05)
  Type: OFPT_HELLO (0)
  Length: 16
  Transaction ID: 4294967295
  Element
```

五、实验报告和结果演示

- 安装 Mininet，熟悉了解 Mininet 的使用
- 按照实验指导的内容建立指定网络拓扑，并进行建议环境测试
- 根据实验内容，说明 SDN 封包传输的流程
- 利用 Wireshark 查看包的状态。为了节约时间，抓包分析仅在实验查验时检查，不需要再实验报告中进行分析
- 完成电子版实验报告【命名格式：学号-姓名-AdvancedNetLab4】，并上传提交
- 报告提交截止时间为：11 月 23 日 0 点
- **严禁相互抄袭，如发现雷同，实验成绩计 0 分**