南京大学本科生实验报告

课程名称：**计算机网络**　　任课教师：田臣/李文中　　助教：

学院	**计算机科学与技术**	专业（方向）	**计算机科学与技术**
学号	**201220062**	姓名	**黄子睿**
Email	**201220062@smail.nju.edu.cn**	开始/完成日期	**3/6-3/7**

1. 实验名称：

Lab1 Switchyard & Mininet

2. 实验目的：

安装完善并熟悉实验环境与相关软件，包括实验平台 linux，git & github、主要实验语言 python 以及相关专业软件 Mininet, Wireshark, Switchyard, etc。并在理解的基础上修改实验要求的用例以完成要求的功能，并能对结果做出对应的解释。

3. 实验内容：

Step 1: Modify the Mininet topology

在 start_mininet.py 中，nodes 定义了网络™拓扑结构中存在的节点。

```
nodes = {
    "server1": {
        "mac": "10:00:00:00:00:{:02x}",
        "ip": "192.168.100.1/24"
    },
    #"server2": {
    #    "mac": "20:00:00:00:00:{:02x}",
    #    "ip": "192.168.100.2/24"
    #},
      "client": {
          "mac": "30:00:00:00:00:{:02x}",
          "ip": "192.168.100.3/24"
      },
```

```
        "hub": {
            "mac": "40:00:00:00:00:{:02x}",
        }
    }
```

在修改代码（注释“server2”部分）前，拓扑结构如下图所示，呈现星形。

这里我选择了实验手册中给出的第一种方案，删去 server2 节点，构造了一个纺锤形的拓扑结构，具体如下图所示。

```
# Host and link configuration
#
#
#   server1
#          \
#           hub----client
#          /
#   server2
#

#After the server2 1s deleted, the topology is like:
#
#   server1 ---- hub ---- client
#
```

删去 server2 后，server1 与 client 作为网络中的 2 个节点，通过 hub 连接，构成了一个三节点的纺锤形结构。代码中连接节点的部分如下所示，注释给出了我的理解。

```
for node in nodes.keys():   #add nodes in the topology
    self.addHost(node, **nodeconfig)
for node in nodes.keys():   #connect non-hub hosts with hub so the topology is connected.
    # all links are 309Mb/s, 100 millisecond prop delay
    if node != "hub":
        self.addLink(node, "hub", bw=10, delay="100ms")
```

最后讲一下我对代码结构的理解：

首先我们定义了一个 PySwitchTopo 类，并定义了拓扑结构中的 nodes。

在 main 函数中，实例化 PySwitchTopo 这个类来生成类定义__init__中规定好的拓扑结构，接着通过实例化一个 Mininet 对象，传入实例化好的 topo 来生成我们的网络 net；完成后通过调用先前定义的 setup_addressing 函数，在其内部通过 reset_macx()与 set_ip()函数来设置网络中的各个节点的 mac 地址与 ip 地址（通过 hub 互相连接），从而完成网络设置。之后 main 调用了 disable_ipv6 来说明禁止了 ipv6 协议生效;最后，执行 net.interact()进入我们的 ping、drop 等测试环节（不过现在还未配置好路由规则因此还 ping 不了）。

Step 2：Modify the logic of a device

为了实现实验手册中要求的功能，我选择在代码中增加两个计数器。

```
#add a packet in/out cnt
in_cnt = 0  #ingress packet count  (这两行是增加的代码)
out_cnt = 0 #egress packet count
```

根据理论模型，in_cnt 应在有信号传入节点时增加 1，out_cnt 则记录该节点发送信号给其余节点的数量。代码如下所示：

```
  log_debug (f"In {net.name} received packet {packet} on {fromIface}")

eth = packet.get_header(Ethernet)
if eth is None:
    log_info("Received a non-Ethernet packet?!")
    return
if eth.dst in mymacs:
    in_cnt += 1
    log_info("Received a packet intended for me")
else:
    in_cnt += 1
    for intf in my_interfaces:
        if fromIface!= intf.name:
            #log_info (f"Flooding packet {packet} to {intf.name}")
```

```
            out_cnt += 1
            net.send_packet(intf, packet)
    log_info(f"in:{in_cnt} out:{out_cnt}")
```

在这段代码中，给出了 in_cnt 与 out_cnt 计数逻辑。对于空包的情况，不做计数。对于目标地址就是 hub 的 packet，只有进入端口的 in_cnt 计数，而不存在向外转发的端口，因此 cnt_out 不做改变。最后对于最一般的情况，hub 从一个端口接收 packet，再检查地址不为自身后，通过所有其余端口向外转发，因此 out_cnt 会在循环中增加。

这里也可以看出 hub 与 layer switch 的不同，前者总是将地址不为自身的包从除接受接口以外的所有接口转发出去，而后者首先会对包的前几节信息进行处理，与内部存储的地址表进行比对，如果发现目标，则将信息直接发往目的地；否则才将信号广播到所有端口。

Step 3: Modify the test scenario of a device

根据我对实验手册与相关代码的理解，我认为 myhub_testscenario.py 中原有的三种情况已经囊括了所有可能的情形，因此我据此设计的 testcase 4 是对上面三种情况中的一种的模仿。

通过阅读材料，我发现 testcase 可能的情况分为以下三种：

a) test case 1: a frame with broadcast destination should get sent out all ports except ingress 即这是一个广播目标的帧，因此从 hub 的其中一个端口传入后，将从所有其他端口传出。

b) test case 2: a frame with any unicast address except one assigned to hub interface should be sent out all ports except ingress，即这是一个定向的

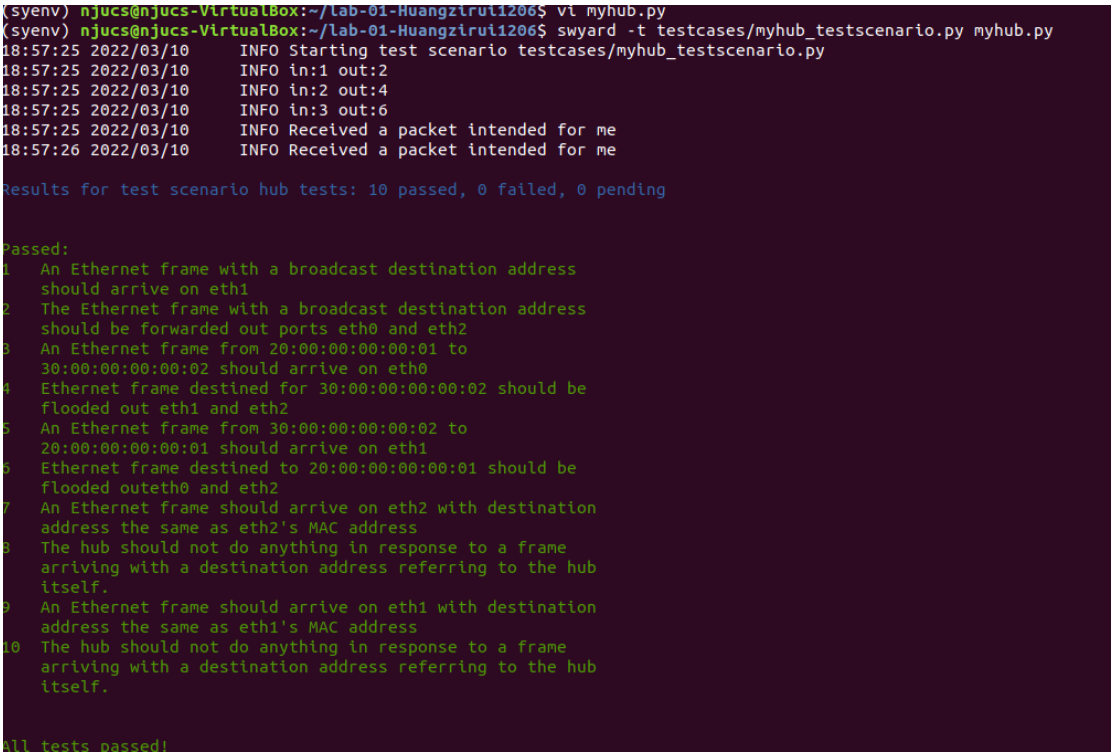
帧，但是目标不是 hub 中。因此该帧从一个端口传入后，仍从其余端口传出，同时将出现应答的情况；

c) test case 3: a frame with dest address of one of the interfaces should result in nothing happening，在这种情况下，这个帧的目标地址就是 hub，因此当从一个端口传入之后，不再有帧从 hub 中传出。

我模仿 test case 3，设计了自己的 test case 4：

```
# test case 4:Similar to test case 3, a frame with dest address of one of the interfaces should
# result in nothing happening
# the packet arrives at eth1 but aims at eth1
reqpkt = new_packet(
    "20:00:00:00:00:01",
    "10:00:00:00:00:02",
    '192.168.1.100',
    '172.16.42.2'
)
 s.expect(
     PacketInputEvent("eth1", reqpkt, display=Ethernet),
     ("An Ethernet frame should arrive on eth1 with destination address "
      "the same as eth1's MAC address")
 )
 s.expect(
     PacketInputTimeoutEvent(1.0),
     ("The hub should not do anything in response to a frame arriving with"
      " a destination address referring to the hub itself.")
 )
```

在命令行中输入 swyard -t testcases/myhub_testscenario.py myhub.py，可以得到修改后 testcase 的测试结果，如下图所示：

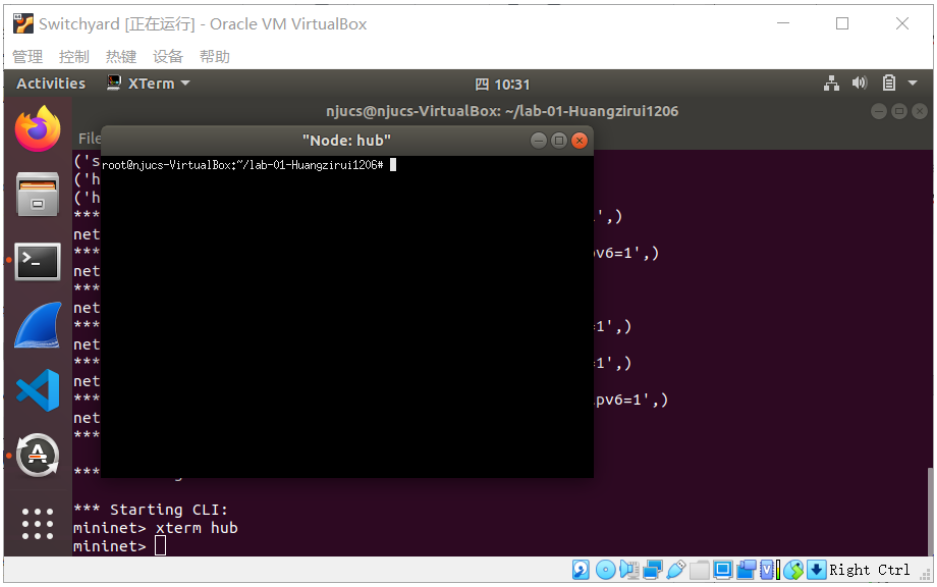


可以看到结果是正确的。

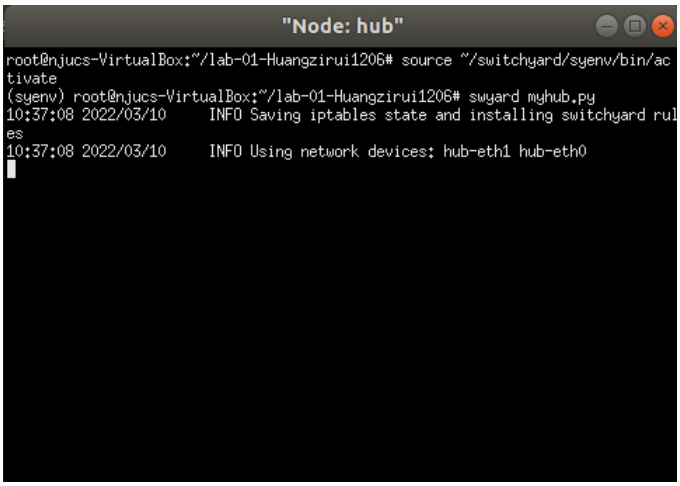
Step 4：Run your device in Mininet

首先在根目录下输入指令：sudo python start_mininet.py，开始之前在 shart_mininet 中实现的拓扑结构。

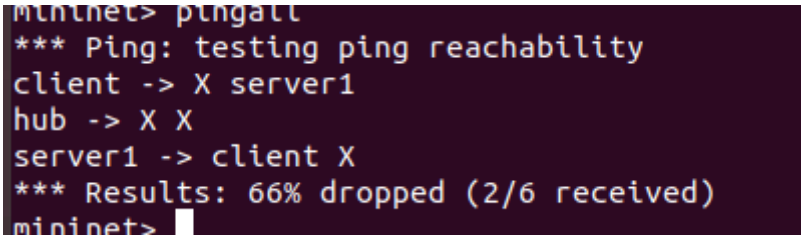
然后在 mininet 下输入 xterm hub 以更好观察输出。得到下图所示窗口：



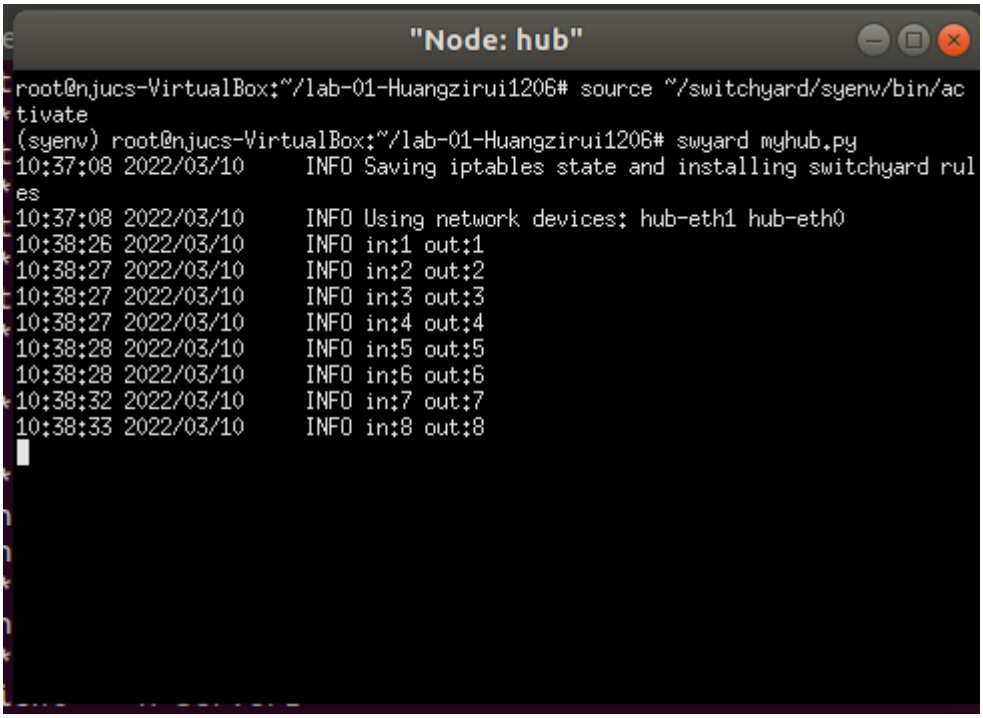
接着在打开的窗口中输入 source ~/switchyard/syenv/bin/active 激活 syenv，再输入 swyard muhub.py 观察 hub 运行结果。得到下图结果：



接下来就可以通过 ping、drop 等命令进行测试了。下面展示再 mininet 命令行下 ping all 的结果：



同时 Step 2 中要求的 in 与 out 的输出在“Node：hub”中显示：



Step 5：Capture using Wireshark

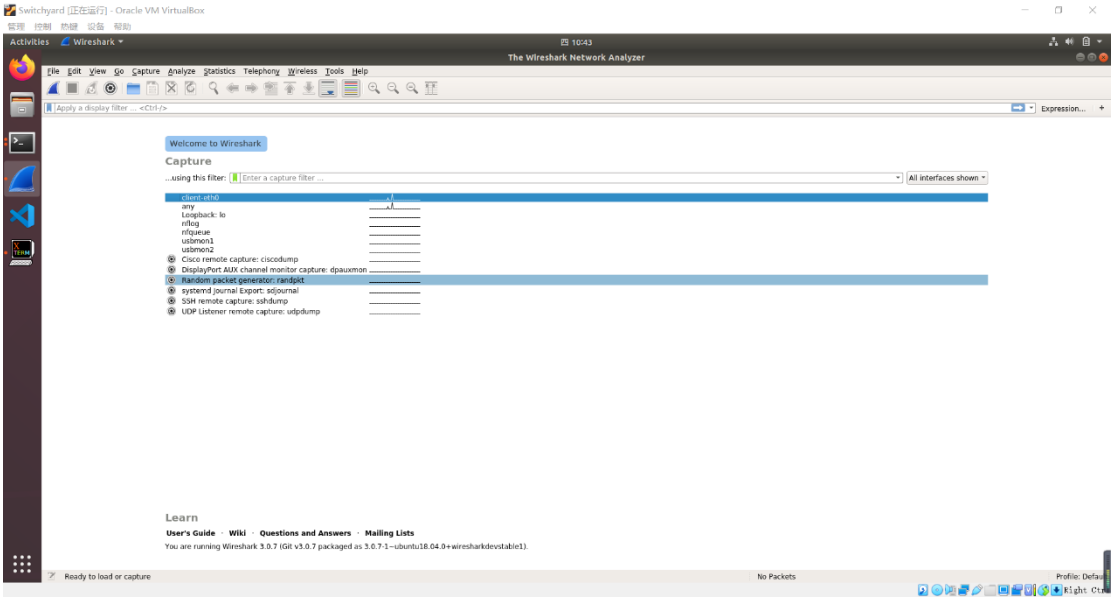
在 Step 4 的基础上，我选择在 mininet 环境下进行捕捉。

在 mininet 命令行下依次输入 client wireshark &，client ping -c1 server1，通过 wireshark 对 hub 与 server1 的连接进行捕捉：

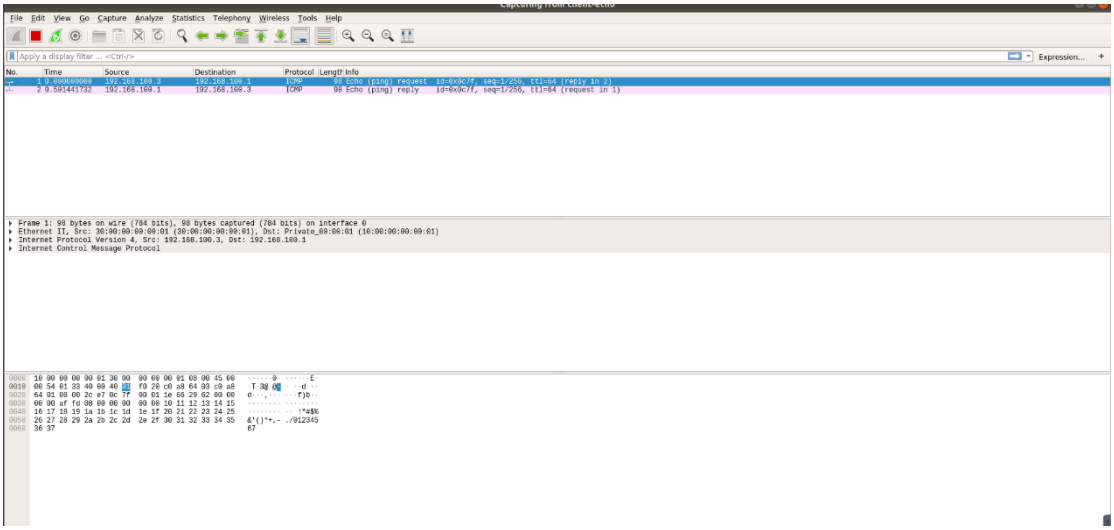
```
mininet> client wireshark &
mininet> client ping -c1 server1
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
PING 192.168.100.1 (192.168.100.1) 56(84) bytes of data.
64 bytes from 192.168.100.1: icmp_seq=1 ttl=64 time=602 ms

--- 192.168.100.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 602.399/602.399/602.399/0.000 ms
mininet>
```

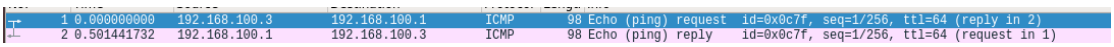
在 wireshark 的界面中选择 client-eth0，捕捉相关信号：



再键入 client ping -c1 server1，得到如下结果：

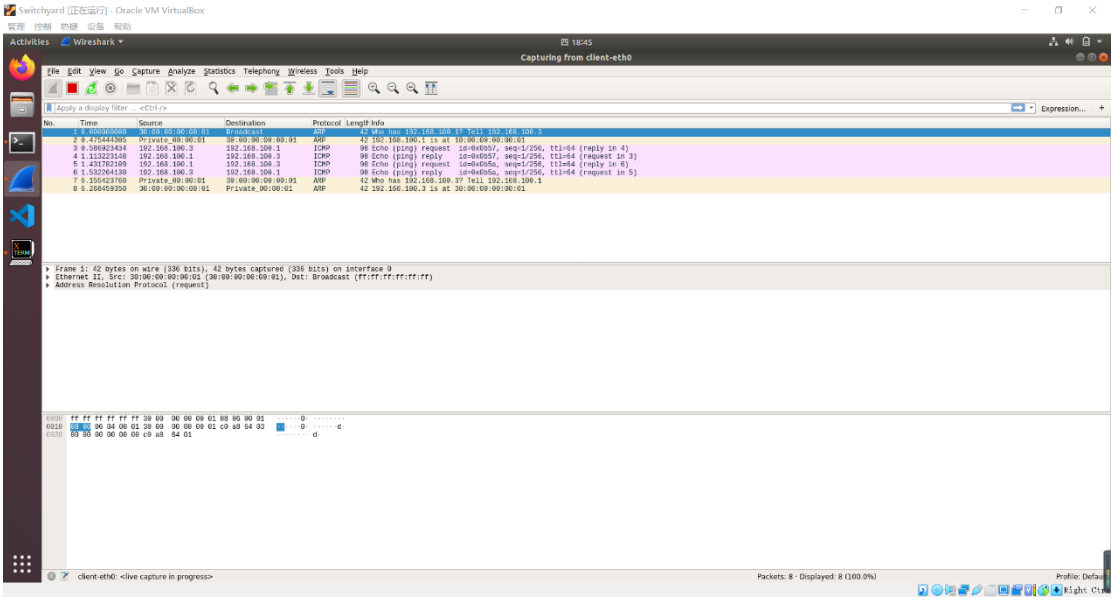


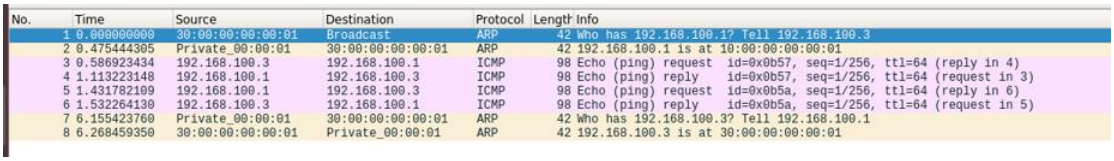
具体这一部分体现了 server1（192.168.100.1）与 client（192.168.100.3）之间的应答过程。下面的 98 bytes on wire, 98 bytes captured 也证明才信号传递过程中没有丢包。



停止捕捉过程后保存图片，即完成 Step 5 的所有要求。

在验收中，助教询问了在 wireshark 中捕捉 pingall 信号的结果含义，先附上对应的结果图：





No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	30:00:00:00:00:01	Broadcast	ARP	42	Who has 192.168.100.1? Tell 192.168.100.3
2	0.475444305	Private_00:00:01	30:00:00:00:00:01	ARP	42	192.168.100.1 is at 10:00:00:00:00:01
3	0.586923434	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) request id=0x0b57, seq=1/256, ttl=64 (reply in 4)
4	1.113223148	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) reply id=0x0b57, seq=1/256, ttl=64 (request in 3)
5	1.431782109	192.168.100.1	192.168.100.3	ICMP	98	Echo (ping) request id=0x0b5a, seq=1/256, ttl=64 (reply in 6)
6	1.532264130	192.168.100.3	192.168.100.1	ICMP	98	Echo (ping) reply id=0x0b5a, seq=1/256, ttl=64 (request in 5)
7	6.155423760	Private_00:00:01	30:00:00:00:00:01	ARP	42	Who has 192.168.100.3? Tell 192.168.100.1
8	6.268459350	30:00:00:00:00:01	Private_00:00:01	ARP	42	192.168.100.3 is at 30:00:00:00:00:01

这里的相关信息中，出现了 ARP 与 ICMP 的协议的内容。通过后面的 info 与课上学习并查阅资料可知， ARP（Address Resolution Protocol）即地址解析协议，，是根据 IP 地址获取物理地址的一个 TCP/IP 协议。在这里用于查询 192.168.100.1 的地址，并将其告知 192.168.100.3，而 ICMP 协议则与 ping 相关，它是 TCP/IP 协议簇的一个子协议，用于在 IP 主机、路由器之间传递控制消息。 控制消息是指网络通不通、主机是否可达、路由是否可用等网络本身的消息。

到这里，Lab 1 的所有实验就完成了。