实验报告

计算机网络-Lab1

专业：计算机科学与技术

任课教师：田臣

周迅 201220037

目录

一、 实验目的

第一次实验为熟悉实验流程，内容有：

- 安装配置好环境
- 获取并修改几处代码
- 运行程序并抓包
- 写实验报告，和代码一并提交

既然助教哥哥讲了，那我就负责搬运一下。

二、 实验内容

学习 Switchyard、Mininet 等软件和模块的使用

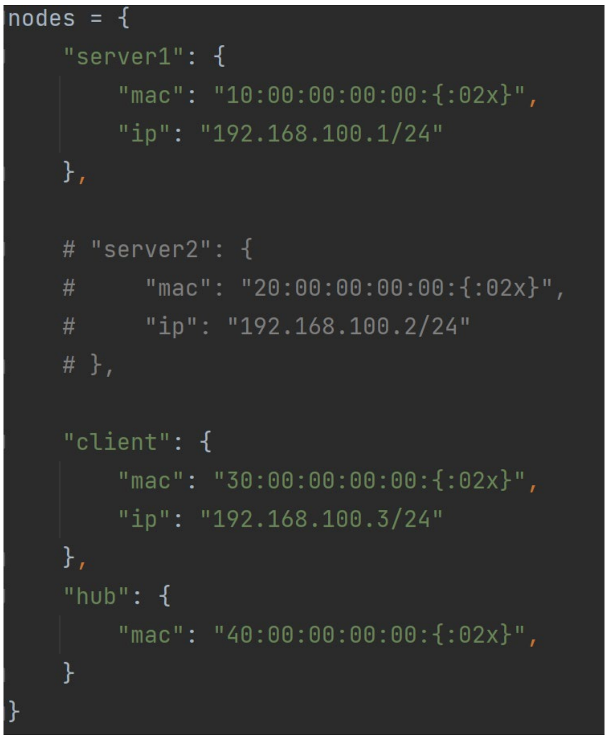
（此处省略一万字；文字已不足以表达这部分的艰难）

1. Modify the Mininet topology

In the section Mininet, we introduced how to construct a topology. So here we have two options for you, choose **one** to implement. ✅ Then show the details of how you build the topology in your report.

- Delete `server2` in the topology,
- Or create a different topology containing 6 nodes using hosts and hubs (don't use other kinds of devices).

二选一是吧？那……还用问选那个嘛？

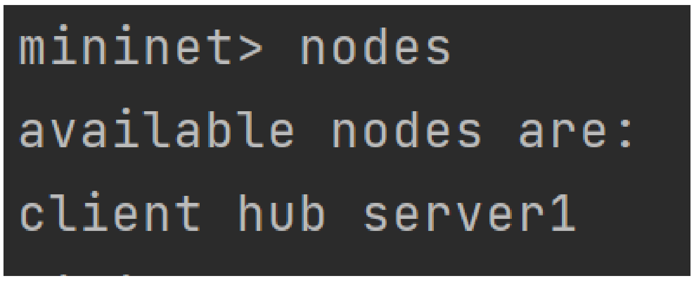


```
nodes = {
    "server1": {
        "mac": "10:00:00:00:00:{:02x}",
        "ip": "192.168.100.1/24"
    },

    # "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}",
    }
}
```

找到 server2，注释。然后，尝试跑一下。



```
mininet> nodes
available nodes are:
client hub server1
```

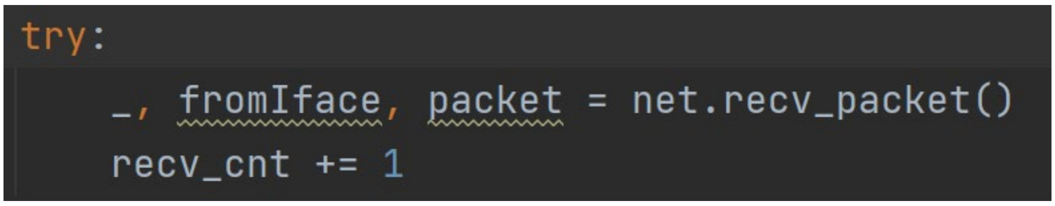
应该，没啥问题吧……

2. Modify the logic of a device

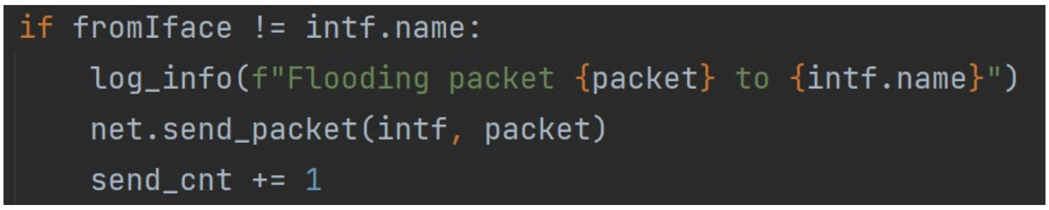
In the section Switchyard, we introduced how to program a device. Your task is to count how many packets pass through a hub in and out. You need to log the statistical result every time you receive one packet with the format of each line `in:<ingress packet count> out:<egress packet count>`. For example, if there is a packet that is not addressed to the hub itself, then the hub may log `in:1 out:2`. ✅ Then show the log of your hub when running it in Mininet and how you implement it in your report.

懂了，数数。

我们每次收到一个包，掰个手指；每次发出一个包，掰个脚趾。

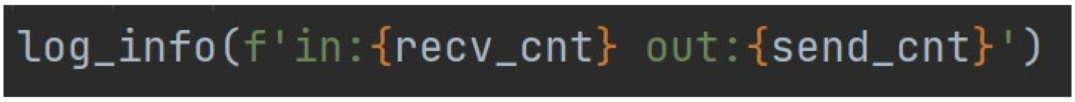


```
try:
    _, fromIface, packet = net.recv_packet()
    recv_cnt += 1
```



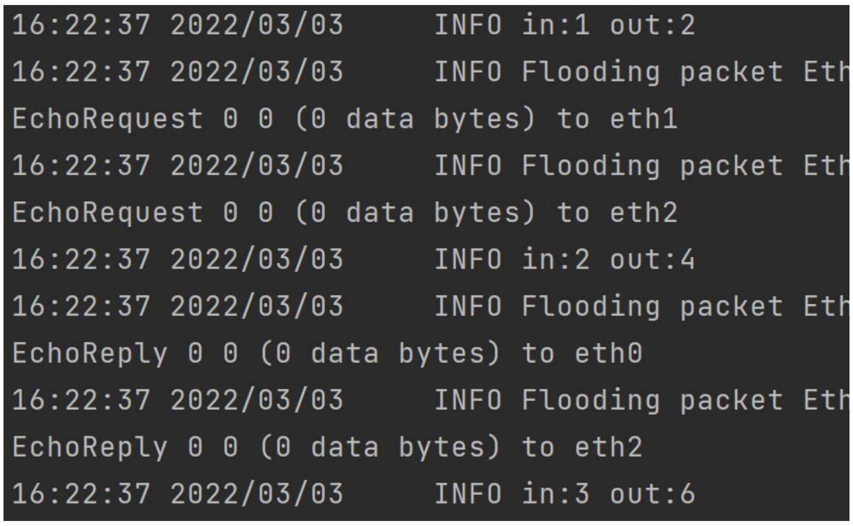
```
if fromIface != intf.name:
    log_info(f"Flooding packet {packet} to {intf.name}")
    net.send_packet(intf, packet)
    send_cnt += 1
```

最后，我们 Log 一下。



```
log_info(f'in:{recv_cnt} out:{send_cnt}')
```

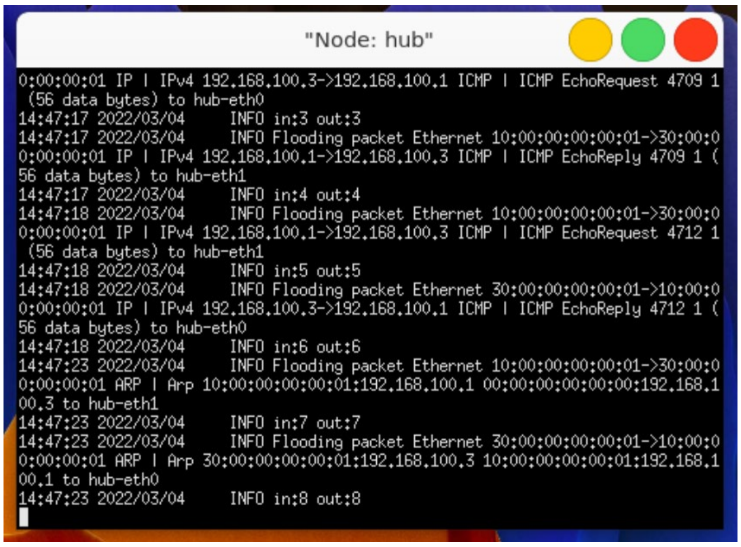
尝试用 myhub_testscenario.py 跑一下。



```
16:22:37 2022/03/03     INFO in:1 out:2
16:22:37 2022/03/03     INFO Flooding packet Eth
EchoRequest 0 0 (0 data bytes) to eth1
16:22:37 2022/03/03     INFO Flooding packet Eth
EchoRequest 0 0 (0 data bytes) to eth2
16:22:37 2022/03/03     INFO in:2 out:4
16:22:37 2022/03/03     INFO Flooding packet Eth
EchoReply 0 0 (0 data bytes) to eth0
16:22:37 2022/03/03     INFO Flooding packet Eth
EchoReply 0 0 (0 data bytes) to eth2
16:22:37 2022/03/03     INFO in:3 out:6
```

可以看到，正常运行。

尝试 `pingall`。由于我们之前删除了 server2，所以 in 和 out 应为相等的值。



```
0:00:00:01 IP | IPv4 192.168.100.3->192.168.100.1 ICMP | ICMP EchoRequest 4709 1
 (56 data bytes) to hub-eth0
14:47:17 2022/03/04     INFO in:3 out:3
14:47:17 2022/03/04     INFO Flooding packet Ethernet 10:00:00:00:00:01->30:00:0
0:00:00:01 IP | IPv4 192.168.100.1->192.168.100.3 ICMP | ICMP EchoReply 4709 1 (
56 data bytes) to hub-eth1
14:47:17 2022/03/04     INFO in:4 out:4
14:47:18 2022/03/04     INFO Flooding packet Ethernet 10:00:00:00:00:01->30:00:0
0:00:00:01 IP | IPv4 192.168.100.1->192.168.100.3 ICMP | ICMP EchoRequest 4712 1
 (56 data bytes) to hub-eth1
14:47:18 2022/03/04     INFO in:5 out:5
14:47:18 2022/03/04     INFO Flooding packet Ethernet 30:00:00:00:00:01->10:00:0
0:00:00:01 IP | IPv4 192.168.100.3->192.168.100.1 ICMP | ICMP EchoReply 4712 1 (
56 data bytes) to hub-eth0
14:47:18 2022/03/04     INFO in:6 out:6
14:47:23 2022/03/04     INFO Flooding packet Ethernet 10:00:00:00:00:01->30:00:0
0:00:00:01 ARP | Arp 10:00:00:00:00:01:192.168.100.1 00:00:00:00:00:00:192.168.1
00.3 to hub-eth1
14:47:23 2022/03/04     INFO in:7 out:7
14:47:23 2022/03/04     INFO Flooding packet Ethernet 30:00:00:00:00:01->10:00:0
0:00:00:01 ARP | Arp 30:00:00:00:00:01:192.168.100.3 10:00:00:00:00:01:192.168.1
00.1 to hub-eth0
14:47:23 2022/03/04     INFO in:8 out:8
```

3. Modify the test scenario of a device

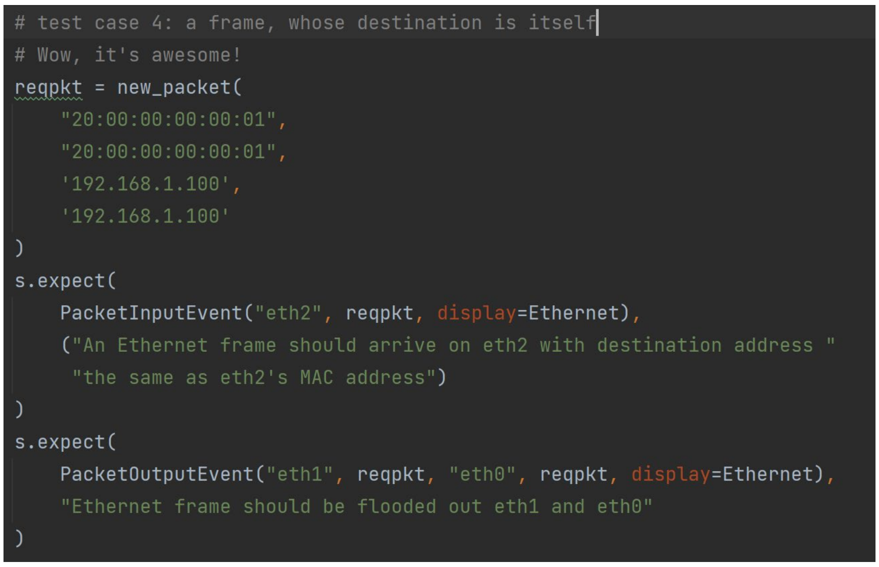
In the section Switchyard, we introduced how to write the test case. So here we have two options for you, choose **one** to implement. ✅ Then show the details of your test cases in your report.

- Create one test case by using the given function `new_packet` with different arguments,
- Or create one test case with your handmade packet.

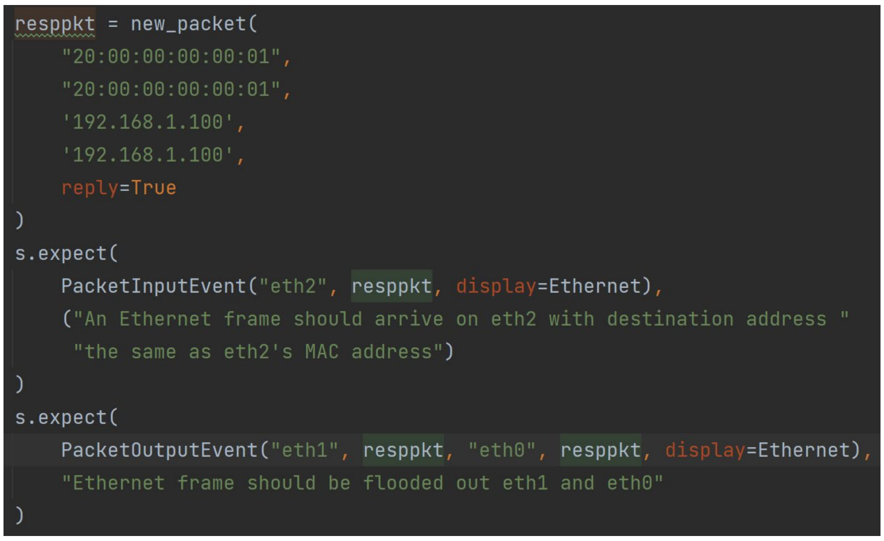
The file you need to modify is `testcases/myhub_testscenario.py`.

又到了愉快的 2 选 1 时间了。当然选 1 啦，重复造轮子是愚蠢的。

来一点骚操作：我们让节点 2 向其自己发送包，那么，集线器会向另外俩接口发出这个包；然后，节点 2 会自己回应自己，所以，几乎相同的过程会再次发生一遍。

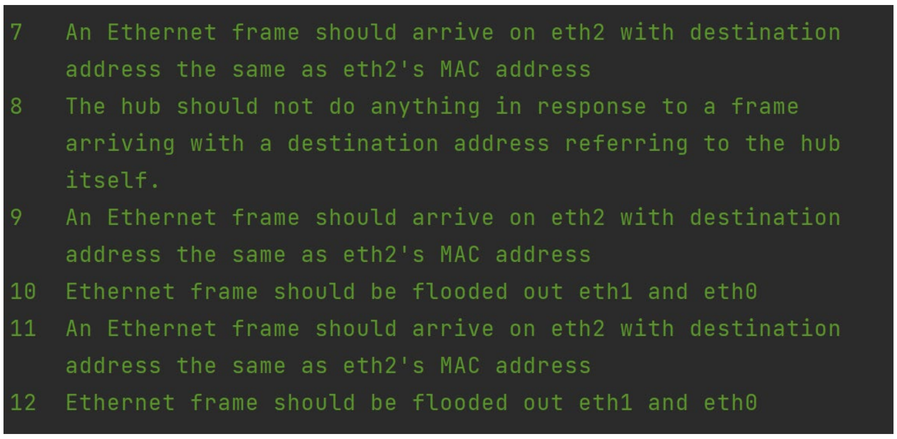


```python
# test case 4: a frame, whose destination is itself
# Wow, it's awesome!
reqpkt = new_packet(
    "20:00:00:00:00:01",
    "20:00:00:00:00:01",
    '192.168.1.100',
    '192.168.1.100'
)
s.expect(
    PacketInputEvent("eth2", reqpkt, display=Ethernet),
    ("An Ethernet frame should arrive on eth2 with destination address "
     "the same as eth2's MAC address")
)
s.expect(
    PacketOutputEvent("eth1", reqpkt, "eth0", reqpkt, display=Ethernet),
    "Ethernet frame should be flooded out eth1 and eth0"
)
```



```python
resppkt = new_packet(
    "20:00:00:00:00:01",
    "20:00:00:00:00:01",
    '192.168.1.100',
    '192.168.1.100',
    reply=True
)
s.expect(
    PacketInputEvent("eth2", resppkt, display=Ethernet),
    ("An Ethernet frame should arrive on eth2 with destination address "
     "the same as eth2's MAC address")
)
s.expect(
    PacketOutputEvent("eth1", resppkt, "eth0", resppkt, display=Ethernet),
    "Ethernet frame should be flooded out eth1 and eth0"
)
```

我们来跑一下玩玩。



```
7   An Ethernet frame should arrive on eth2 with destination
    address the same as eth2's MAC address
8   The hub should not do anything in response to a frame
    arriving with a destination address referring to the hub
    itself.
9   An Ethernet frame should arrive on eth2 with destination
    address the same as eth2's MAC address
10  Ethernet frame should be flooded out eth1 and eth0
11  An Ethernet frame should arrive on eth2 with destination
    address the same as eth2's MAC address
12  Ethernet frame should be flooded out eth1 and eth0
```

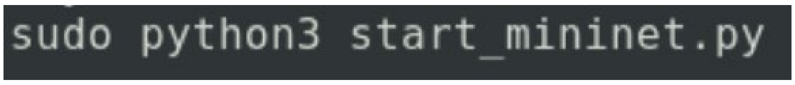
这就对哩!

4. Run your device in Mininet

In the section Switchyard, we introduced how to run Switchyard programs in Mininet. So run your new hub in your new topology and make sure it works. ✅ Show the procedure in your report.

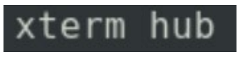
啊？这难道不是在之前跑过了吗？

先用 mininet 跑起来。



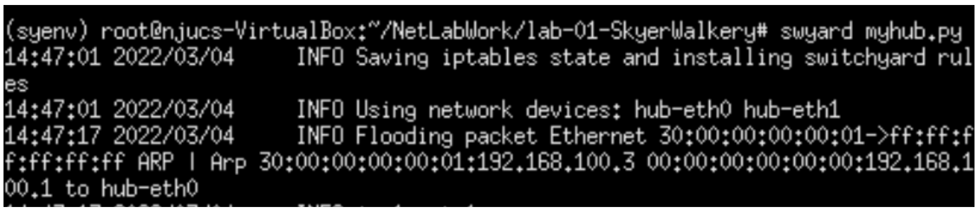
```
sudo python3 start_mininet.py
```

关注点放在 hub 上。



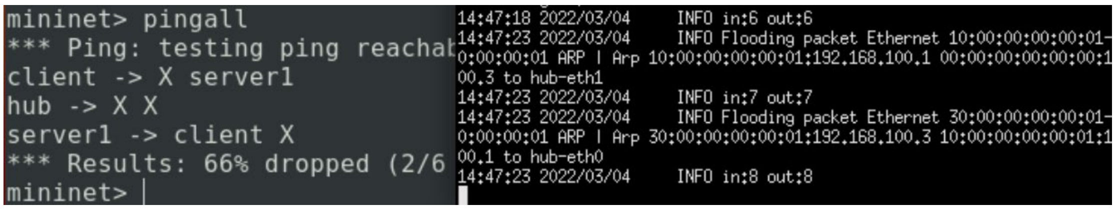
```
xterm hub
```

在新窗口运行 switchyard。



```
(syenv) root@njucs-VirtualBox:~/NetLabWork/lab-01-SkyerWalkery# swyard myhub.py
14:47:01 2022/03/04     INFO Saving iptables state and installing switchyard rul
es
14:47:01 2022/03/04     INFO Using network devices: hub-eth0 hub-eth1
14:47:17 2022/03/04     INFO Flooding packet Ethernet 30:00:00:00:00:01->ff:ff:f
f:ff:ff:ff ARP | Arp 30:00:00:00:00:01:192.168.100.3 00:00:00:00:00:00:192.168.1
00.1 to hub-eth0
```

pingall，得到运行结果。

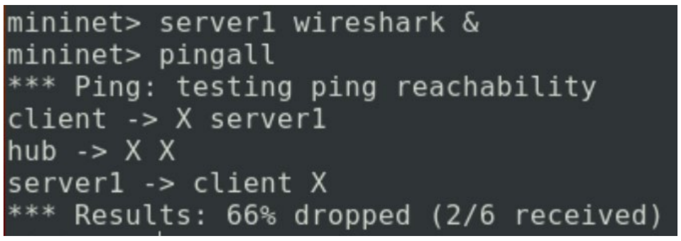


```
mininet> pingall
*** Ping: testing ping reachab
client -> X server1
hub -> X X
server1 -> client X
*** Results: 66% dropped (2/6
mininet> |
```

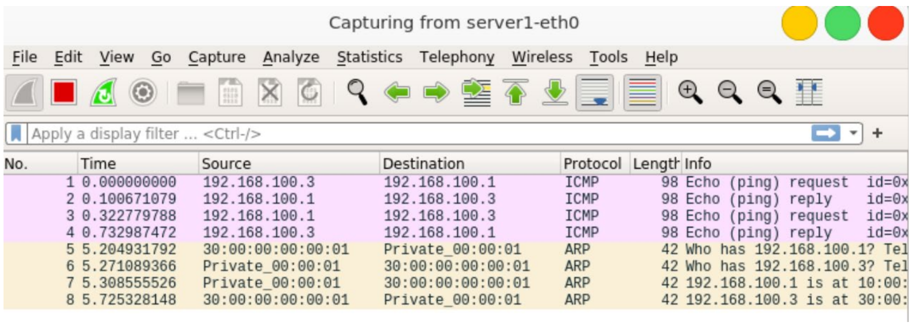
```
14:47:18 2022/03/04     INFO in:6 out:6
14:47:23 2022/03/04     INFO Flooding packet Ethernet 10:00:00:00:00:01-
0:00:00:01 ARP | Arp 10:00:00:00:00:01:192.168.100.1 00:00:00:00:00:00:1
00.3 to hub-eth1
14:47:23 2022/03/04     INFO in:7 out:7
14:47:23 2022/03/04     INFO Flooding packet Ethernet 30:00:00:00:00:01-
0:00:00:01 ARP | Arp 30:00:00:00:00:01:192.168.100.3 10:00:00:00:00:01:1
00.1 to hub-eth0
14:47:23 2022/03/04     INFO in:8 out:8
```

5. Capture using Wireshark

选择 server1，pingall，使用 wireshark 抓包。



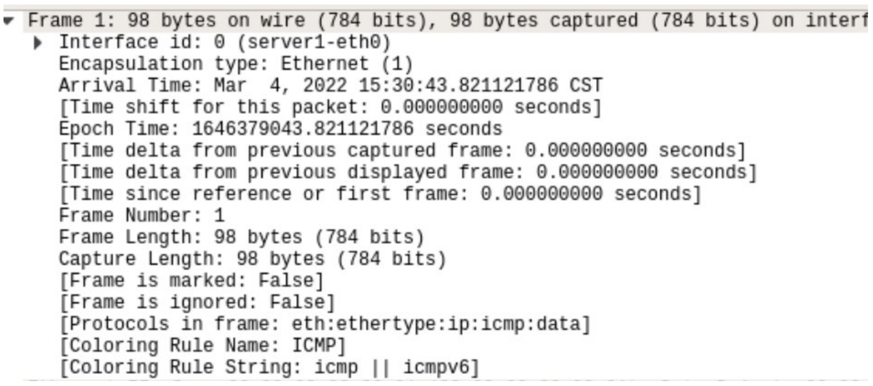
```
mininet> server1 wireshark &
mininet> pingall
*** Ping: testing ping reachability
client -> X server1
hub -> X X
server1 -> client X
*** Results: 66% dropped (2/6 received)
```


Capturing from server1-eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.   Time          Source             Destination        Protocol Length Info
    1 0.000000000   192.168.100.3      192.168.100.1      ICMP       98 Echo (ping) request  id=0x
    2 0.100671079   192.168.100.1      192.168.100.3      ICMP       98 Echo (ping) reply    id=0x
    3 0.322779788   192.168.100.1      192.168.100.3      ICMP       98 Echo (ping) request  id=0x
    4 0.732987472   192.168.100.3      192.168.100.1      ICMP       98 Echo (ping) reply    id=0x
    5 5.204931792   30:00:00:00:00:01  Private_00:00:01   ARP        42 Who has 192.168.100.1? Tel
    6 5.271089366   Private_00:00:01   30:00:00:00:00:01  ARP        42 Who has 192.168.100.3? Tel
    7 5.308555526   Private_00:00:01   30:00:00:00:00:01  ARP        42 192.168.100.1 is at 10:00:
    8 5.725328148   30:00:00:00:00:01  Private_00:00:01   ARP        42 192.168.100.3 is at 30:00:
```

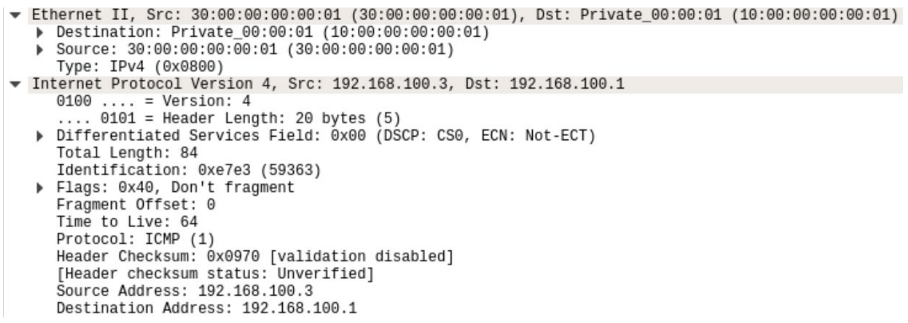
由于我们之前已经 pingall 过了，所以 server 和 client 都得知了对方的 MAC，而不会在一开始发送 ARP；后面的 ARP，经助教提示，应为节点向对方确认其地址是否发生改变而发送的。

点开一个详细看，首先是物理层帧的一些基本信息，如到达时间、大小等等；



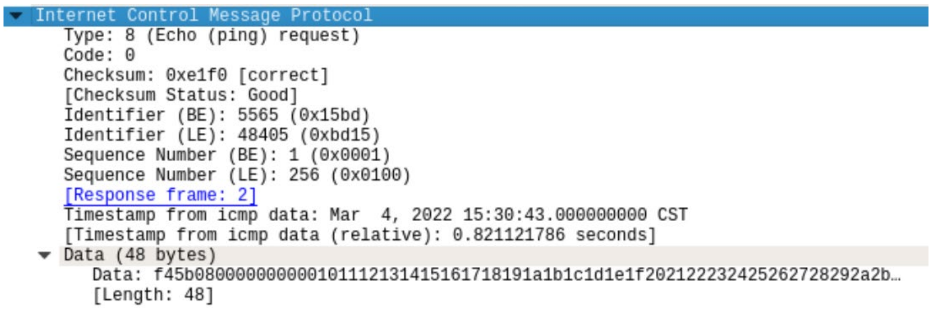
```
▾ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interf
  ▸ Interface id: 0 (server1-eth0)
    Encapsulation type: Ethernet (1)
    Arrival Time: Mar  4, 2022 15:30:43.821121786 CST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1646379043.821121786 seconds
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 98 bytes (784 bits)
    Capture Length: 98 bytes (784 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:icmp:data]
    [Coloring Rule Name: ICMP]
    [Coloring Rule String: icmp || icmpv6]
```

接着，分别是链路层和网络层的信息，包括源、目的地址（Mac, IP）、协议（IPv4）等。



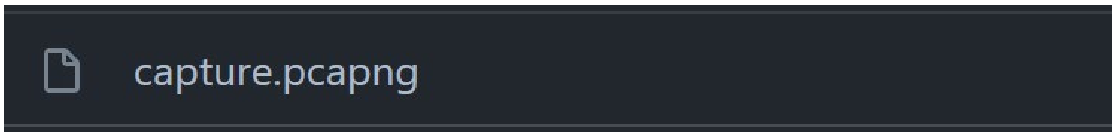
```
▾ Ethernet II, Src: 30:00:00:00:00:01 (30:00:00:00:00:01), Dst: Private_00:00:01 (10:00:00:00:00:01)
  ▸ Destination: Private_00:00:01 (10:00:00:00:00:01)
  ▸ Source: 30:00:00:00:00:01 (30:00:00:00:00:01)
    Type: IPv4 (0x0800)
▾ Internet Protocol Version 4, Src: 192.168.100.3, Dst: 192.168.100.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 84
    Identification: 0xe7e3 (59363)
  ▸ Flags: 0x40, Don't fragment
    Fragment Offset: 0
    Time to Live: 64
    Protocol: ICMP (1)
    Header Checksum: 0x0970 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.100.3
    Destination Address: 192.168.100.1
```

最后是 ICMP 协议。



```
▼ Internet Control Message Protocol
      Type: 8 (Echo (ping) request)
      Code: 0
      Checksum: 0xe1f0 [correct]
      [Checksum Status: Good]
      Identifier (BE): 5565 (0x15bd)
      Identifier (LE): 48405 (0xbd15)
      Sequence Number (BE): 1 (0x0001)
      Sequence Number (LE): 256 (0x0100)
      [Response frame: 2]
      Timestamp from icmp data: Mar  4, 2022 15:30:43.000000000 CST
      [Timestamp from icmp data (relative): 0.821121786 seconds]
   ▼ Data (48 bytes)
         Data: f45b0800000000000101112131415161718191a1b1c1d1e1f202122232425262728292a2b…
         [Length: 48]
```

我们把相关信息以 pcapng 文件导出。

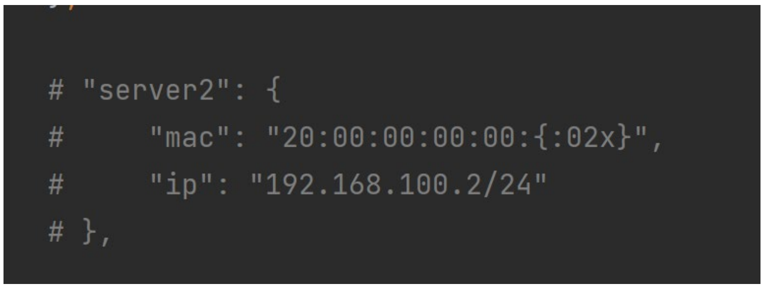


capture.pcapng

三、 核心代码

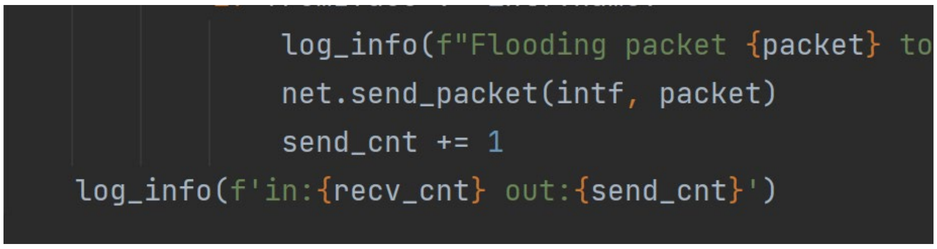
主要有两处：

删除 server2



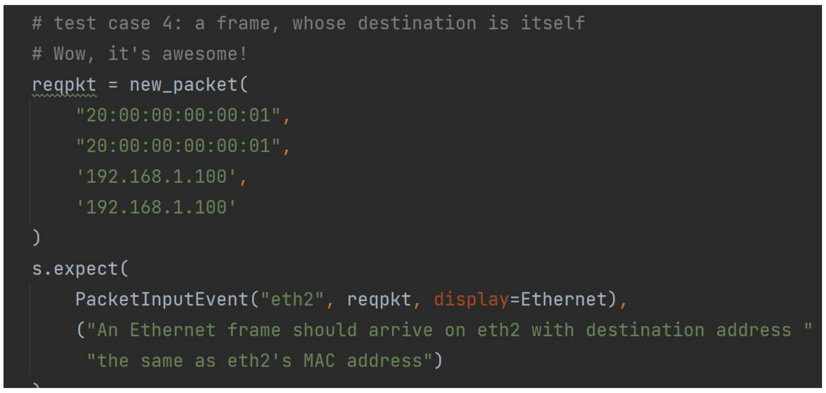
```
# "server2": {
#     "mac": "20:00:00:00:00:{:02x}",
#     "ip": "192.168.100.2/24"
# },
```

集线器对包的数数



```
                log_info(f"Flooding packet {packet} to
                net.send_packet(intf, packet)
                send_cnt += 1
    log_info(f'in:{recv_cnt} out:{send_cnt}')
```

自己的测试用例



```
# test case 4: a frame, whose destination is itself
# Wow, it's awesome!
reqpkt = new_packet(
    "20:00:00:00:00:01",
    "20:00:00:00:00:01",
    '192.168.1.100',
    '192.168.1.100'
)
s.expect(
    PacketInputEvent("eth2", reqpkt, display=Ethernet),
    ("An Ethernet frame should arrive on eth2 with destination address "
     "the same as eth2's MAC address")
```

四、 总结与感想

- 大部分时间用来学习这几件工具了，真正做的时间不多；
- 由于学的还不多，所以做得一知半解、似懂非懂；特别是抓包那块，待我去知乎和谷歌上好好看看。