# Computer Networks*
# *Lab I* LaTeX

* Teacher: Chen Tian, Wenzhong Li. TAs: Yuchu Fang, Jianhui Dduan, Xiang Li.

1st 张逸凯 171840708 (转专业到计科, 非重修)
*Department of Computer Science and Technology*
*Nanjing University*
zykhelloha@gmail.com

*171840708 张逸凯 Lab1*

---

Task 3: Your Modification
- Step1 Delete `server2` in the topology:
  - 如何实现:
- Step 2 Count how many packets pass through a hub both in and out
  - 如何实现:
- Step 3: Create one test case by using the given function `mk_pkt` with different arguments
  - test case1的理解:
  - test case2的理解:
  - test case3的理解:
  - 如何实现: Create one test case by using the given function `mk_pkt` with different arguments
- Step 4: Run your device in Mininet
  - 如何实现:
- Step 5: Capture using Wireshark
  - 如何实现:
    - wireshark通过颜色帮助识别the types of traffic:
    - 使用filter来只看想看的
- 开始抓包:
  - 接下来我展示一下其他的包, 具体信息阅读类似上面, 相同的不再赘述:
- 总结

# Task 3: Your Modification

*接下来将**分小节**展示详细实验过程, 谢谢助教哥的批改!***非常抱歉没有完全使用实验报告模板**, 但是似乎分小节叙述能更充分展示, 见谅!

## Step1 Delete `server2` in the topology:

### 如何实现:

在阅读手册以及官方文档之后, Delete server2 in the topology可以通过更改`start_mininet.py`中`def __init__(self, args):`关于网络拓扑结构的定义, 并在之后有关设置MAC地址的函数中, 注释相关代码即可:

```
        nodeconfig = {'cpu':-1}
        self.addHost('server1', **nodeconfig)
        # self.addHost('server2', **nodeconfig) # 1. Delete server2
        self.addHost('hub', **nodeconfig)
        self.addHost('client', **nodeconfig)

        # for node in ['server1','server2','client']: # 1. Delete server2
        for node in ['server1', 'client']:
            # all links are 10Mb/s, 100 millisecond prop delay
            self.addLink(node, 'hub', bw=10, delay='100ms')
```

```
def setup_addressing(net):      You, 20 hours ago • my init commit at 3-4
    # {:02x} 数转为两位16进制表示.
    reset_macs(net, 'server1', '10:00:00:00:00:{:02x}')
    # reset_macs(net, 'server2', '20:00:00:00:00:{:02x}') # 1. Delete server2
    reset_macs(net, 'client', '30:00:00:00:00:{:02x}')
    reset_macs(net, 'hub', '40:00:00:00:00:{:02x}')
    set_ip(net, 'server1','hub','192.168.100.1/24')
    # set_ip(net, 'server2','hub','192.168.100.2/24') # 1. Delete server2
    set_ip(net, 'client','hub','192.168.100.3/24')
```

经过测试, 我们可以发现这样成功Delete server2 in the topology:

```
*** Starting CLI:
mininet> net
client client-eth0:hub-eth1
hub hub-eth0:server1-eth0 hub-eth1:client-eth0
server1 server1-eth0:hub-eth0
mininet> nodes
available nodes are:
client hub server1
mininet>
```

## Step 2 Count how many packets pass through a hub both in and out

> You need to log it out every time you receive one packet with the format of each line `<timestamp> 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 `1583314030.0679464 in:1 out:2`.

上面这个例子是因为`hub`收到一个包后, 如果`eth.dst in mymacs:`就是发向自己的, 那么hub就不会通过端口发出去, 反之会从例子中的从另外两个口转出去.

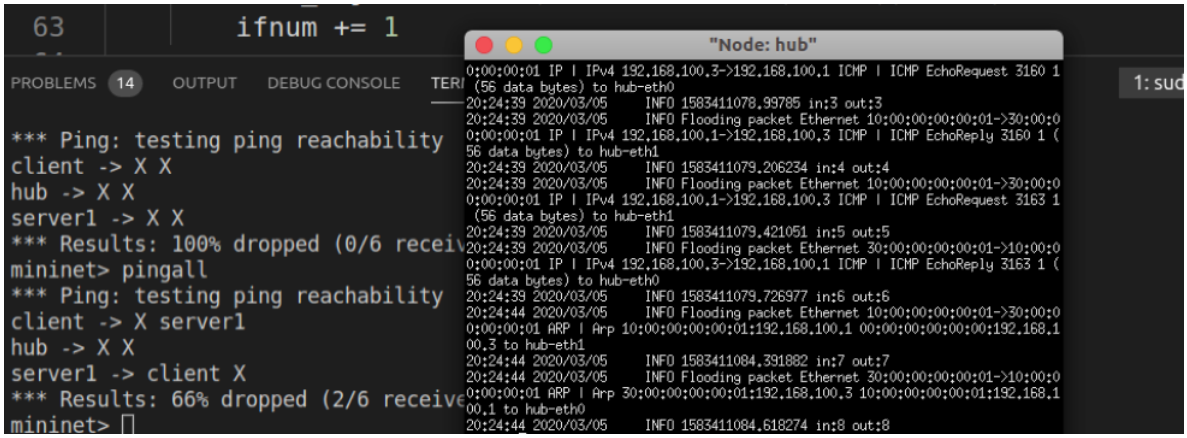
### 如何实现:

在接收到包, 在发送包的`send_packet`部分记录发出去的数量:

```
    # 主循环：
    while True:
        try:
            # 获得最先到达的包， 并返回 时间和对应的网卡
            timestamp,dev,packet = net.recv_packet()
            inPkNum += 1
        except NoPackets:
            continue
        except Shutdown:
```

```
        for intf in my_interfaces:
            if dev != intf.name:
                log_info ("Flooding packet {} to {}".format(packet, intf.name))
                # 将包从对应网卡发出去
                net.send_packet(intf, packet)
                outPkNum += 1      You, 9 minutes ago • Uncommitted changes
    log_info("{} in:{} out:{}".format(timestamp, inPkNum, outPkNum))
```

但是因为我第一题做的是Delete server2 in the topology, 所以略有差别:




使用 `hubtest.py` 测试可以更清楚地发现结果是正确的, 因为testcase里面有设计的发到hub的包, 所以in和out是有差值的:

```
(syenv) kai@kai-virtual-machine:~/switchyard/lab_1$ swyard -t hubtests.py myhub.py
20:34:23 2020/03/05     INFO Starting test scenario hubtests.py
20:34:23 2020/03/05     INFO Flooding packet Ethernet 30:00:00:00:00:02->ff:ff:ff:ff:ff:ff IP | IPv4 172.16.42.2->255.255.255
.255 ICMP | ICMP EchoRequest 0 0 (0 data bytes) to eth0
20:34:23 2020/03/05     INFO Flooding packet Ethernet 30:00:00:00:00:02->ff:ff:ff:ff:ff:ff IP | IPv4 172.16.42.2->255.255.255
.255 ICMP | ICMP EchoRequest 0 0 (0 data bytes) to eth2
20:34:23 2020/03/05     INFO 0.0 in:1 out:2
20:34:23 2020/03/05     INFO Flooding packet Ethernet 20:00:00:00:00:01->30:00:00:00:00:02 IP | IPv4 192.168.1.100->172.16.42
.2 ICMP | ICMP EchoRequest 0 0 (0 data bytes) to eth1
20:34:23 2020/03/05     INFO Flooding packet Ethernet 20:00:00:00:00:01->30:00:00:00:00:02 IP | IPv4 192.168.1.100->172.16.42
.2 ICMP | ICMP EchoRequest 0 0 (0 data bytes) to eth2
20:34:23 2020/03/05     INFO 2.0 in:2 out:4
20:34:23 2020/03/05     INFO Flooding packet Ethernet 30:00:00:00:00:02->20:00:00:00:00:01 IP | IPv4 172.16.42.2->192.168.1.1
00 ICMP | ICMP EchoReply 0 0 (0 data bytes) to eth0
20:34:23 2020/03/05     INFO Flooding packet Ethernet 30:00:00:00:00:02->20:00:00:00:00:01 IP | IPv4 172.16.42.2->192.168.1.1
00 ICMP | ICMP EchoReply 0 0 (0 data bytes) to eth2
20:34:23 2020/03/05     INFO 4.0 in:3 out:6
20:34:23 2020/03/05     INFO Received a packet intended for me
20:34:23 2020/03/05     INFO 6.0 in:4 out:6

Results for test scenario hub tests: 8 passed, 0 failed, 0 pending
```

## Step 3: Create one test case by using the given function `mk_pkt` with different arguments

```
def hub_tests():
    s = TestScenario("hub tests")
    # 加入three device interfaces with name and MAC.
    s.add_interface('eth0', '10:00:00:00:00:01')
    s.add_interface('eth1', '10:00:00:00:00:02')
    s.add_interface('eth2', '10:00:00:00:00:03')
```

```
(syenv) kai@kai-virtual-machine:~/switchyard/lab_1$ swyard -t hubtests.py myhub.py
12:26:47 2020/03/06     INFO Starting test scenario hubtests.py
12:26:47 2020/03/06     INFO mymacs: [EthAddr('10:00:00:00:00:01'), EthAddr('10:00:00:00:00:02'), EthAddr('10:00:00:00:00:03'
)]
12:26:47 2020/03/06     INFO Flooding packet Ethernet 30:00:00:00:00:02->ff:ff:ff:ff:ff:ff IP | IPv4 172.16.42.2->255.255.255
```

由上述代码和我做的一些输出`mymacs:`可以知道hub的mac地址是多少.

## test case1的理解:

包从`30:00:00:00:00:02`被广播之后, `eth1`模拟包到达, 其他两个端口模拟包发出.

## test case2的理解:

`hub`只是一条线, 所以包的目的mac写的是下一跳的mac, 而不是hub的mac, 这样test case2发就不难理解了. 除了指定给hub接口的地址外, 任何单播地址的帧都应该被发送到除入口之外的所有端口. 要给到下一跳.

## test case3的理解:

传到了自己的mac地址, 由myhub的代码可以知道不会转发.

## 如何实现: Create one test case by using the given function `mk_pkt` with different arguments

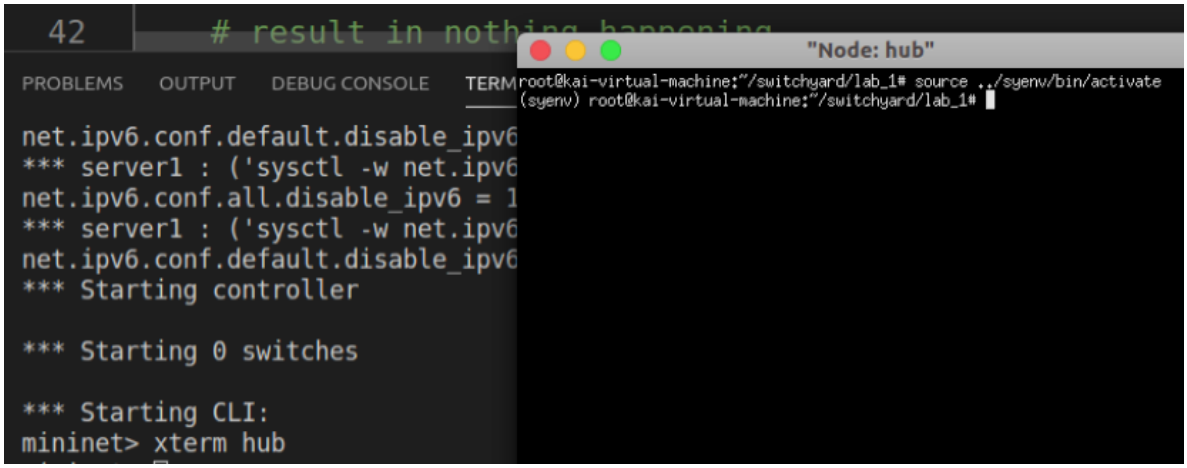
```
lab_1 > hubtests.py > hub_tests
47      # test case 4: a frame with dest address of one of the
        interfaces should
48      # result in nothing happening
49      reqpkt = mk_pkt("20:00:00:00:00:01", "10:00:00:00:00:02", '192.
        168.1.100','172.16.42.2')
50      s.expect(PacketInputEvent("eth1", reqpkt, display=Ethernet),
        "An Ethernet frame should arrive on eth1 with destination
        address the same as eth1's MAC address")
51      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.")
52      return s        You, 2 days ago • my init commit at 3-4

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                    1: bash

20:42:44 2020/03/06     INFO Flooding packet Ethernet 20:00:00:00:00:01->30:00:00:00:00:02 IP | IPv4 1
92.168.1.100->172.16.42.2 ICMP | ICMP EchoRequest 0 0 (0 data bytes) to eth2
20:42:44 2020/03/06     INFO 2.0 in:2 out:4
20:42:44 2020/03/06     INFO Flooding packet Ethernet 30:00:00:00:00:02->20:00:00:00:00:01 IP | IPv4 1
72.16.42.2->192.168.1.100 ICMP | ICMP EchoReply 0 0 (0 data bytes) to eth0
20:42:44 2020/03/06     INFO Flooding packet Ethernet 30:00:00:00:00:02->20:00:00:00:00:01 IP | IPv4 1
72.16.42.2->192.168.1.100 ICMP | ICMP EchoReply 0 0 (0 data bytes) to eth2
20:42:44 2020/03/06     INFO 4.0 in:3 out:6
20:42:44 2020/03/06     INFO Received a packet intended for me
20:42:44 2020/03/06     INFO 6.0 in:4 out:6
20:42:45 2020/03/06     INFO Received a packet intended for me
20:42:45 2020/03/06     INFO 6.0 in:5 out:6
```

传到带有hub的接口(eth1), 由myhub的代码可以知道不会发生什么, 红圈的记录也显示了这一点.

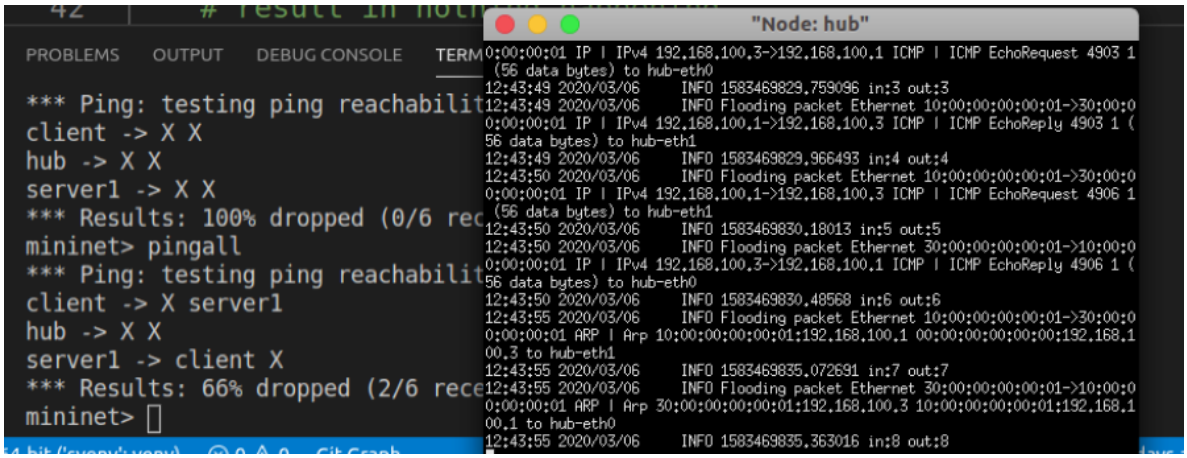
## Step 4: Run your device in Mininet

### 如何实现：

开启`xterm`并激活环境:




在`xterm`键入`swyard myhub.py`, 并在`CLI`键入`pingall`:




可以验证new topology works了!

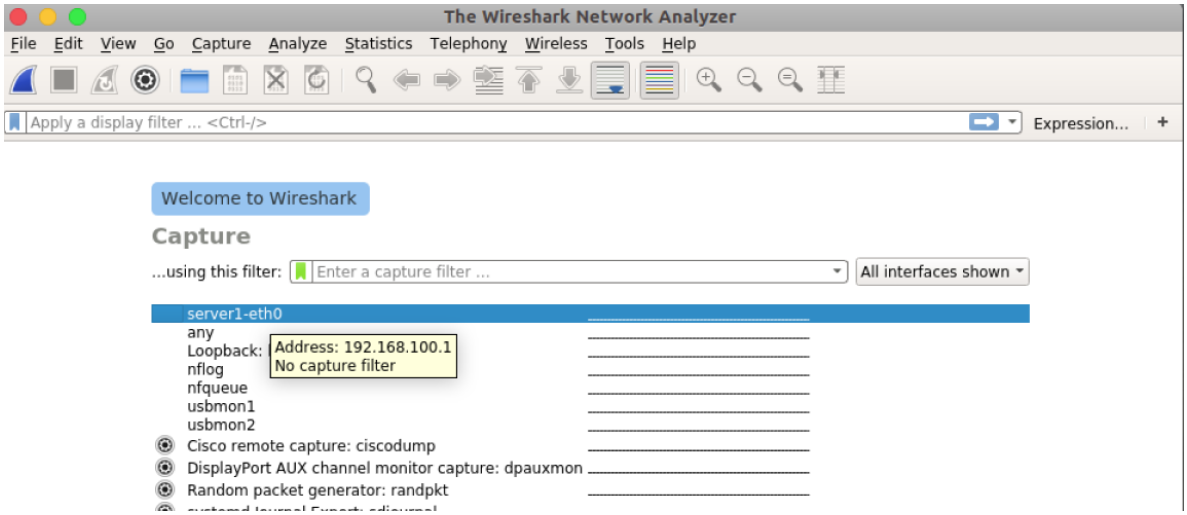
## Step 5: Capture using Wireshark

### 如何实现：

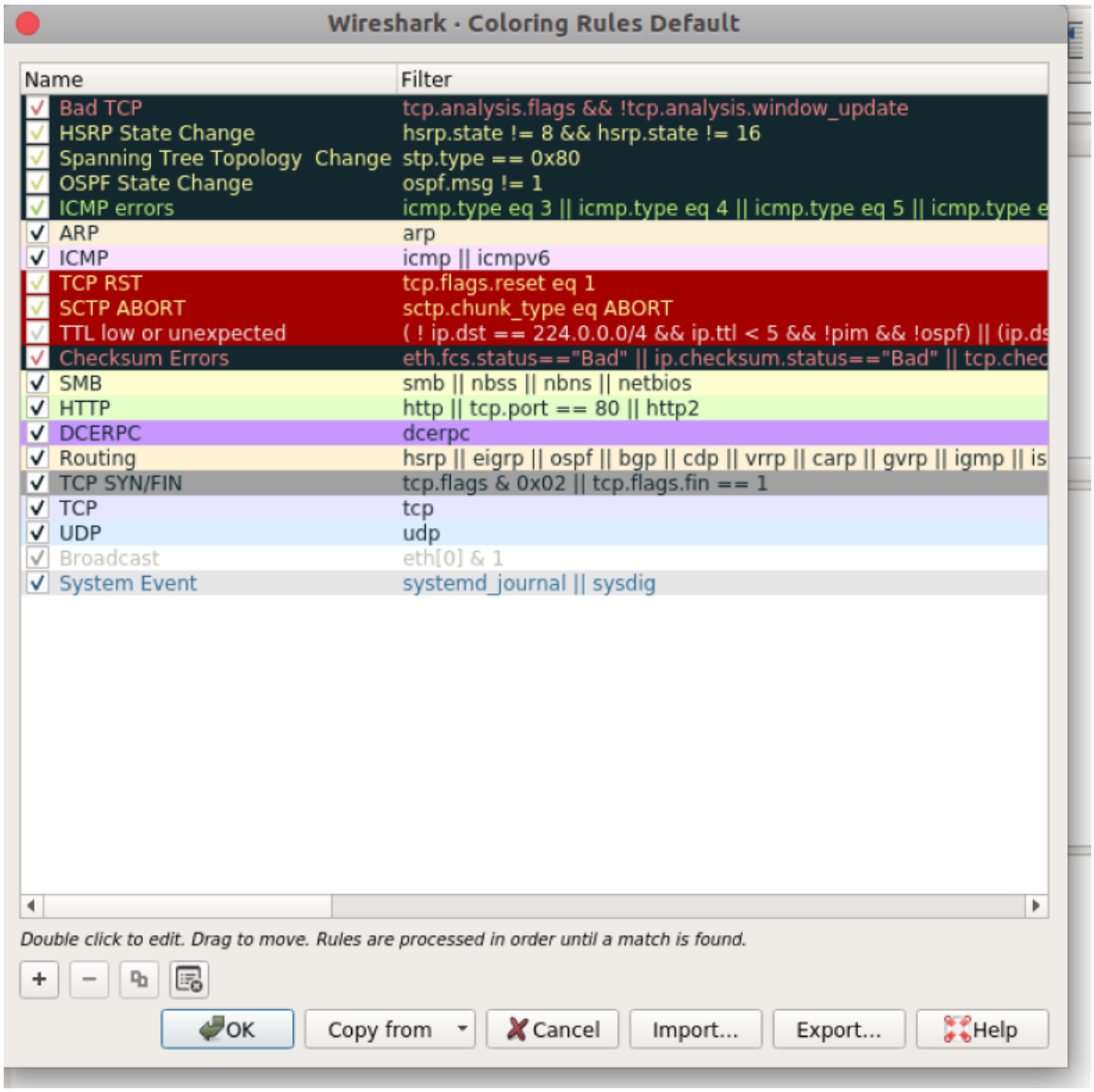
`CLI`键入, 来open wireshark in `server1`(a host which meet the requirements):

```
server1 wireshark &
```

选择traffic

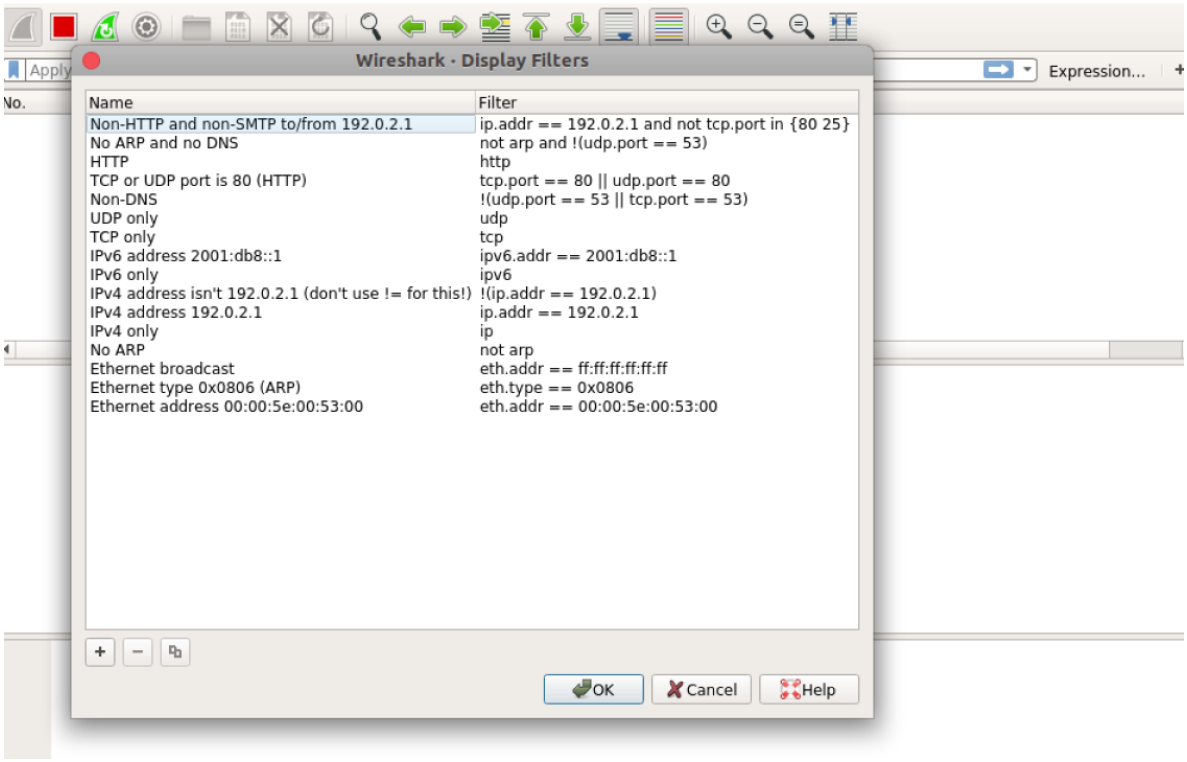



**wireshark通过颜色帮助识别the types of traffic:**

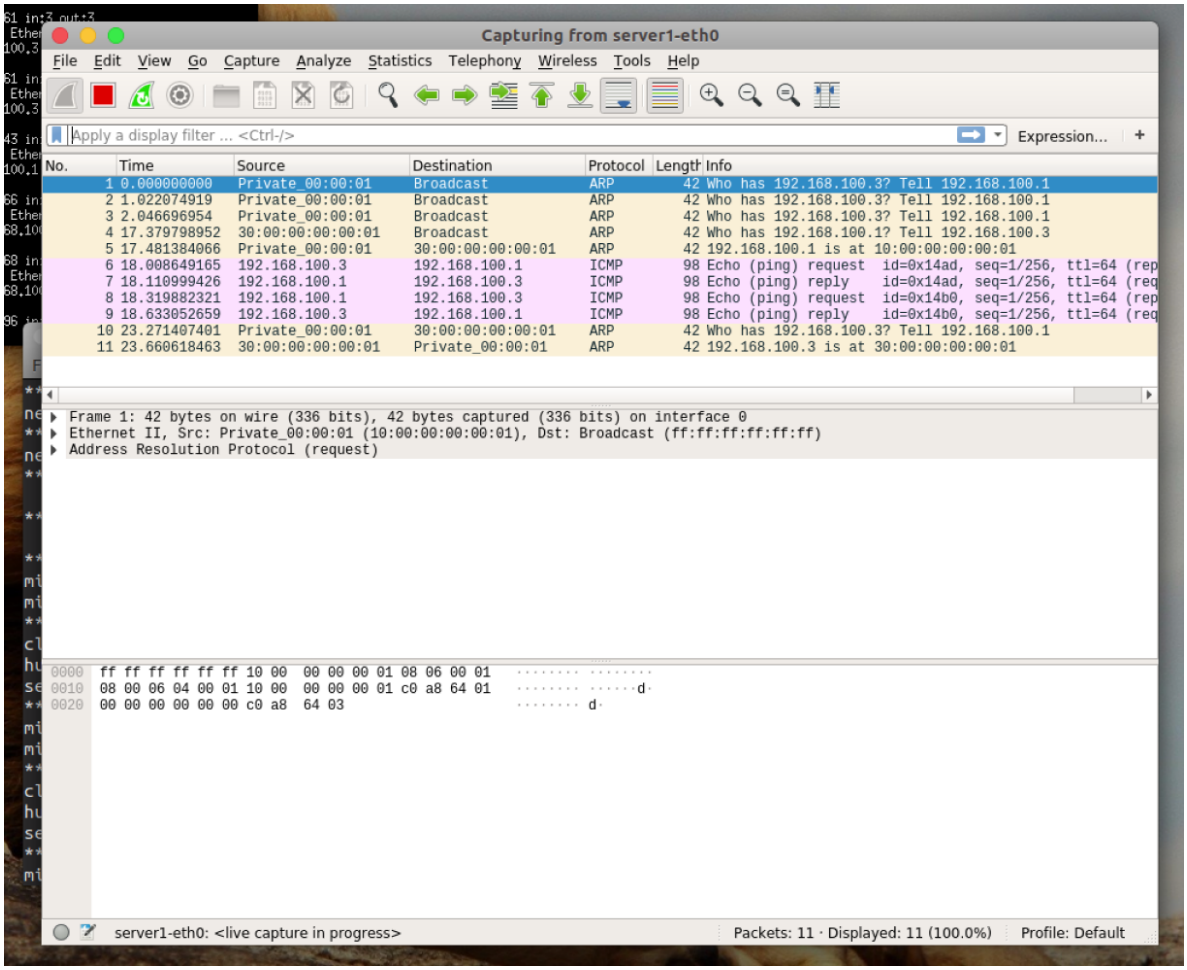



淡紫色表示TCP流量, 淡蓝色表示UDP流量, 黑色表示有错误的数据包.

使用**filter**来只看想看的


Wireshark · Display Filters
Name
Filter
Non-HTTP and non-SMTP to/from 192.0.2.1
ip.addr == 192.0.2.1 and not tcp.port in {80 25}
No ARP and no DNS
not arp and !(udp.port == 53)
HTTP
http
TCP or UDP port is 80 (HTTP)
tcp.port == 80 || udp.port == 80
Non-DNS
!(udp.port == 53 || tcp.port == 53)
UDP only
udp
TCP only
tcp
IPv6 address 2001:db8::1
ipv6.addr == 2001:db8::1
IPv6 only
ipv6
IPv4 address isn't 192.0.2.1 (don't use != for this!)
!(ip.addr == 192.0.2.1)
IPv4 address 192.0.2.1
ip.addr == 192.0.2.1
IPv4 only
ip
No ARP
not arp
Ethernet broadcast
eth.addr == ff:ff:ff:ff:ff:ff
Ethernet type 0x0806 (ARP)
eth.type == 0x0806
Ethernet address 00:00:5e:00:53:00
eth.addr == 00:00:5e:00:53:00
OK
Cancel
Help

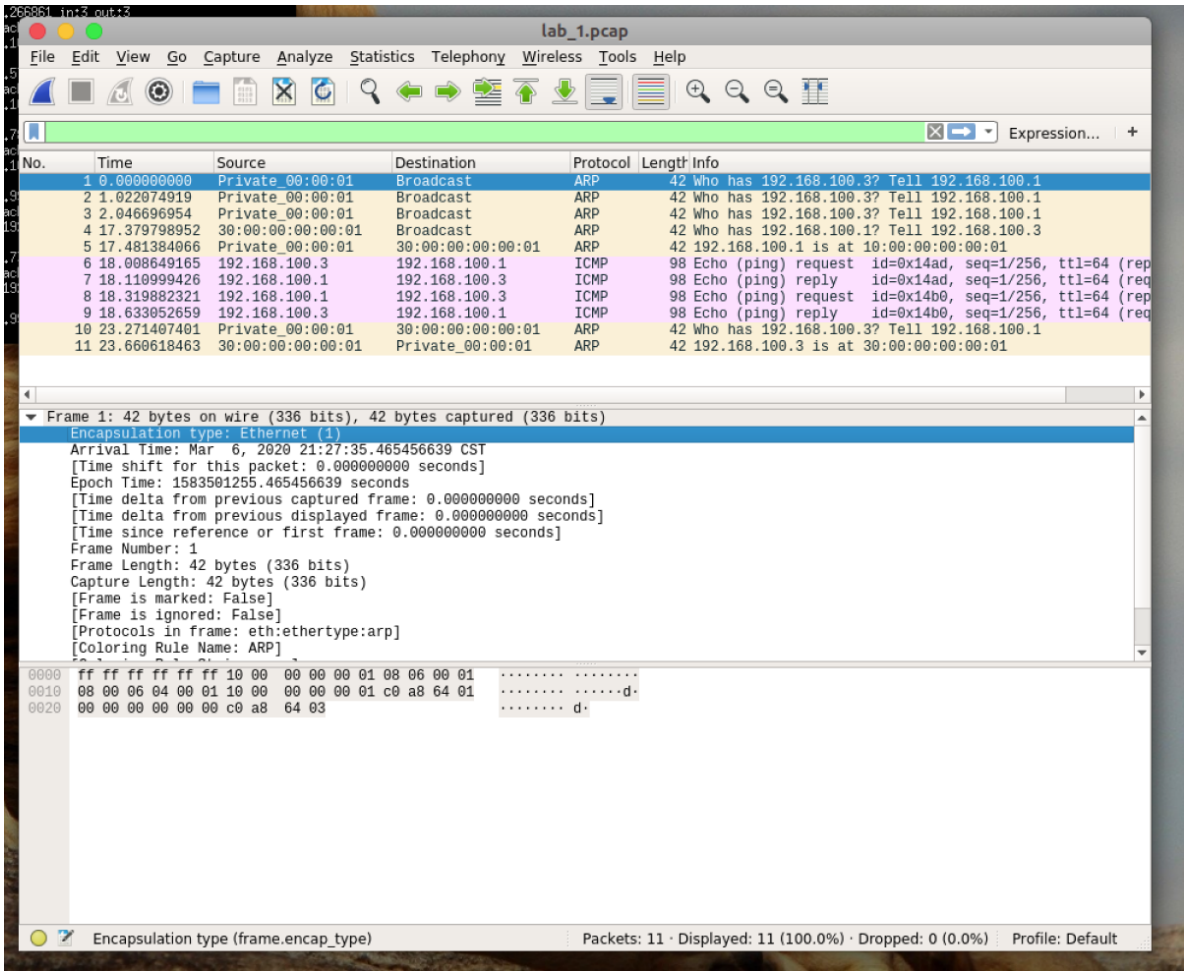
# 开始抓包：


Capturing from server1-eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
No. Time Source Destination Protocol Length Info
1 0.000000000 Private_00:00:01 Broadcast ARP 42 Who has 192.168.100.3? Tell 192.168.100.1
2 1.022074919 Private_00:00:01 Broadcast ARP 42 Who has 192.168.100.3? Tell 192.168.100.1
3 2.046696954 Private_00:00:01 Broadcast ARP 42 Who has 192.168.100.3? Tell 192.168.100.1
4 17.379798952 30:00:00:00:00:01 Broadcast ARP 42 Who has 192.168.100.1? Tell 192.168.100.3
5 17.481384066 Private_00:00:01 30:00:00:00:00:01 ARP 42 192.168.100.1 is at 10:00:00:00:00:01
6 18.008649165 192.168.100.3 192.168.100.1 ICMP 98 Echo (ping) request id=0x14ad, seq=1/256, ttl=64 (rep
7 18.110999426 192.168.100.1 192.168.100.3 ICMP 98 Echo (ping) reply id=0x14ad, seq=1/256, ttl=64 (req
8 18.319882321 192.168.100.1 192.168.100.3 ICMP 98 Echo (ping) request id=0x14b0, seq=1/256, ttl=64 (rep
9 18.633052659 192.168.100.3 192.168.100.1 ICMP 98 Echo (ping) reply id=0x14b0, seq=1/256, ttl=64 (req
10 23.271407401 Private_00:00:01 30:00:00:00:00:01 ARP 42 Who has 192.168.100.3? Tell 192.168.100.1
11 23.660618463 30:00:00:00:00:01 Private_00:00:01 ARP 42 192.168.100.3 is at 30:00:00:00:00:01
Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
Ethernet II, Src: Private_00:00:01 (10:00:00:00:00:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Address Resolution Protocol (request)
0000 ff ff ff ff ff ff 10 00 00 00 00 01 08 06 00 01
0010 08 00 06 04 00 01 10 00 00 00 00 01 c0 a8 64 01
0020 00 00 00 00 00 00 c0 a8 64 03
server1-eth0: <live capture in progress>
Packets: 11 · Displayed: 11 (100.0%)
Profile: Default

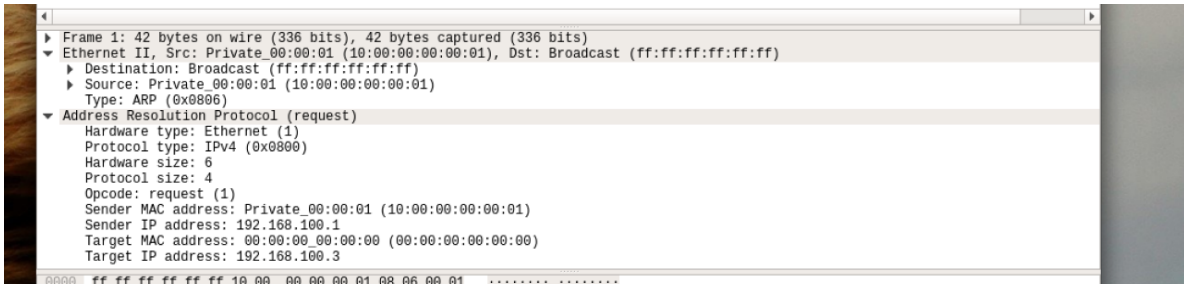
首先在我的拓扑结构里面pingall, 由手册知道这是100% drop的, 所以抓到的只有`ARP`协议的广播的包. 然后Then run your hub code to the device you want. 就是使用`swyard myhub.py`进行, 我们发现received了一些包.

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

观察myhub的代码就知道, 从server1发的包是会被转发的, 所以client(另一个结点)可以, 反之亦然, 理清收到的包是来自哪里后, 我们来深入看一下wireshark的详细内容.



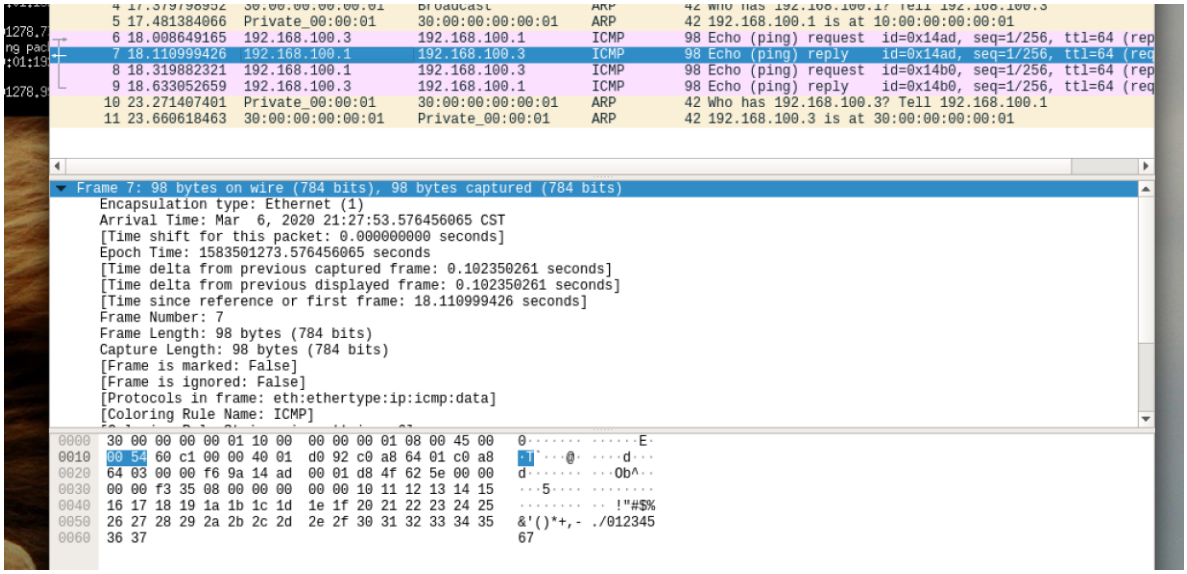

- encapsulation type ethernet (1), 它是一个Wireshark-internal值，表示所讨论的包的特定链接层报头类型.
- Arrival time: 是它的到达时间.
- Epoch time: 是自1970年1月1日以来的秒数.
- Time delta: 上面的都是零是因为它是第一个包.
- Frame的属性: 比如Frame Length, 协议等信息都可以在上面看到.

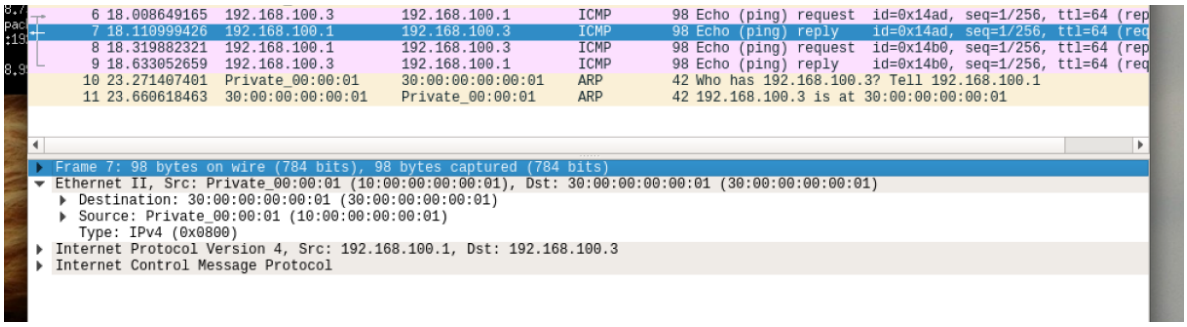



继续:

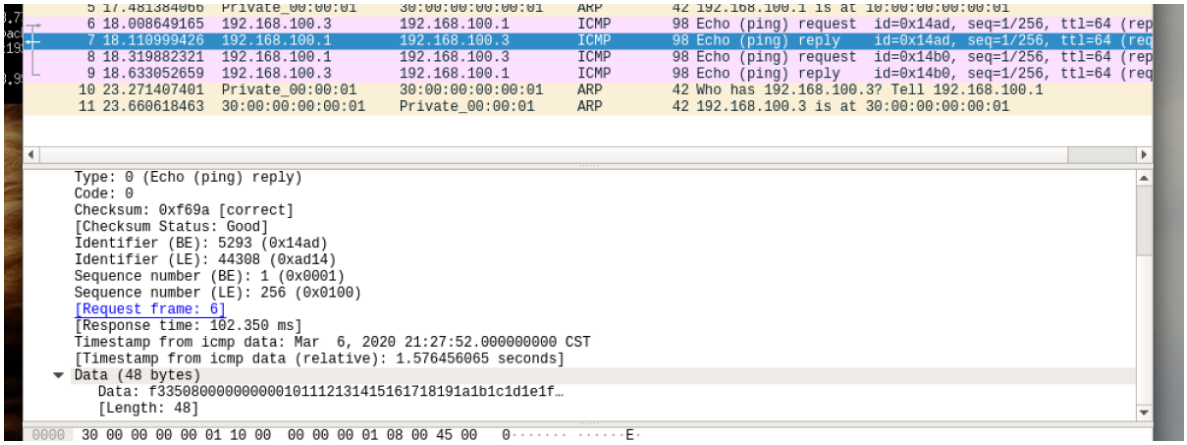
- 关于destination的相关信息, 这个是一个broadcast的包, 在下面我还会展示其他类型的包. MAC地址是FF..
- ARP 协议: 这里似乎可以看到一个简易的ARP table, 但是只有发送者和接受者, 因为这不是在主机上的, 如果是主机上的应还有TTL.

## 接下来我展示一下其他的包，具体信息阅读类似上面，相同的不再赘述：




这里就可以很清晰地看到**类似**我们在`mk_pkt`中设置的那些参数. 这也很好地解释了为什么我们使用自己写的hub就不会100%drop, 因为转发规则和接口的MAC被我们设置了.




这里有data哦, 其他都是读名字可以知道是什么的, 注意有一个Identifier, BE和LE代表大端和小端, 根据生成标识符和序列号的操作系统, 为了更容易地检查丢失的序列(或进程ID在某些操作系统上的标识符). 以便这些序列号从一个ICMP echo请求/应答递增到下一个ICMP时更容易遵循.

# 总结

还是有一些难点的, 比如那三个文件的关系, 运行时为什么本身会drop 100%, 为什么加上自己的hub结构就不会.

我会继续努力的! 😄