南京大学本科生实验报告

课程名称：**计算机网络**　　任课教师：[redacted]　　助教：

学院	计算机科学与技术系	专业（方向）	计算机科学与技术
学号		姓名	
Email		开始/完成日期	[redacted]

1.实验名称

Lab 1: Switchyard & Mininet

2.实验目的

此次实验主要介绍了整个计算机网络实验重所需要的所有的准备工作，并且能够通过学习指导手册来根据要求来对老师提供的一些例子进行一定程度的修改。

3.实验内容

- **Preparations for the lab**
- **Play the Tutorial Again**
  - **Step 1: Modify the Mininet topology**
  - **Step 2: Modify the logic of a device**
  - **Step 3: Modify the test scenario of a device**
  - **Step 4: Run your device in Mininet**
  - **Step 5: Capture using Wireshark**

4.实验结果

注：为了方便，此部分同时也展开如何进行实验的，以及对于实验中的为何需要这么做的一定的解释。

STEP 1:Modify the Mininet topology

由于我们需要做的事情是**delete `server2` in the topology**，加上之前的指导手册的提及，由此可知，此项任务的关键是**start_mininet.py**程序，着眼于**main**函数：

```
def main():
    topo = PySwitchTopo(args)
    net = Mininet(controller=None, topo=topo, link=TCLink, cleanup=True)
    setup_addressing(net)
    disable_ipv6(net)
    net.interact()
```

由此可知，我们需要修改的是**topo**这一变量，但是**topo**又是类**PySwitchTopo**的一个实例，这样就追溯到类**PySwitchTopo**的类定义了，其类定义如下：

```
class PySwitchTopo(Topo):
    def __init__(self, args):
        # Add default members to class.
        super(PySwitchTopo, self).__init__()
        # Host and link configuration
        #
        #
        #   server1
        #          \
        #           hub----client
        #          /
        #   server2
        #
        nodeconfig = {"cpu": -1}
        for node in nodes.keys():
            self.addHost(node, **nodeconfig)
        for node in nodes.keys():
            # all links are 10Mb/s, 100 millisecond prop delay
            if node != "hub":
                self.addLink(node, "hub", bw=10, delay="100ms")
```

由上述代码可知，类**PySwitchTopo**是继承于**Topo**类的，根据实验指导手册上面提供的 Mininet Walkthrough，我找到了**Python API**中关于**Topo类的定义**。上面代码所使用的**api**的解释如下：

```
class Topo( object ):
    "Data center network representation for structured multi-trees."

    #.....

    def addHost( self, name, **opts ):
        """Convenience method: Add host to graph.
           name: host name
           opts: host options
           returns: host name"""
        if not opts and self.hopts:
            opts = self.hopts
```

```
        return self.addNode( name, **opts )

    def addLink( self, node1, node2, port1=None, port2=None,
                 key=None, **opts ):
        """node1, node2: nodes to link together
           port1, port2: ports (optional)
           opts: link options (optional)
           returns: link info key"""
        if not opts and self.lopts:
            opts = self.lopts
        port1, port2 = self.addPort( node1, node2, port1, port2 )
        opts = dict( opts )
        opts.update( node1=node1, node2=node2, port1=port1, port2=port2 )
        return self.g.add_edge(node1, node2, key, opts )
    # ....
```

经过此分析可知，按我们代码中所使用：**addHost**方法就是将**nodes**变量中所有的**host**加入到整张**graph**中；**addLink**方法就是将**nodes**变量中所有除了**hub**的**node**全部与**hub**链接起来。

这样看来，如果想要在整个**topology**中将**server2**删除的话，就可以将**nodes**变量中的**server2**节点删除即可满足条件。具体的操作如下：

```
nodes = {
    "server1": {
        "mac": "10:00:00:00:00:{:02x}",
        "ip": "192.168.100.1/24"
    },
    #"server2": {
    #    "mac": "20:00:00:00:00:{:02x}",      You, 3 days ago • lab1_step1
    #    "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}",
    }
}
```

这样改动就完成了，利用**python**启动**start_mininet.py**来观察是否删除**server2**成功：

```
(syenv) njucs@njucs-VirtualBox:[redacted]$ sudo python start_mininet.py
[sudo] password for njucs:
*** Creating network
*** Adding hosts:
client hub server1
*** Adding switches:

*** Adding links:
(10.00Mbit 100ms delay) (10.00Mbit 100ms delay) (client, hub) (10.00Mbit 100ms delay) (10.00Mbit 100ms delay) (server1, hub)
*** Configuring hosts
client hub server1
('client', <TCIntf client-eth0>, '30:00:00:00:00:01')
('server1', <TCIntf server1-eth0>, '10:00:00:00:00:01')
('hub', <TCIntf hub-eth0>, '40:00:00:00:00:01')
('hub', <TCIntf hub-eth1>, '40:00:00:00:00:02')
*** client : ('sysctl -w net.ipv6.conf.all.disable_ipv6=1',)
net.ipv6.conf.all.disable_ipv6 = 1
*** client : ('sysctl -w net.ipv6.conf.default.disable_ipv6=1',)
net.ipv6.conf.default.disable_ipv6 = 1
*** hub : ('sysctl -w net.ipv6.conf.all.disable_ipv6=1',)
net.ipv6.conf.all.disable_ipv6 = 1
*** hub : ('sysctl -w net.ipv6.conf.default.disable_ipv6=1',)
net.ipv6.conf.default.disable_ipv6 = 1
*** server1 : ('sysctl -w net.ipv6.conf.all.disable_ipv6=1',)
net.ipv6.conf.all.disable_ipv6 = 1
*** server1 : ('sysctl -w net.ipv6.conf.default.disable_ipv6=1',)
net.ipv6.conf.default.disable_ipv6 = 1
*** Starting controller

*** Starting 0 switches

*** Starting CLI:
mininet> nodes
available nodes are:
client hub server1
mininet>
```

由**nodes**命令展示可知，**server2**被成功删除，结果正确。

STEP 2:Modify the logic of a device

正如指导手册中写的这样

> 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.

当**hub**收到一个**packet**时，且这个**packet**不是发送给**hub**的时候，就会通过两个口转发出去，结合下图可视化这种过程

```
server1
          \
           hub----client
          /
server2
```

所以就会有`in:1 out:2`这种结果的诞生。

现在回到代码的修改部分，因为这里主要涉及的是hub的内容的增加，所以应该涉及的代码改动就是**myhub.py**了。

利用助教哥哥提供的方法实现了**Ctrl+点击**能实现方法的跳转后，就可以比较方便的对代码的理解了。

由于我们必须**count how many packets pass through a hub in and out.**所以就着眼于比较相似于接受**packet**和发送**packet**的方法；观察代码可知，**net**的**recv_packet(...)**和**send_packet(...)**的这两个方法都值得重视，跳转到方法的定义后，定义如下：

```python
class LLNetBase(metaclass=ABCMeta):
    '''
    Base class for the low-level networking library in Python.
    "net" objects are constructed from classes derived from this
    class.
    '''
    def __init__(self, name=None):
        self._devupdown_callback = None
        self._devinfo = {} # dict(str -> Interface)
    # .....
    @abstractmethod
    def recv_packet(self, timeout=None):
        '''
        Receive a packet on any port/interface.
        If a non-None timeout is given, the method will block for up
        to timeout seconds.  If no packet is available, the exception
        NoPackets will be raised.  If the Switchyard framework is being
        shut down, the Shutdown exception will be raised.
        If a packet is available, the ReceivedPacket named tuple
        (timestamp, input_port, packet) will be returned.
        '''
        raise NoPackets()

    @abstractmethod
    def send_packet(self, output_port, packet):
        '''
```

```
        Send a packet out the given output port/interface.
        Returns None.
        '''
        pass
```

由这些方法的注释可知：

1. **recv_packet方法就是Receive a packet on any port/interface**
2. **send_packet方法就是Send a packet out the given output port/interface.**

所以就可以分别在其后加上对应变量的自增，然后在最后输出**log**即可完成对应的任务，具体的代码改动如下：

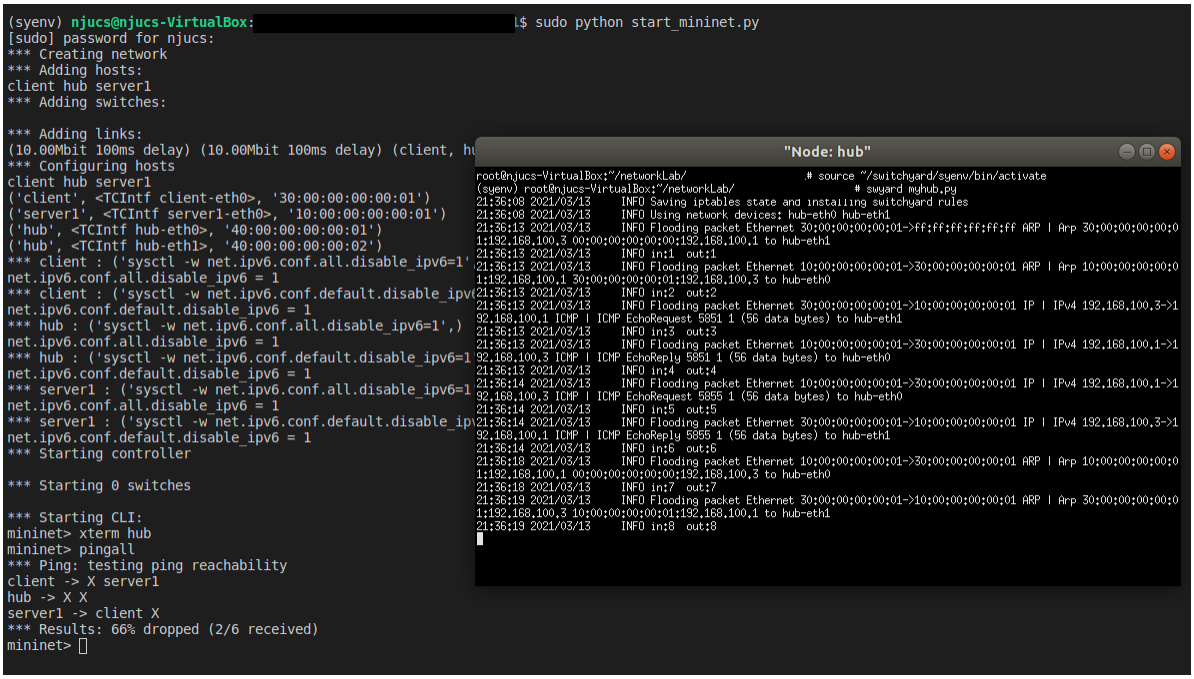
```
def main(net: switchyard.llnetbase.LLNetBase):
    my_interfaces = net.interfaces()
    mymacs = [intf.ethaddr for intf in my_interfaces]
    num_in, num_out = 0, 0
    while True:
        try:
            _, fromIface, packet = net.recv_packet()
            num_in = num_in + 1

        except NoPackets:
            continue
        except Shutdown:
            break

        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:
            log_info("Received a packet intended for me")
        else:
            for intf in my_interfaces:
                if fromIface!= intf.name:
                    log_info (f"Flooding packet {packet} to {intf.name}")
                    net.send_packet(intf, packet)
                    num_out = num_out + 1
        log_info (f"in:{num_in}  out:{num_out}")
    net.shutdown()
```

然后是对此结果进行一定的验证：

①首先搭建的网络打开

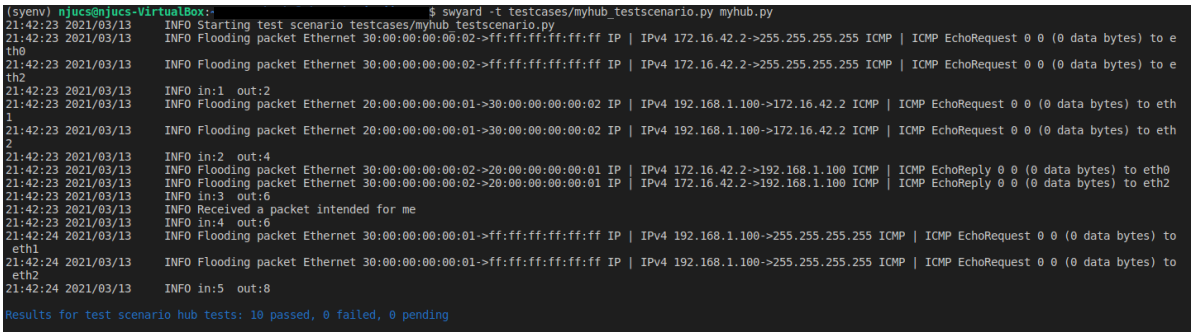
```
sudo python start_mininet.py
```

②输入***xterm hub***，然后在其中输入**swyard myhub.py**运行程序代码，得到的结果如下：



由右图可以看到能打印出了**log**，但是**in**和**out**是一样的，这是因为在**STEP 1**中我们已经将**server2**的这一个节点删除了，所以就有此现象的发生。

现接着采用**testcases/myhub_testscenario.py**对其进行测试：



由于在**testcases/myhub_testscenario.py**中有专门发送给**hub**的**packet**，这样**in**和**out**就有了一定的大小差距，也进一步说明了**log**的正确性。

STEP 3:Modify the test scenario of a device

这一STEP就是需要创造自己的**testcase**，我选择的是**Create one test case by using the given function** `new_packet` **with different arguments**，也就是通过传入不同的参数给写好的`new_packet`函数来创造一个新的**testcase**

首先先进行对于代码中给出的三个**testcase**进行一定的分析

1. **testcase 1:包从30:00:00:00:00:02 被广播之后, eth1 模拟包到达, 然后模拟包从其他两个端口发出.**
2. **testcase 2: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**
3. **testcase 3:因为这是一个发送给hub自己本身的包，所以应该无事发生**

以下就是我新创造的**testcase**，这是仿照**testcase1**所设计的。

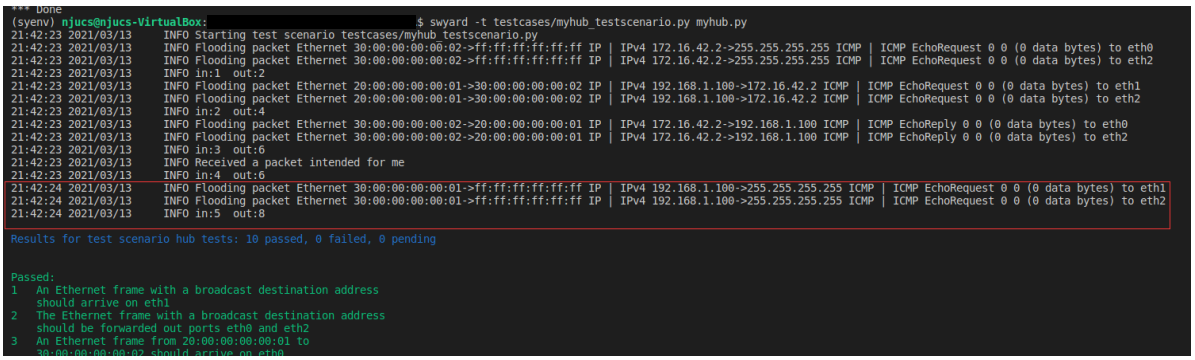
```
mytestpkt = new_packet(
    "30:00:00:00:00:01",
    "ff:ff:ff:ff:ff:ff",
    "192.168.1.100",
    "255.255.255.255"
```

```
)
s.expect(
    PacketInputEvent("eth0", mytestpkt, display=Ethernet),
    ("An Ethernet frame with a broadcast destination address "
     "should arrive on eth0")
)
s.expect(
    PacketOutputEvent("eth1", mytestpkt, "eth2", mytestpkt,
display=Ethernet),
    ("The Ethernet frame with a broadcast destination address should be "
     "forwarded out ports eth1 and eth2")
)
```

具体的含义就是：

包从30:00:00:00:00:01 被广播之后, eth0 模拟包到达, 模拟包从其他两个端口发出.

产生的结果如下面红色框内的内容：

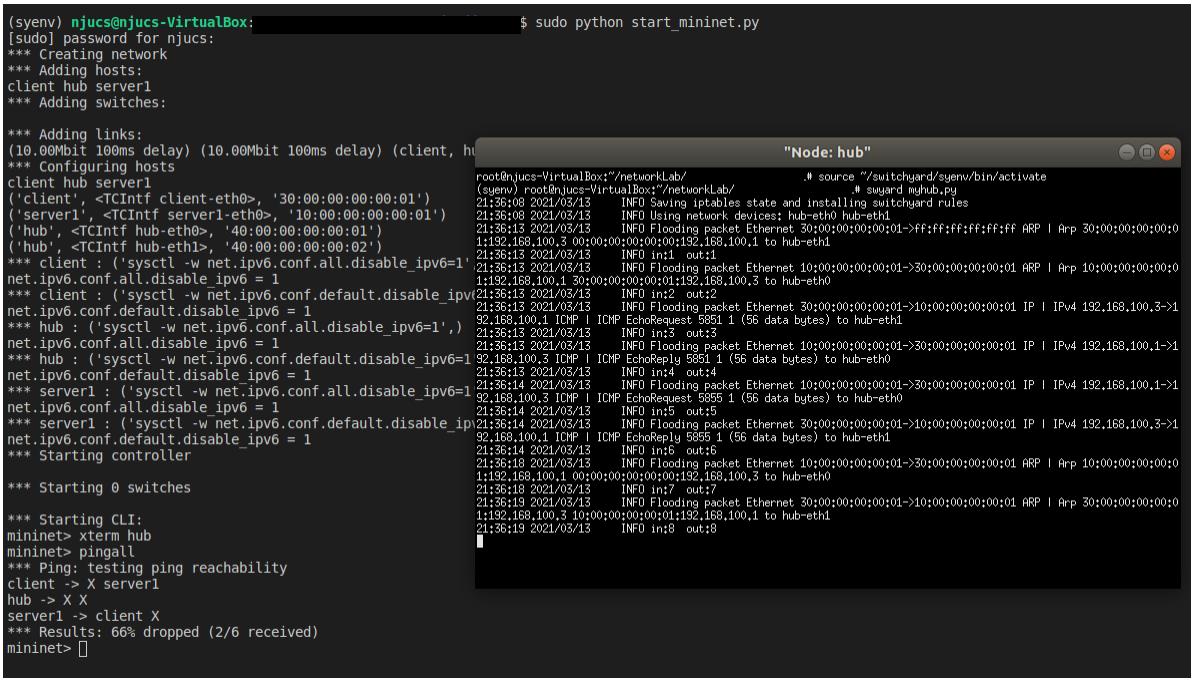


STEP 4:Run your device in Mininet

①首先搭建的网络打开：

```
sudo python start_mininet.py
```

②输入***xterm hub***，然后在其中输入**swyard myhub.py**运行程序代码，然后**CIL**中输入**pingall**得到的结果如下：



这样就完成了在**mininet**上面成功地**run my device**。

STEP 5:Capture using Wireshark

此STEP就可以开始实现抓取数据包了，根据手册中的 Wireshark 和 Switchyard，就可以得知该如何获取数据包，同样的在**CLI**输入对应指令

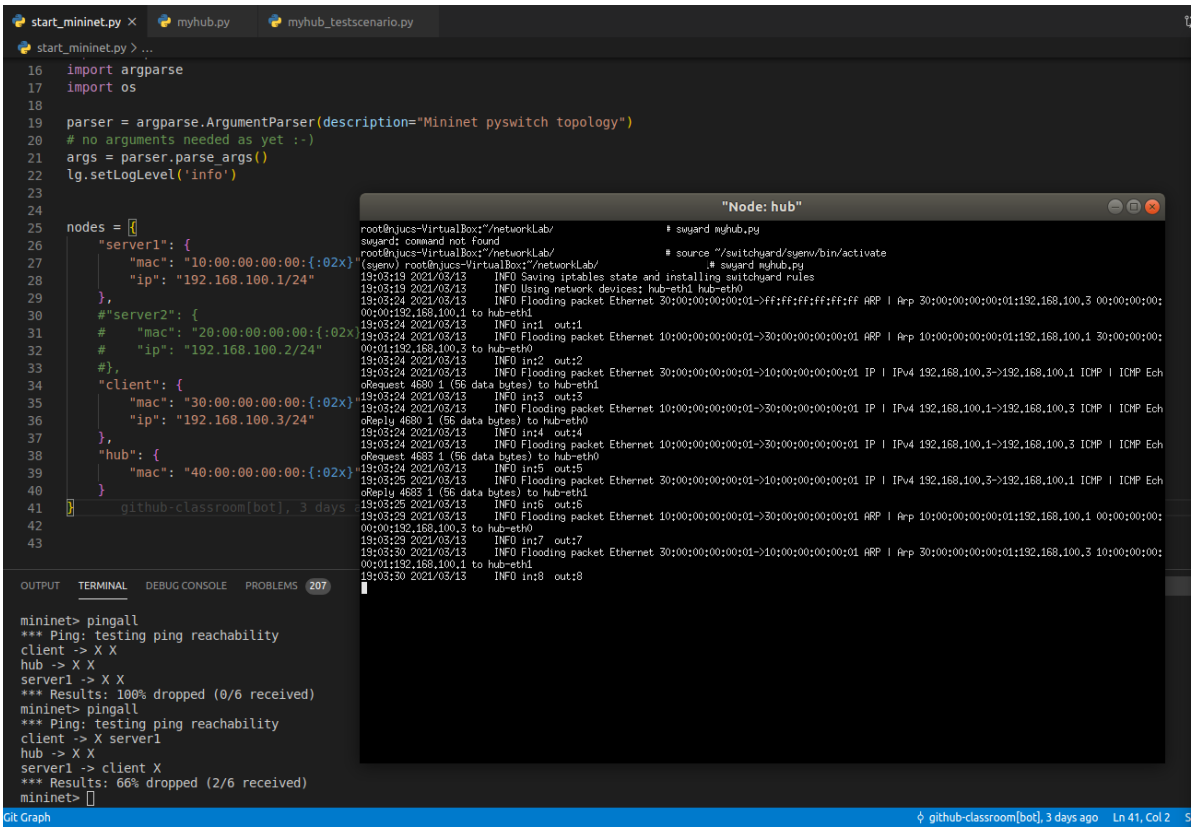
```
server1 wireshark &
```

来打开**wireshark**并选择对应的**server1-eth0**，进行数据包的抓取：

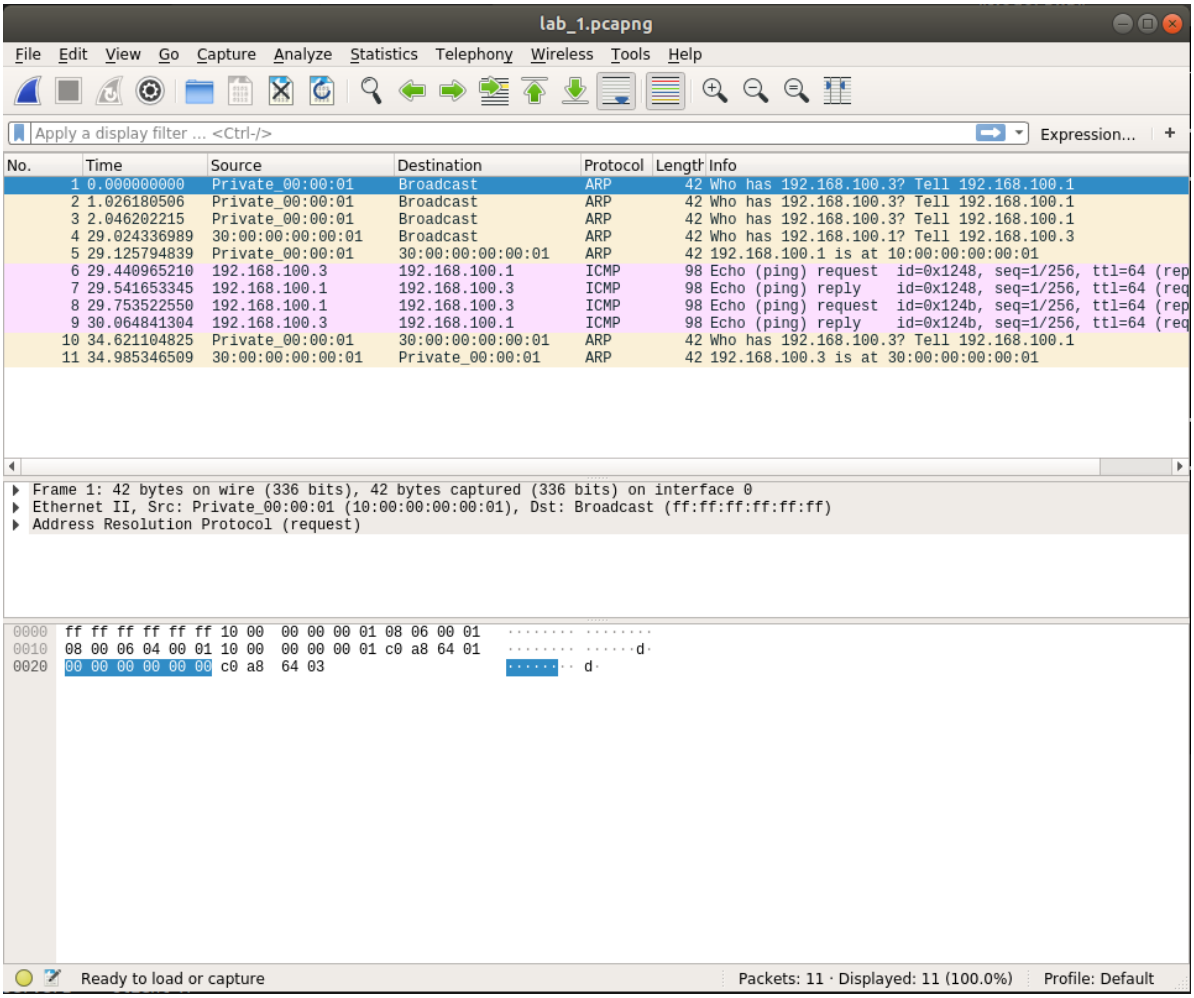
①首先在**mininet**中输入**pingall**，发现是**100% dropped**

②然后在**"Node :hub"**中输入**swyard myhub.py**

就能够成功地创造流量了。



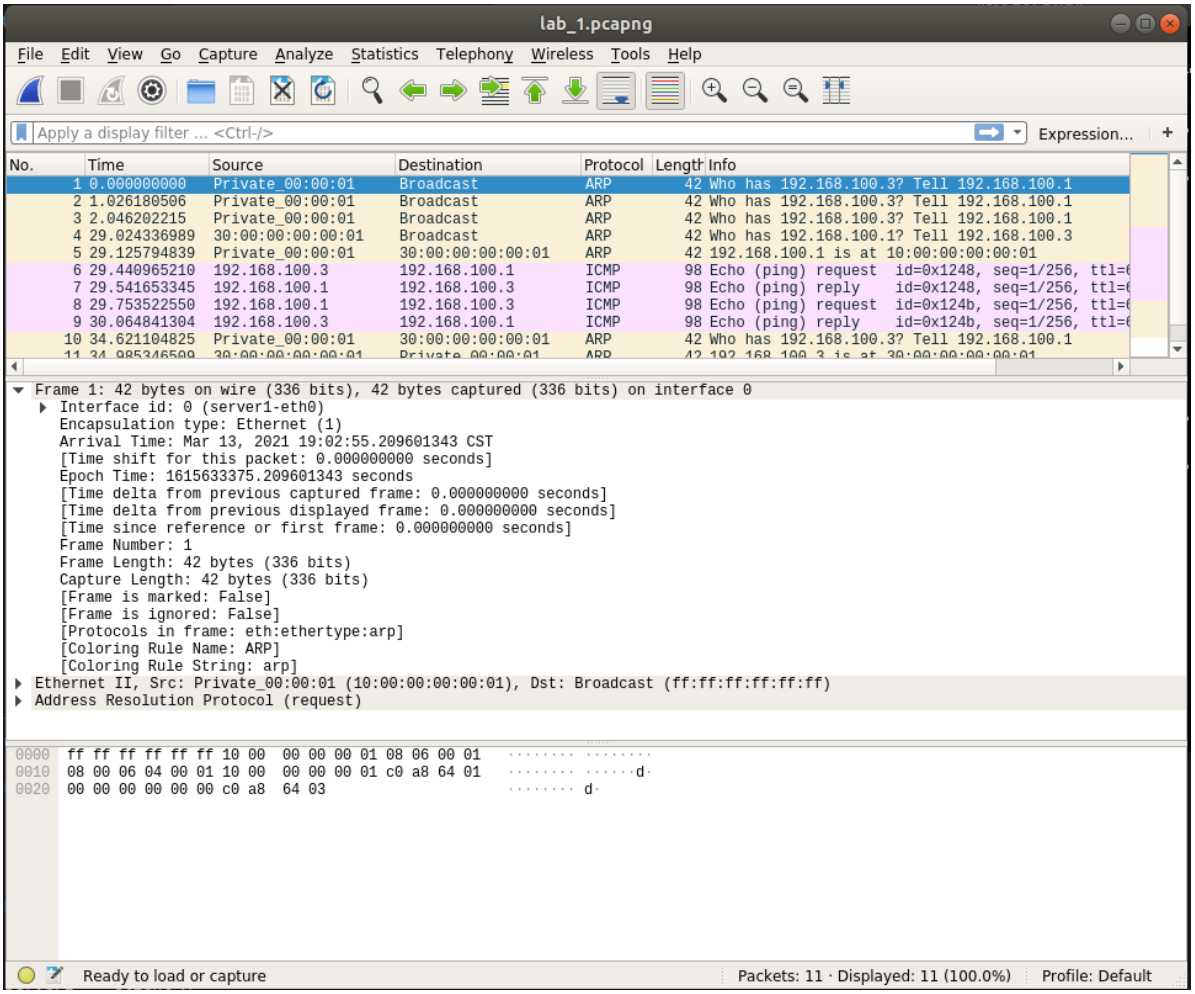
接下来是对于一个包为例子进行主要参数的简单分析：



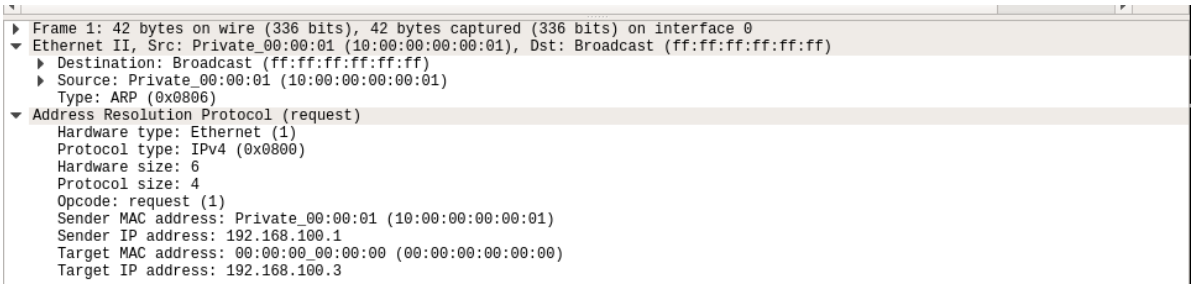
①第一部分：

帧**Frame 1** 指的是要发送的数据块，其中，所抓帧的序号为**0**，捕获字节数等于传送字节数也即**42bytes。**接下来还有许多关于这一部分的细节参数：

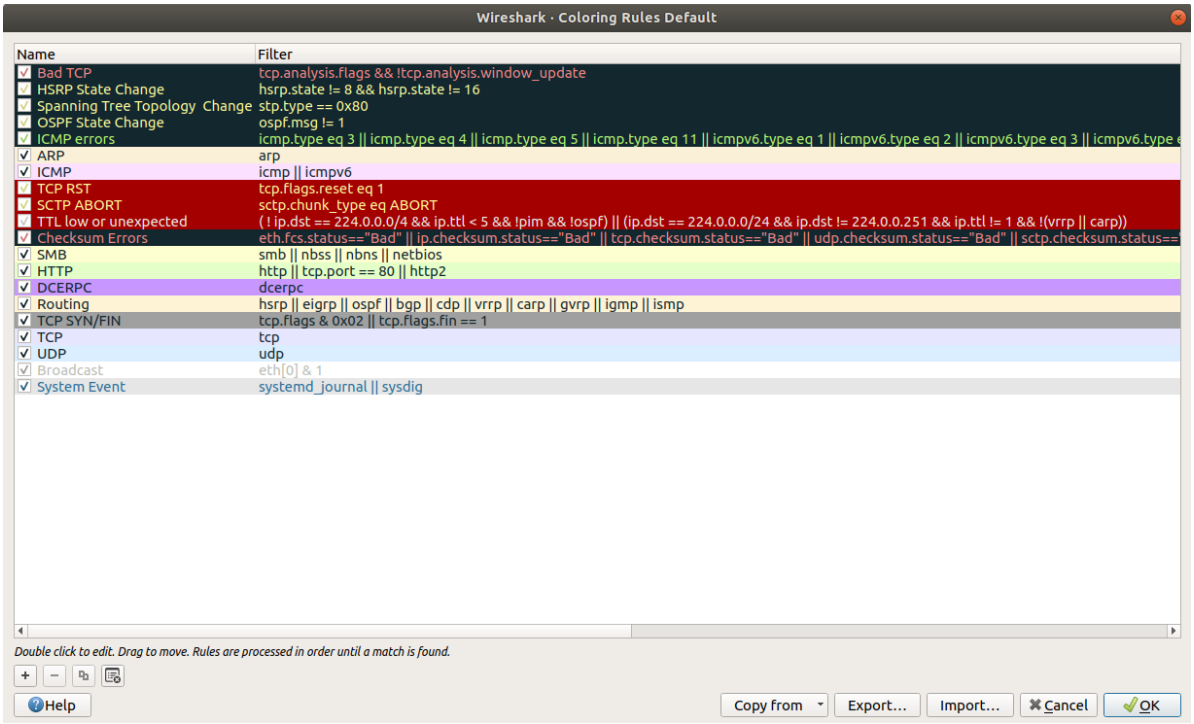
1. **Encapsulation type** ： Ethernet (1)，这代表的是一个**Wireshark-internal**值，表示所讨论的包的特定链接层报头类型.
2. **Arrival time**：表示的是数据包的**到达的时间**
3. **Epoch time**：表示的是接受的时间自**1970年1月1日**以来的秒数.
4. **Time delta**：由于这是第一个接受到的包，所以均为 **0 seconds**
5. **Frame**： 这里展示的是**Frame**的部分属性，包括**Frame Number** ， **Frame Length**, 协议等信息。



②第二部分：



（注：每种不同颜色的流量包含义如下：）

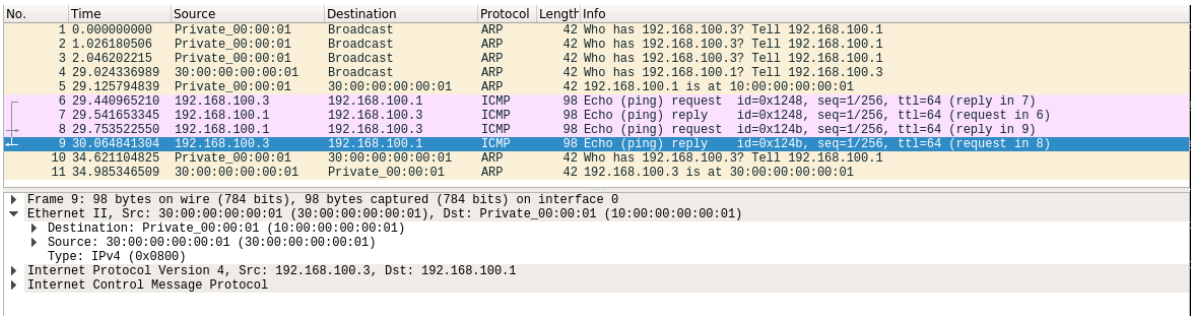


接下来来看看其他类型的数据包：

③以太网，有线局域网技术，是数据链路层。

源**Mac**地址为**30:00:00:00:00:01**；目标**Mac**地址为**10:00:00:00:00:01**；

type是**0x0800**表明了此数据包是**IP**数据包



④IPV4协议，也称网际协议，是网络层。

源IP地址为**192.168.100.3**；目标IP地址为**192.168.100.1**。

version4表明**IPV4**协议。

Time to live (TTL)为64表明包经过64路由器没到达会被丢弃。

Protocol：**ICMP**说明采用的协议是**ICMP**。

Headchecksum：头部校验和**0x608e**。

⑤ICMP协议

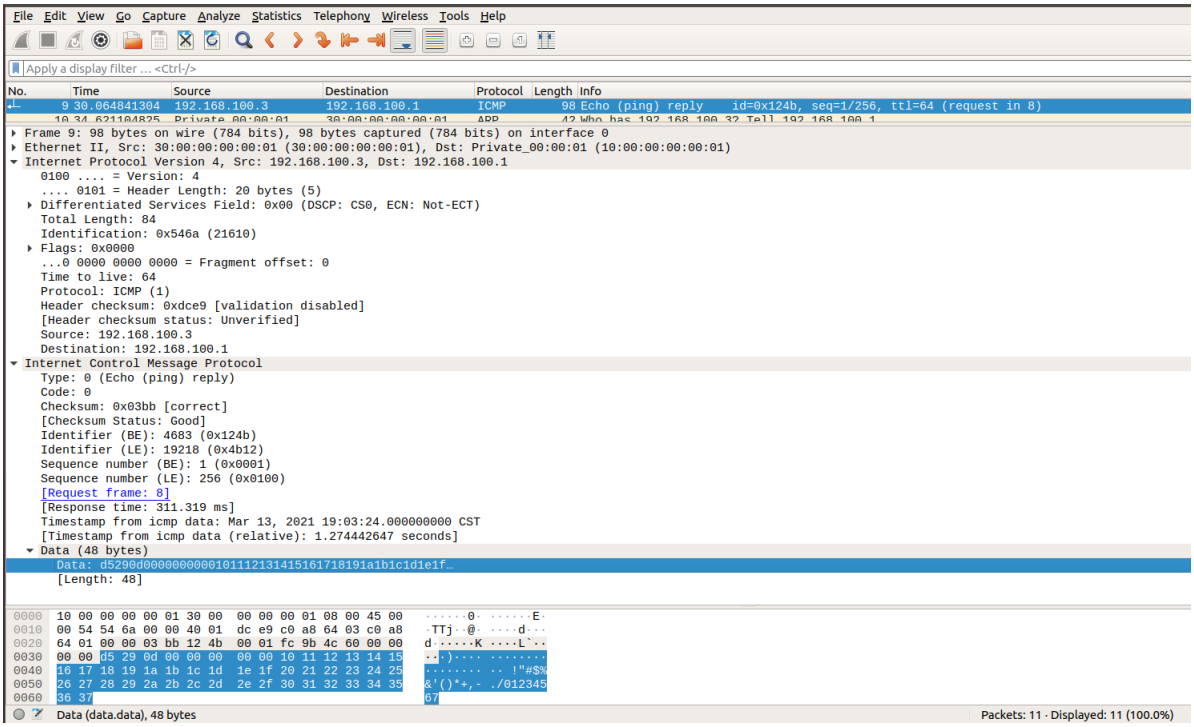
Type是**0**对应了**ping reply**；

checksum校验和的状态是**correct**正确的；

Identifier后面的括号里，**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，来更容易地检查丢失的序列。

以及最后的时间戳**Timestamp**。

在包的最后我们也可以看到包中所包含的**data**，足有**48 bytes**。



5.核心代码

具体的所需要改动的实验核心代码在上一部分**实验结果**中全部提及了，在此部分就不再过多赘述了。

6.总结与感想

总的来说，对于第一次的计算机网络实验，都是一些对于各种软件和api的熟练下进行一定的准备工作，包括但不限于对于**swyard**，**mininet**等工具的熟悉。但是还是有些不是很懂的东西，毕竟还有许多没有教学到的内容，后面我会一点一点将这些内容全部弄明白的。

感想：因为首次面对这种全英文的指导手册，而且各种帮助文档也都是英文的，所以读起来是非常费劲的其实。但是需要通过时间的积累来减少阅读英文的时间，况且以后也都是像类似于这样的英文的帮助文档的，所以需要长时间的锻炼的。