Computer Network lab6实验报告

姓名	学号
张洋彬	191220169
邮箱	完成日期
1016466918@qq.com	2021.5.27

1、实验名称

Lab 6: Reliable Communication

2、实验目的

1、在接收方实现ACK机制

2、在发送方实现滑动窗口机制、超时重传机制、ACK机制

3、在middlebox里面转发包

3、实验进行

3.1 Preparation

文件结构如下图所示：

github-classroom Setting up GitHub Classroom Feedback	b47bf97 yesterday	**3** commits
.github	GitHub Classroom Feedback	yesterday
testcases	Initial commit	yesterday
.gitignore	Initial commit	yesterday
README.md	Initial commit	yesterday
blastee.py	Initial commit	yesterday
blaster.py	Initial commit	yesterday
middlebox.py	Initial commit	yesterday
start_mininet.py	Initial commit	yesterday

3.2 Middlebox

1、根据传进来的参数初始化数据

```
def __init__(
            self,
            net: switchyard.llnetbase.LLNetBase,
            dropRate="0.19"
    ):
        self.net = net
        self.dropRate = float(dropRate)
```

2、处理收到来自连接blaster端口的包

首先生成一个0到1的随机数，如果大于丢包率则发包，如果小于丢包率则丢包。

发包的步骤如下：

- 修改包的源mac地址和目的mac地址
- 将包的ttl减1
- 从middlebox-eth1(与blastee相连的端口)将包发送出去

如遇到丢包：打印丢包的随机数和序列号。对arp包不处理，因为本实验中的ip和mac地址是绑定的，不存在需要询问mac地址的情况 。

```
  ipv4=packet.get_header(IPv4)
  rand=random.random()
  if ipv4 is not None:
    if rand>self.dropRate:
      packet[Ethernet].src=  '40:00:00:00:00:02'
      packet[Ethernet].dst = '20:00:00:00:00:01'
      packet[IPv4].ttl -= 1
      # send to blastee
      self.net.send_packet("middlebox-eth1", packet)
    else:
      seq = int.from_bytes(packet[RawPacketContents].to_bytes()[:4],'big')
      print("rdn = {}, seq = {}".format(rand,seq))
```

3、处理收到来自连接blastee端口的包

大致和第二步类似，只是不用判断是否需要丢包

```
ipv4=packet.get_header(IPv4)
 if ipv4 is not None:
   packet[Ethernet].src=  '40:00:00:00:00:01'
   packet[Ethernet].dst = '10:00:00:00:00:01'
   packet[IPv4].ttl -= 1
   # send to blaster
   self.net.send_packet("middlebox-eth0", packet)
```

3.3 Blastee

1、首先处理传进来的参数

```
def __init__(
            self,
            net: switchyard.llnetbase.LLNetBase,
            blasterIp,
            num
    ):
    self.net = net
    self.blasterIp=IPv4Address(blasterIp)
    self.num=int(num)
```

2、然后产生一个ACK，为其添加 Ethernet 、 IPv4 、 UDP 头，并设置源MAC地址为 blastee 的MAC地址，目的MAC地址为 middlebox-eth1 的MAC地址，源IP地址为 blastee 的IP地址，目的IP地址为 blaster 的IP地址，ttl字段为64

```
  ack = Ethernet() + IPv4(protocol = IPProtocol.UDP) + UDP()
  ack[Ethernet].src = '20:00:00:00:00:01'
  ack[Ethernet].dst = '40:00:00:00:00:02'
  ack[IPv4].src = '192.168.200.1'
  ack[IPv4].dst = self.blaster_IP
  ack[IPv4].ttl = 64
```

3、 然后从发来的 pkt 中提取序列号 seq 和负载 payload 。根据规约 seq 为 RawPacketContents 字段的前4个字节， payload 在 RawPacketContents 字段的第7个字节及以后，为了方便调试，这里还利用 from_bytes 方法将大端编码的 seq 转换为数字打印了出来：

```
  seq = packet[RawPacketContents].to_bytes()[:4]
  payload = packet[RawPacketContents].to_bytes()[6:]

  print("recv seq: {}".format(int.from_bytes(seq,'big')))
```

4、根据约定，ACK的 payload 字段有8个字节的固定大小，故这里还需要对其进行修整，截取其前8个字节或者进行补齐：

```
# limit the length of payload
    if len(payload) > 8:
      payload = payload[:8]
    elif len(payload) < 8:
      info = int.from_bytes(payload, 'big')
```

5、最后将 seq 和 payload 添加到ACK中并进行发送：

```
  ack.add_header(RawPacketContents(seq))
  ack.add_header(RawPacketContents(payload))
  myintf=None
  for intf in self.net.interfaces():
    if intf.ipaddr=='192.168.200.1':
      myintf=intf
      break
    # send ACK
  self.net.send_packet(myintf, ack)
```

3.4 Blaster

1、首先处理传进来的参数

```
def __init__(
            self,
            net: switchyard.llnetbase.LLNetBase,
            blasteeIp,
```

```
            num,
            length="100",
            senderWindow="5",
            timeout="300",
            recvTimeout="100"
    ):
        self.net = net
        # TODO: store the parameters
        ...
        self.blasteeIp=str(blasteeIp)
        self.length=int(length)
        self.senderWindow=int(senderWindow)
        self.timeout=int(timeout)
        self.recvTimeout=int(recvTimeout)
```

2、设置一些常量，用于控制超时机制，窗口：

```
    self.LHS=1#left
    self.RHS=0#right
    self.window=[]#窗口队列
    self.starttime=datetime.now()#开始时间 seq:datetime.now()
    self.reTX_num=0#重传的数目
    self.suc_num=0#成功传送的数目
    self.timeout_num=0#超时的包的数目
    self.total_num=0#目前发包的所有数目
    self.total_time=0#时间
    self.ack_queue=[]#成功收到ACK的队列
    self.nonack_queue=[]#没有收到ACK的队列
    self.out_time=datetime.now()# 计时器
```

3、然后修改处理包的函数，如果收到一个ACK包，提取出他的序列号seq，然后将它从window和nonack_queue之中移除（需判断一下，防止冗余ACK），然后suc_num+1;重置计时器，然后打印出来我们需要的信息，最后是关于LHS的修改，如果此时所有包都得到确认，就取ack_queue里的最大序列号+1，否则就取nonack_queue里的最小序列号。

```
        ipv4=packet.get_header(IPv4)
        if ipv4 is not None:
            seq = int.from_bytes(packet[RawPacketContents].to_bytes()[:4], 'big')
            if seq in self.window:
                self.window.remove(seq)
            if seq in self.nonack_queue:
                self.nonack_queue.remove(seq)
                self.ack_queue.append(seq)
                self.total_time=datetime.now().timestamp()-
self.starttime.timestamp()
                self.suc_num += 1
                self.timeout=datetime.now()
                self.reTX_num=self.total_num-self.suc_num
```

```
                print("Total TX time is {} seconds.".format(self.total_time))
                print("Number of reTX is {}.".format(self.reTX_num))
                # print Number of coarse TOs
                print("Number of coarse TOs is {}.".format(self.timeout_num))
                # print Throughput(Bps)
                Throughput = self.length * self.total_num / self.total_time
                print("Throughput is {} Bps.".format(Throughput))
                # print Goodput(Bps)
                Goodput = self.length * self.suc_num / self.total_time
                print("Goodput is {} Bps.".format(Goodput))

            if self.nonack_queue == []:
                LHS = max(self.ack_queue) + 1
            else:
                LHS = min(self.nonack_queue)

```

4、如果没有收到ACK包的话，则需要创建一个包并发送给blastee,此时需要注意的问题有：

- 如果超时，则需要重置out_time，然后将未收到ACK的包重新加入window中
- 如果此时window还没满且RHS未到达尾部，则将RHS右移一位（+1），然后将其加入window和nonack_queue
- 移除window里的seq，并用seq来构建包发送给blastee，此时全部发包数+1

```
# Creating the headers for the packet
        pkt = Ethernet() + IPv4() + UDP()
        pkt[1].protocol = IPProtocol.UDP
        pkt[Ethernet].src = '10:00:00:00:00:01'
        pkt[Ethernet].dst = '40:00:00:00:00:01'
        pkt[IPv4].src = '192.168.100.1'
        pkt[IPv4].dst = IPv4Address(self.blastee_IP)
        pkt[IPv4].ttl = 64
   # if timeout, retransmit nacked packet and reset out_time
        if datetime.now().timestamp-self.timeout.timestamp > self.timeout / 1000:
            self.timeout_num +=1
            self.out_time = datetime.now()
            self.window=self.nonack_queue.copy()


        # send packet if C1 is met
        if self.RHS - self.LHS + 1 < self.sender_window and self.RHS < self.num:
            self.RHS += 1
            self.window.append(self.RHS)
            self.nonack_queue.append(self.RHS)

        if self.window != []:
            seq = self.window.pop(0)
            pkt.add_header(RawPacketContents(seq.to_bytes(4,'big')))
            pkt.add_header(RawPacketContents(self.length.to_bytes(2,'big')))
```

```
26
pkt.add_header(RawPacketContents(int(123456789).to_bytes(self.length,'big')))
27              self.total_num += 1
28              myintf=None
29              for intf in self.net.interfaces():
30                  if intf.ipaddr == '192.168.100.1':
31                      myintf=intf
32                      break
33              self.net.send_packet(myintf,pkt)
```

测试：

1、输入`sudo python start_mininet.py`进入mininet中

2、xterm打开各个node

`mininet> xterm middlebox`

`mininet> xterm blastee`

`mininet> xterm blaster`

3、在xterm传入参数，查看过程：

`middlebox# swyard middlebox.py -g 'dropRate=0.19'`
`blastee# swyard blastee.py -g 'blasterIp=192.168.100.1 num=100'`
`blaster# swyard blaster.py -g 'blasteeIp=192.168.200.1 num=100 length=100 senderWindow=5 timeout=300 recvTimeout=100'`

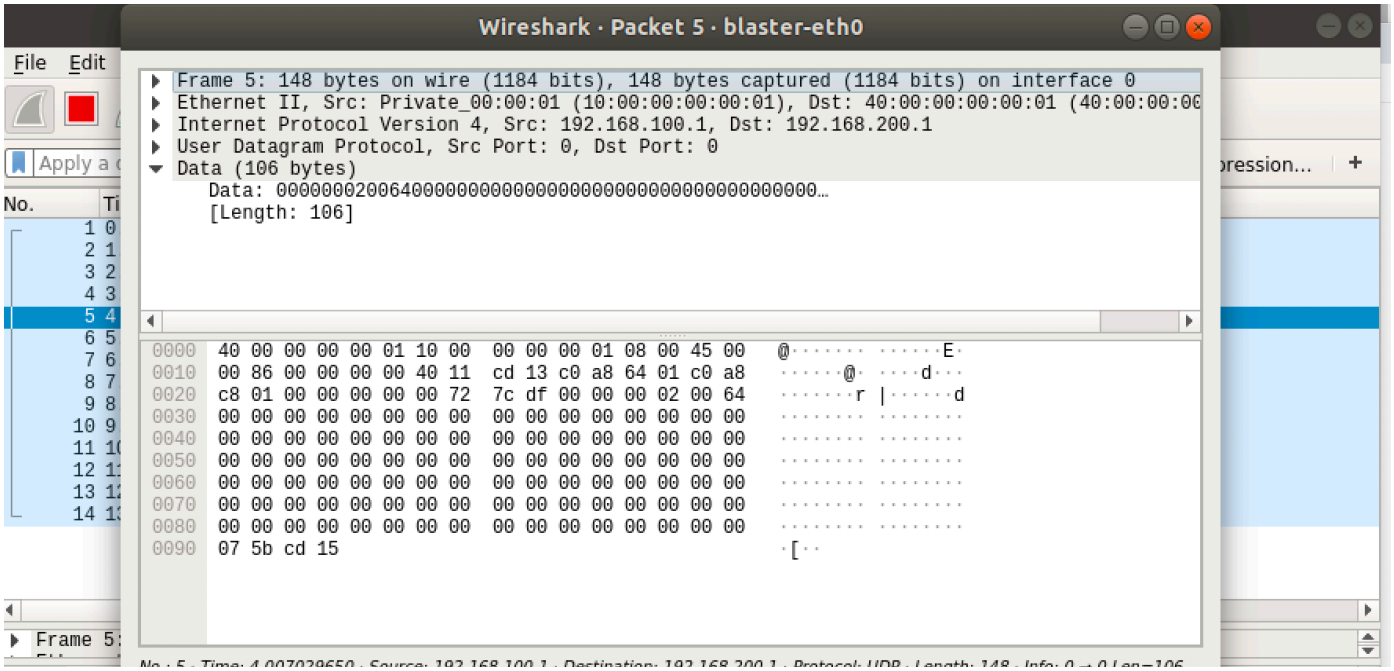
1、首先测试丢包率为1的特殊情况：

可见所有随机数都小于1，序列号一直停留在1-5（这些包都被丢了，没有得到回复），所以blaster一直在重复发1-5，超时的时间设置为4s

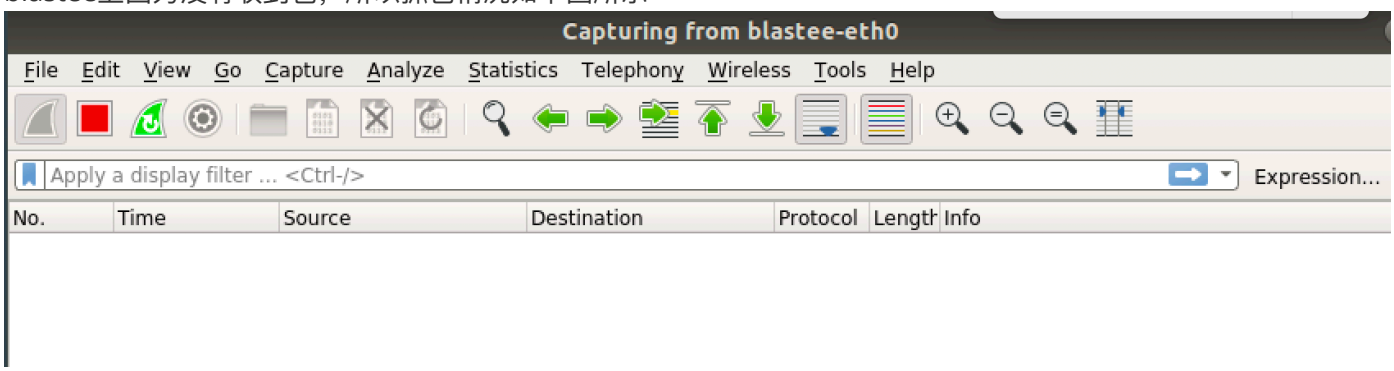
window的更新情况以及发出的包的序号如下图所示：

```
(syenv) root@njucs-VirtualBox:~# swyard lab-6-bbzunyi/blaster.py -g 'blasteeIp=192.168.200.1 num=100 length=100 senderWindow=5 timeout=4000 recvT
imeout=100'
16:05:33 2021/05/27     INFO Saving iptables state and installing switchyard rules
16:05:33 2021/05/27     INFO Using network devices: blaster-eth0
[1]
1
[2]
2
[3]
3
[1, 2, 3, 4]
1
[2, 3, 4, 5]
2
[3, 4, 5]
3
[4, 5]
4
[1, 2, 3, 4, 5]
1
[2, 3, 4, 5]
2
[3, 4, 5]
3
[4, 5]
4
[1, 2, 3, 4, 5]
1
[2, 3, 4, 5]
2
[3, 4, 5]
3
^C16:05:48 2021/05/27     INFO Restoring saved iptables state
```

blaster上端口的抓包情况如下图所示：

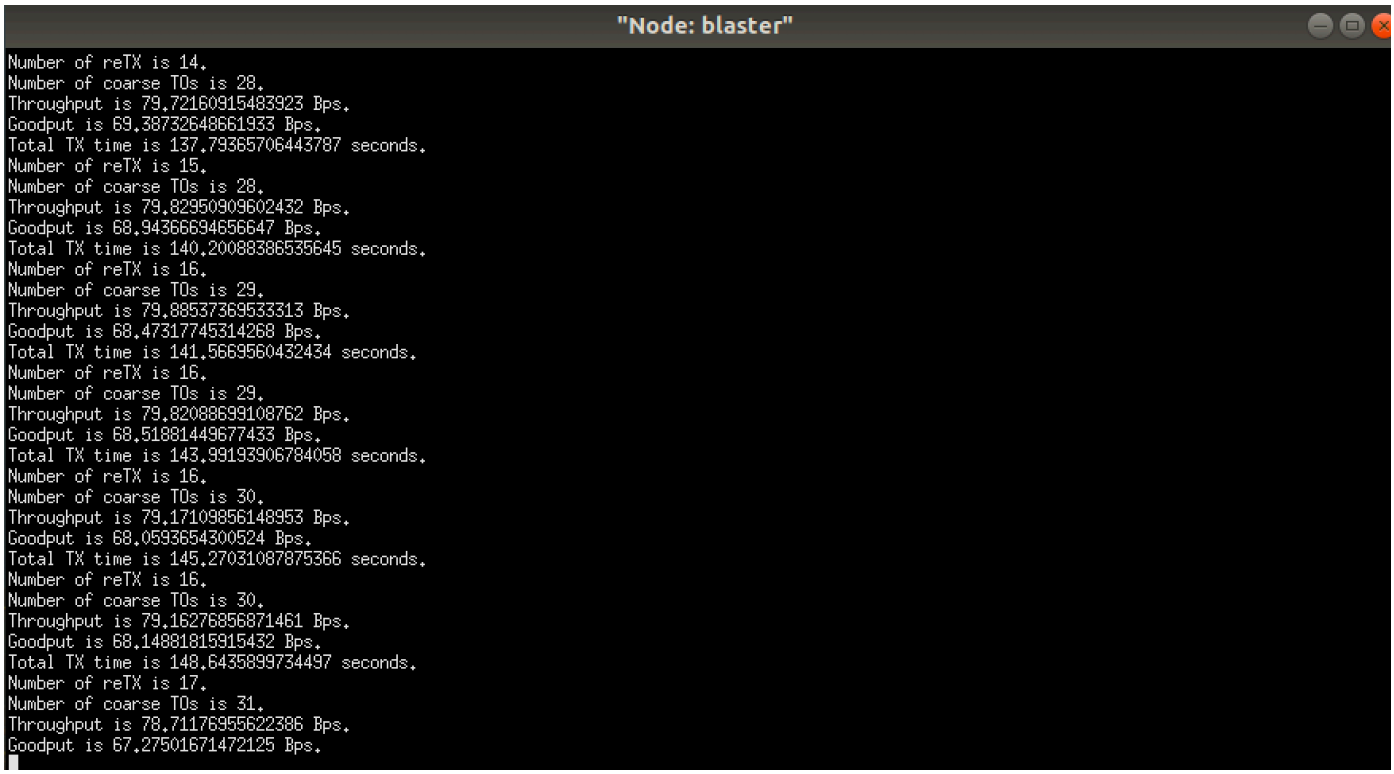


blastee上因为没有收到包，所以抓包情况如下图所示：

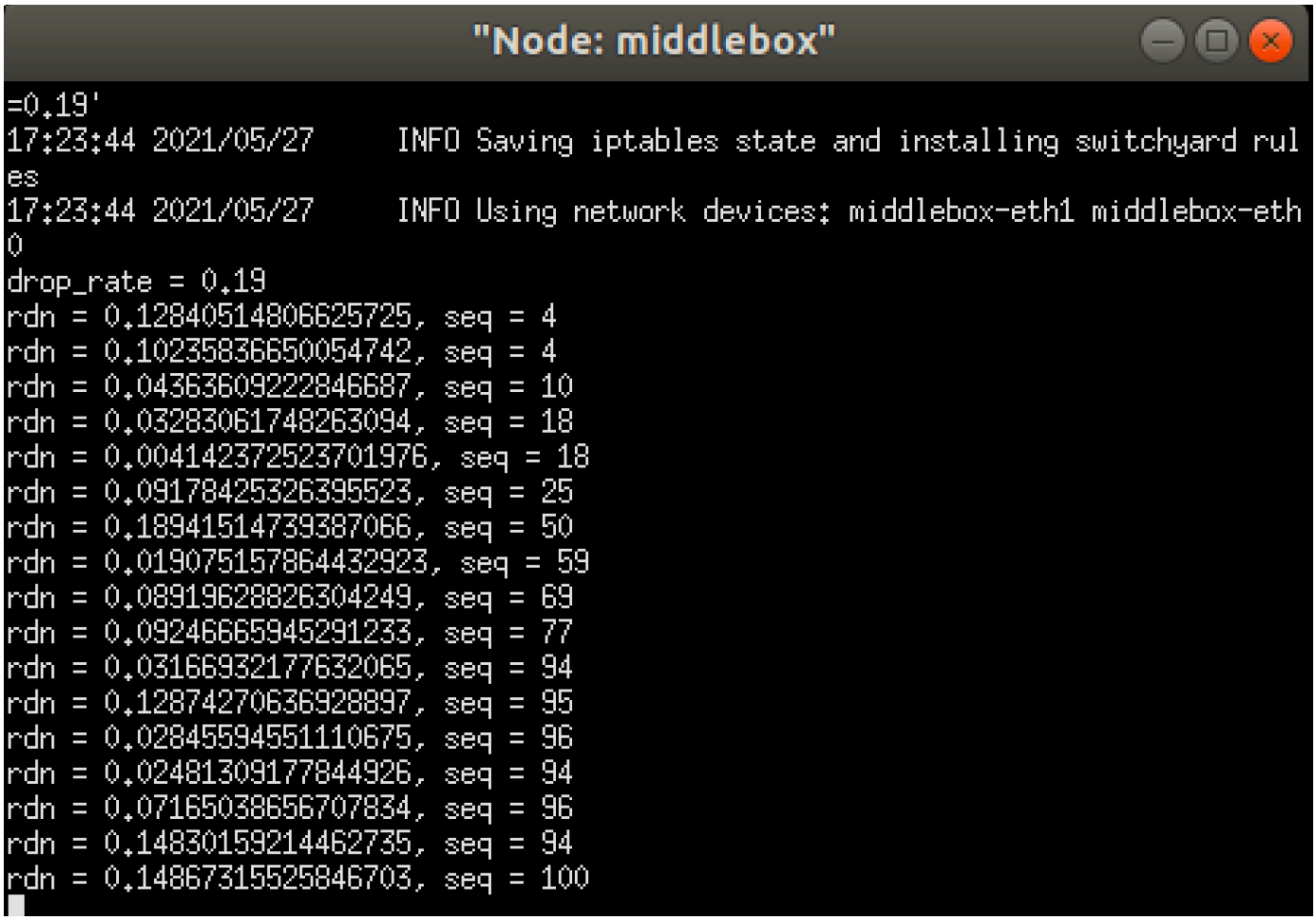


2、使用默认的参数进行测试：(timeout时间太短会顺序发包，所以这里还是把他调成4s)：

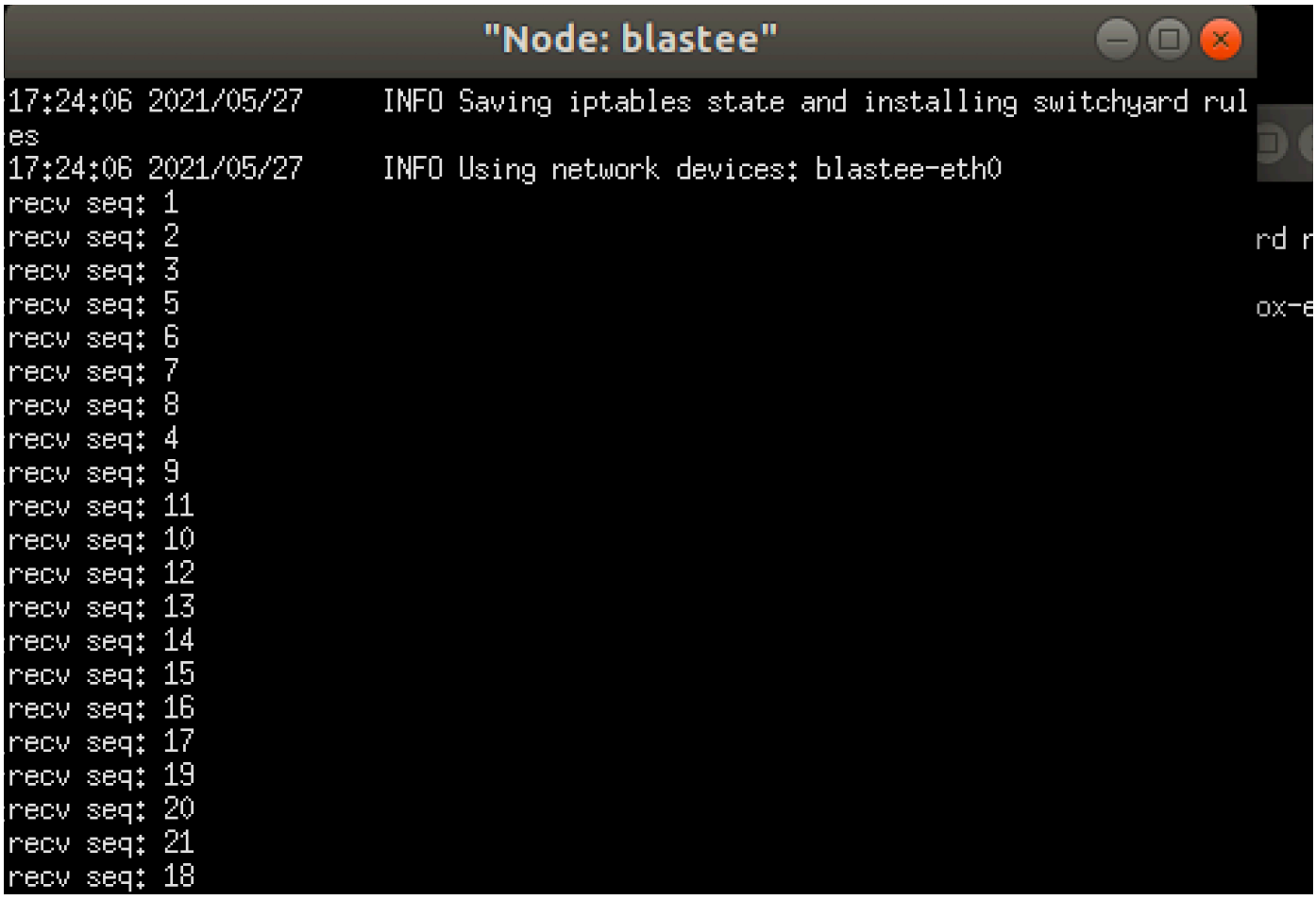
blaster的情况如下：



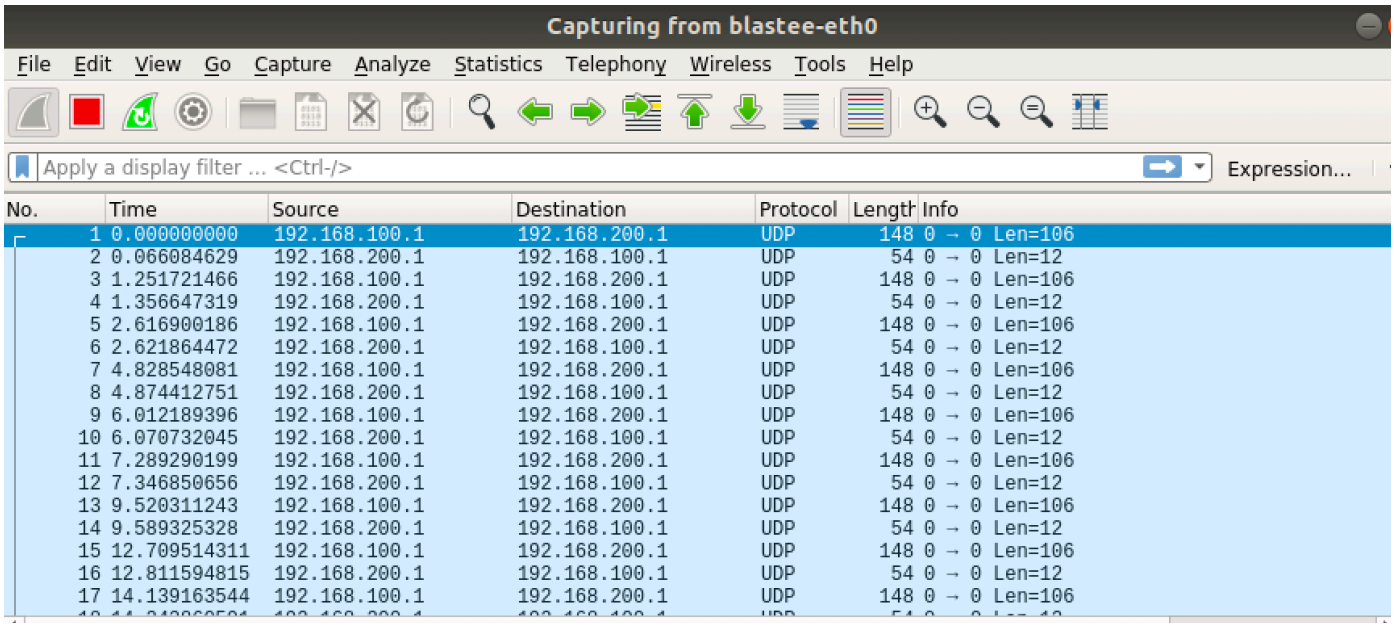
丢包的序号：

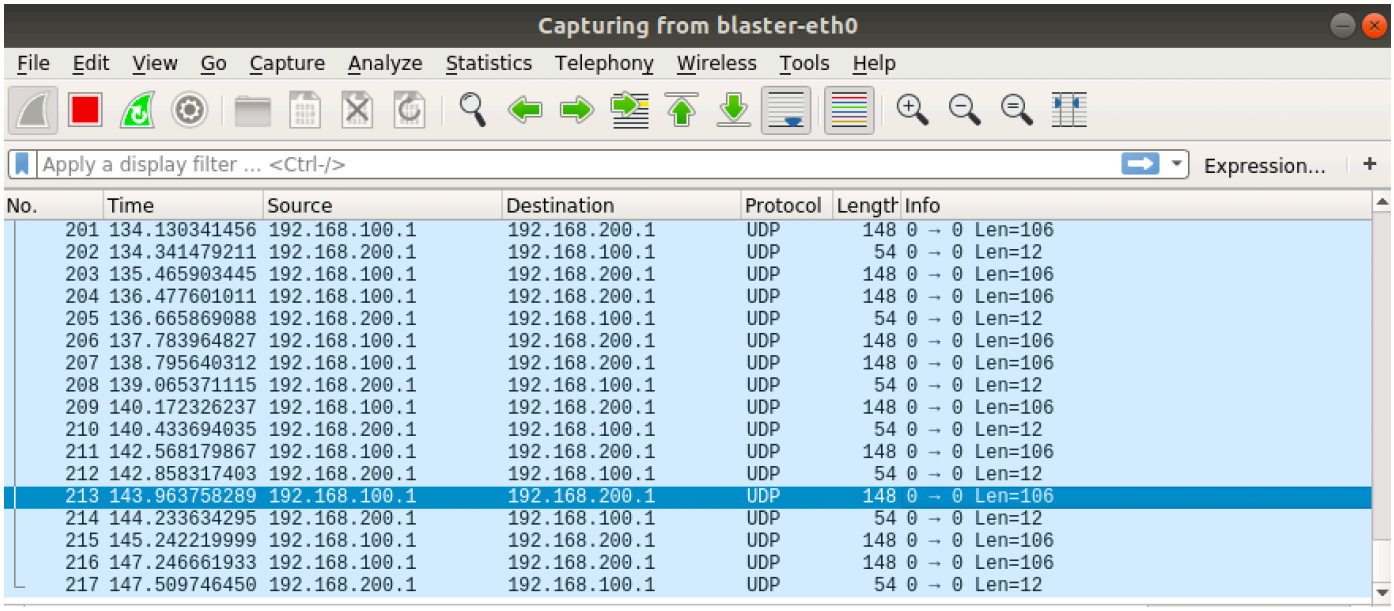


blastee的收包情况，可见4发生了丢包，等到4收到回复后才发9



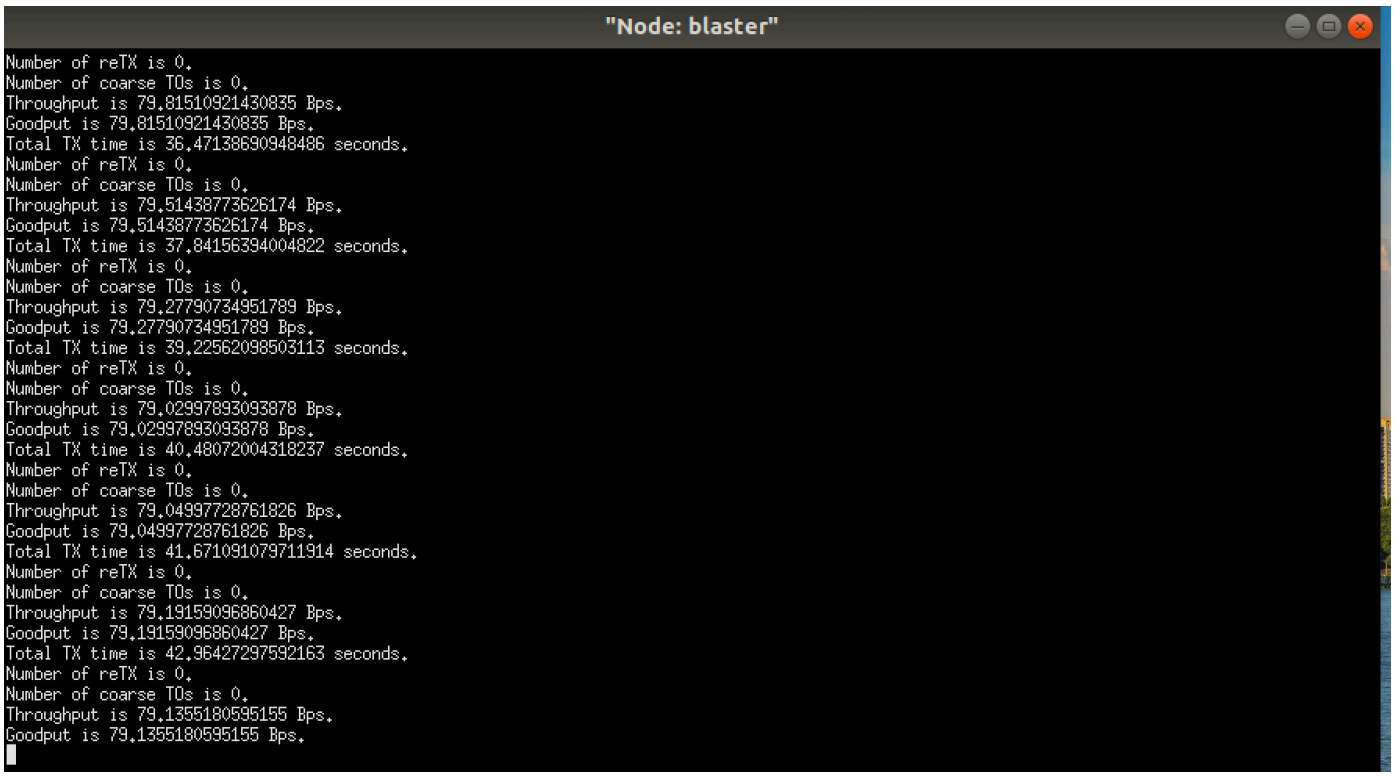
blaster和blastee上的抓包图如下：



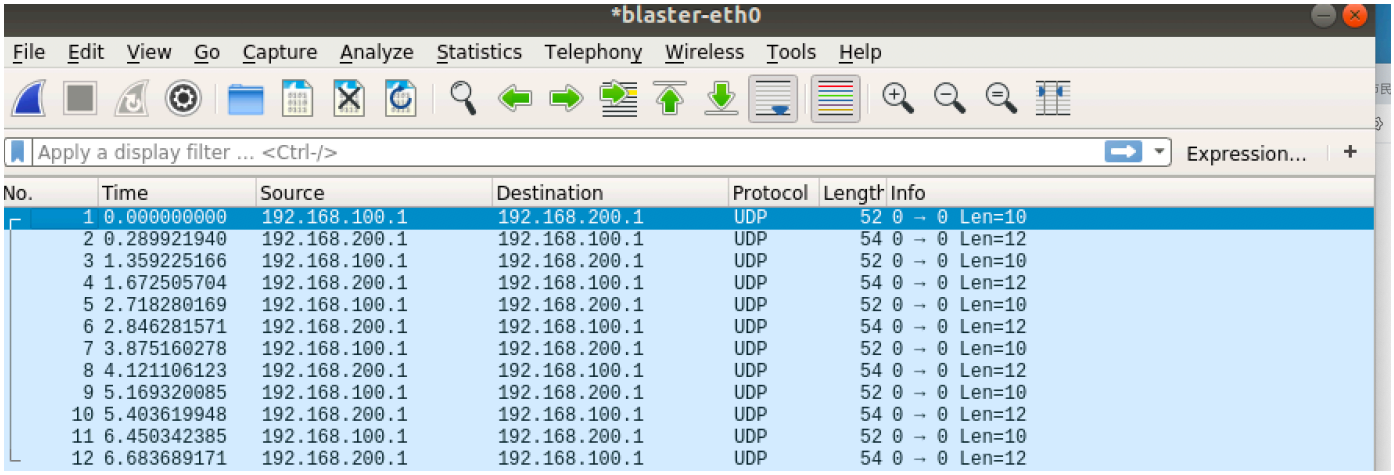


3、dropRate=0的情况测试

goodput=Throughput，没有发生丢包



4、将payload的值设为4个子节，检查payload的长度变化，可见从 blaster 发往 blastee 的包的 Len = 10 ，满足我们的假设(4个字节的 sequence number 加两个字节的 length ， 加4个字节的可变长payload)，而从 blastee 发往 blaster 的包的 Len 仍为12，这是因为我们约定了ACK的 payload 有定长8个字节，再加上4个字节sequence 字段，总计12字节



4、实验感想

本次实验难度还是有的，测试的方式也和之前不一样了，只是感觉报错没有之前那么清楚。在设计怎么测试的过程中，加多了一些参数的显示，在整个过程中体会到了实验的乐趣，这次实验给自己的发挥空间比较大，不像之前那样“保姆教学”了，虽然说有点累，但是在里面也学到了很多知识，加深了对滑动窗口（简易实现）、超时重传、ACK机制的了解，对运输层也有了部分的理解，总的来说，还是收获满满的。