Lab 6: Reliable Communication

181180050 孔孟荀

Task 1: Preparation

略

Task 2: Middlebox

因为有一个drop的比率，考虑每一次从blaster收到包要发往blastee的时候获得一个0，1之间的随机浮点数与droprate比较，如果大于才发送。
除此之外还要修改包的header，主要根据start_mininet.py里的地址进行硬编码

```
def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
    _, fromIface, packet = recv
    if fromIface == self.intf1:
        log_debug("Received from blaster")
        '''
        Received data packet
        Should I drop it?
        If not, modify headers & send to blastee
        '''
        ra = random.random()#gennerate a float between 0 and 1
        if ra<=self.dropRate:
            return
        else:
            e = packet.get_header(Ethernet)
            e.src = '40:00:00:00:00:{:02x}'
            e.dst = '20:00:00:00:00:{:02x}'
            ipv4 = packet.get_header(IPv4)
            ipv4.src = '192.168.200.2/30'
            ipv4.dst = '192.168.200.1/30'
            packet[Ethernet]=e
            packet[IPv4]=ipv4
            self.net.send_packet("middlebox-eth1", packet)
    elif fromIface == "middlebox-eth1":
        log_debug("Received from blastee")
        '''
        Received ACK
        Modify headers & send to blaster. Not dropping ACK packets!
        net.send_packet("middlebox-eth0", pkt)
        '''
        e = packet.get_header(Ethernet)
        e.src = '40:00:00:00:00:{:01x}'
        e.dst = '10:00:00:00:00:{:02x}'
        ipv4 = packet.get_header(IPv4)
        ipv4.src = '192.168.100.2/30'
        ipv4.dst = '192.168.100.1/30'
        packet[Ethernet]=e
        packet[IPv4]=ipv4
        self.net.send_packet("middlebox-eth1", packet)
    else:
        log_debug("Oops :))")
```

Task 3: Blastee

对于序列号，发出的ack与原来包的序列号相同，只要
用`packet.get_header(RawPacketContents)`取得raw bytes，再把他看作一个列表取前四个字节，即可。
payload则来自于

> You will populate these bytes from the first 8 bytes of the variable length payload of the blaster's packet

所以只要把raw bytes从第六个字节开始的部分取出来，再看看长度够不够8字节，如果不够再自己pad就好

```
def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
    _, fromIface, packet = recv
    log_debug(f"I got a packet from {fromIface}")
    log_debug(f"Pkt: {packet}")
    raw = packet.get_header(RawPacketContents)
    seq = int.from_bytes(raw.data[:4],'big').to_bytes(4,'big')
    payload = raw.data[6:]
    if len(payload)<8:
        payload += "\0".encode()*(8-len(payload))#padding
    payload = payload[0:8]
    e=Ethernet()
    e.src = '20:00:00:00:00:01'
    e.dst = '40:00:00:00:00:02'
    ip =IPv4(protocol=IPProtocol.UDP)
    ip.src = '192.168.200.1'
    ip.dst = '192.168.200.2'
    udp = UDP()
    sepkt = e + ip + udp + seq + payload
    itf = self.net.interface_by_name('blastee-eth0')
    self.net.send_packet(itf,sepkt)
```

Task 4: Blaster

由于最后要输出很多东西，我为blaster类定义了很多属性：

```
self.net = net
self.blasteeip = blasteeIp
self.num = num
self.length = length
self.sw = senderWindow
self.timeout = float(timeout)/1000
self.recvtimeout = float(recvTimeout)/1000
self.lhs= 0
self.rhs= 0
self.notacklist = []
self.resendtime = 0
self.resendpos=0
self.firstsend = -1
self.lastacked = -1
self.retxnum = 0
self.coarsetonum = 0
self.throughputnum = 0
self.goodputnum = 0
```

由faq知道

> You should be sending a single packet per each recv_timeout loop

1. 在blaster中，每个循环判断有没有收到ack包，如果没有，那就进入

```
def handle_no_packet(self):
    log_debug("Didn't receive anything")

    if len(self.notacklist)!=0 and int(self.rhs -self.lhs +1) == int(self.sw) : #resend

        if time.time()-self.resendtime>float(self.timeout) and self.resendpos<len(self.notacklist):
            pkt = self.make_packet(self.notacklist[self.resendpos])
            itf = self.net.interface_by_name('blaster-eth0')
            self.net.send_packet(itf,pkt)
            self.retxnum +=1
            self.resendpos+=1
            self.throughputnum +=int(self.length)

            if self.resendpos == len(self.notacklist):
                self.resendtime = time.time()
                self.resendpos = 0
                self.coarsetonum+=1


    elif self.rhs-self.lhs+1<int(self.sw):
        log_info("2 :self.rhs={},self.lhs={}\n".format(self.rhs,self.lhs))
        if self.rhs == int(self.num)-1:
            return

        if self.rhs==0:
            self.firstsend=time.time()
        self.rhs +=1
        pkt = self.make_packet(self.rhs)
        itf = self.net.interface_by_name('blaster-eth0')
        self.net.send_packet(itf,pkt)
        self.notacklist.append(self.rhs)
        self.throughputnum +=int(self.length)
        self.goodputnum +=int(self.length)
```

这个函数中先判断目前应该是发包还是发重传包，**首先判断要不要重传**

- 如果说定义的目前需要重传的列表self.notacklist长度不为0，而且rhs-lhs+1==senderwindow长度，说明目前**需要重传**，因为一次发一个包，所以定义了一个resendpos的属性，每次coarse timeout引发的重传全部完成后，他被置为0，与此同时重传判定的初始值resendtime被置为当前时间。
  每重传一个包，resendpos加一，当他的值等于self.notacklist的长度，说明已经全部重传完成。
- 如果说目前rhs-lhs+1< senderwindow，说明**不需要重传**，需要正常发包，发包后增

加rhs即可。

2. 而如果收到了ack包，那就进入

```
def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
    _, fromIface, packet = recv
    log_debug("I got a packet")
    raw = packet.get_header(RawPacketContents)
    seq = int.from_bytes(raw.data[:4],'big')
    if seq == int(self.num)-1:
        self.lastacked = time.time()
    if seq == self.notacklist[0]:
        if len(self.notacklist)==1:
            self.lhs = self.rhs+1
        else :
            self.lhs = self.notacklist[1]
        self.resendtime=time.time()
        self.resendpos = 0
    self.notacklist.remove(seq)
    if self.rhs == int(self.num)-1 and len(self.notacklist)==0:
        self.printmessage()
        self.shutdown()
```

在这个函数中，先取出ack包中的序号看看目前ack的是哪个包，并在self.notacklist移除这个表项，如果ack的是self.notacklist中的第一个包，说明目前lhs可以开始移动了。

lhs移动的方式：

- 如果说目前self.notacklist长度为1，只有一个没被ack，说明lhs应该移动到rhs + 1的位置，因为所有小于rhs+1的都已经被ack
- 如果说目前self.notacklist长度不为1，lhs应该移动到self.notacklist[1]的位置

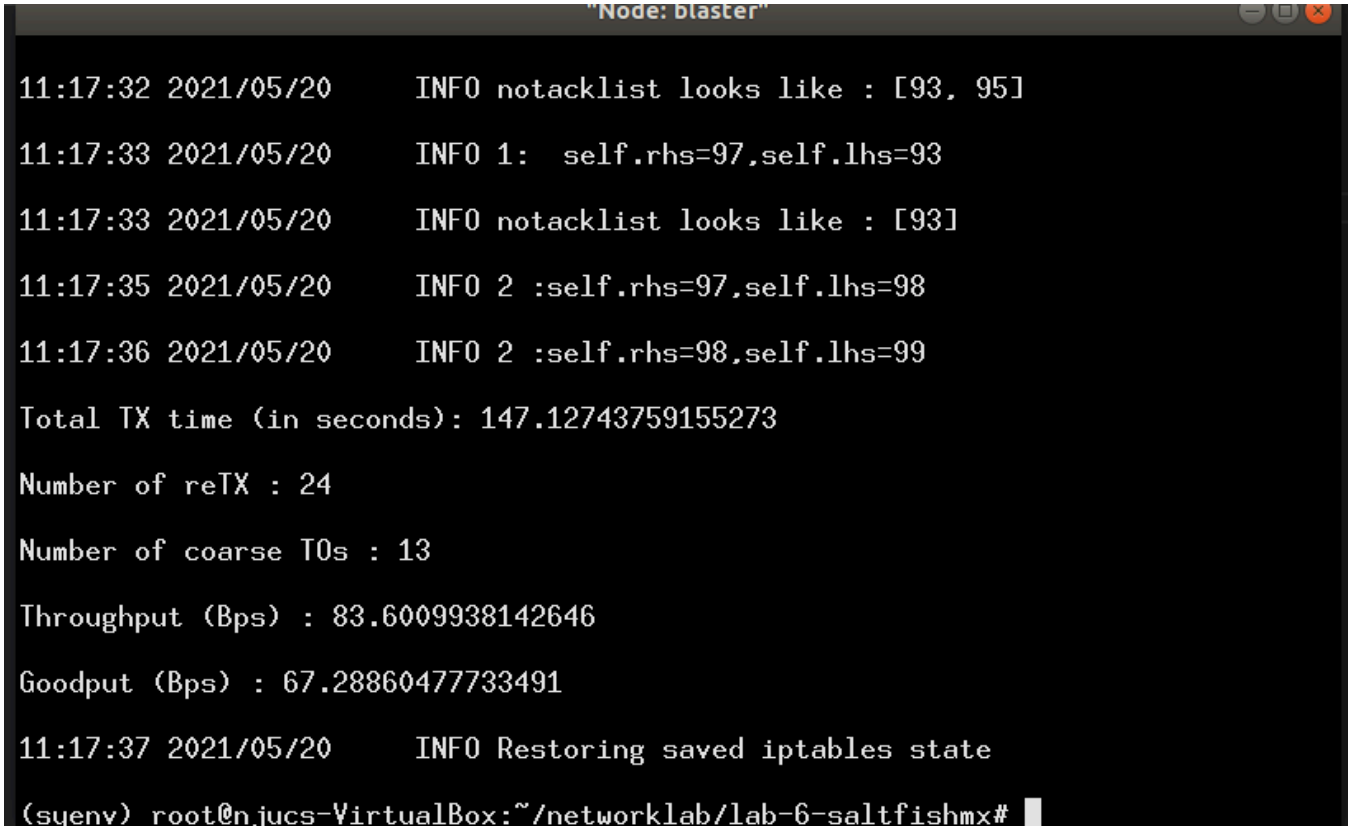
如果收到包是rhs == num-1而且self.notacklist为空，说明所有的都ack了，进入打印信息的流程。

Task 5: Running your code

1. 当参数为：

> 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'
```



上图下面几项是输出的结果（最后rhs=98是因为在handlenopacket函数里我先输出再增加了rhs的值）

```
11:17:31 2021/05/20 INFO notacklist looks like : [93, 95]
11:17:32 2021/05/20 INFO 1: self.rhs=97,self.lhs=93
11:17:32 2021/05/20 INFO notacklist looks like : [93, 95]
11:17:33 2021/05/20 INFO 1: self.rhs=97,self.lhs=93
11:17:33 2021/05/20 INFO notacklist looks like : [93]
```

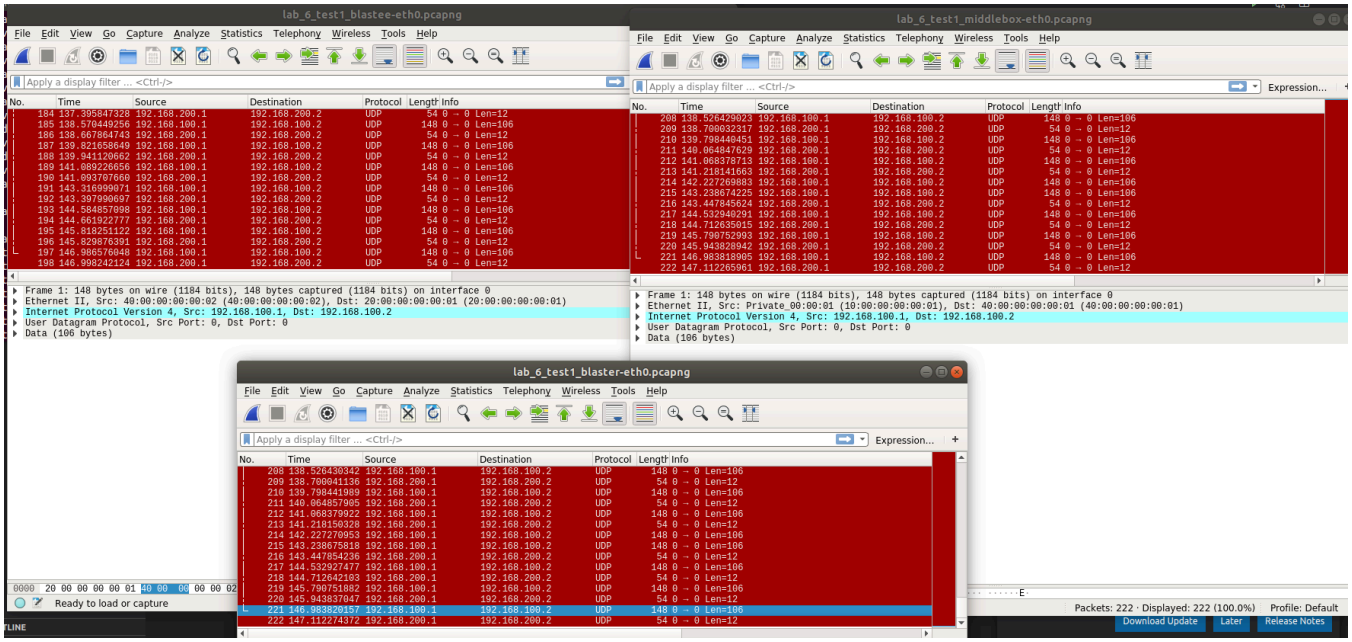
在上图中可以看到，self.notacklist中有93和95，而先被ack的是95，此时lhs是不会变的

```
11:17:26 2021/05/20     INFO notacklist looks like : [91, 92, 93, 95]
11:17:27 2021/05/20     INFO 2 :self.rhs=95,self.lhs=92
11:17:29 2021/05/20     INFO 1:  self.rhs=96,self.lhs=92
11:17:29 2021/05/20     INFO notacklist looks like : [92, 93, 95]
11:17:30 2021/05/20     INFO 2 :self.rhs=96,self.lhs=93
```

在上图中可以看到，self.notacklist中有91，92，93和95，91被ack后，lhs的值改为了92，说明目前序号为92以前的包已经全部被ack

上述的情况都符合我们的期望

抓包结果：



看起来不是很清新易懂，看log info就行了

2. 调整参数为：droprate = 0.5，senderwindow = 10，其他不变
   做到这里时因为self.notacklist中的表项增多了，发现了前面的一个bug，在handle_packet中加了这样一句

```
if self.resendpos>0:
    self.resendpos-=1
```

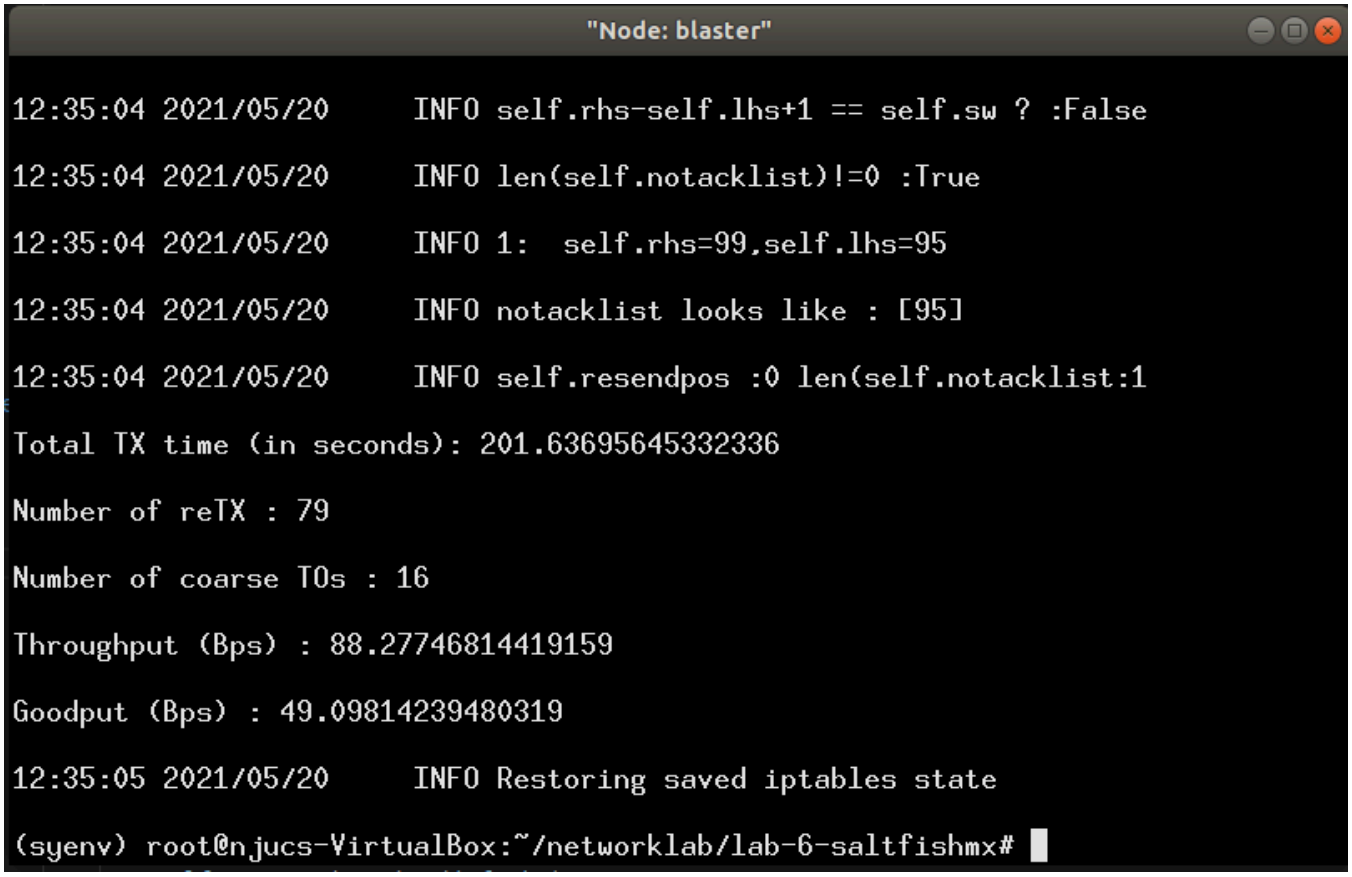
因为原来收到ack包没有改变resendpos的位置，导致notacklist的长度发生变化后还是按照原来的索引在发送重传包

另一个bug是重传条件的判断，改为

```
if len(self.notacklist)!=0 and (int(self.rhs -self.lhs +1) == int(self.sw) or self.rhs == int(self.
```

原来对rhs = num-1的边界情况没有考虑仔细

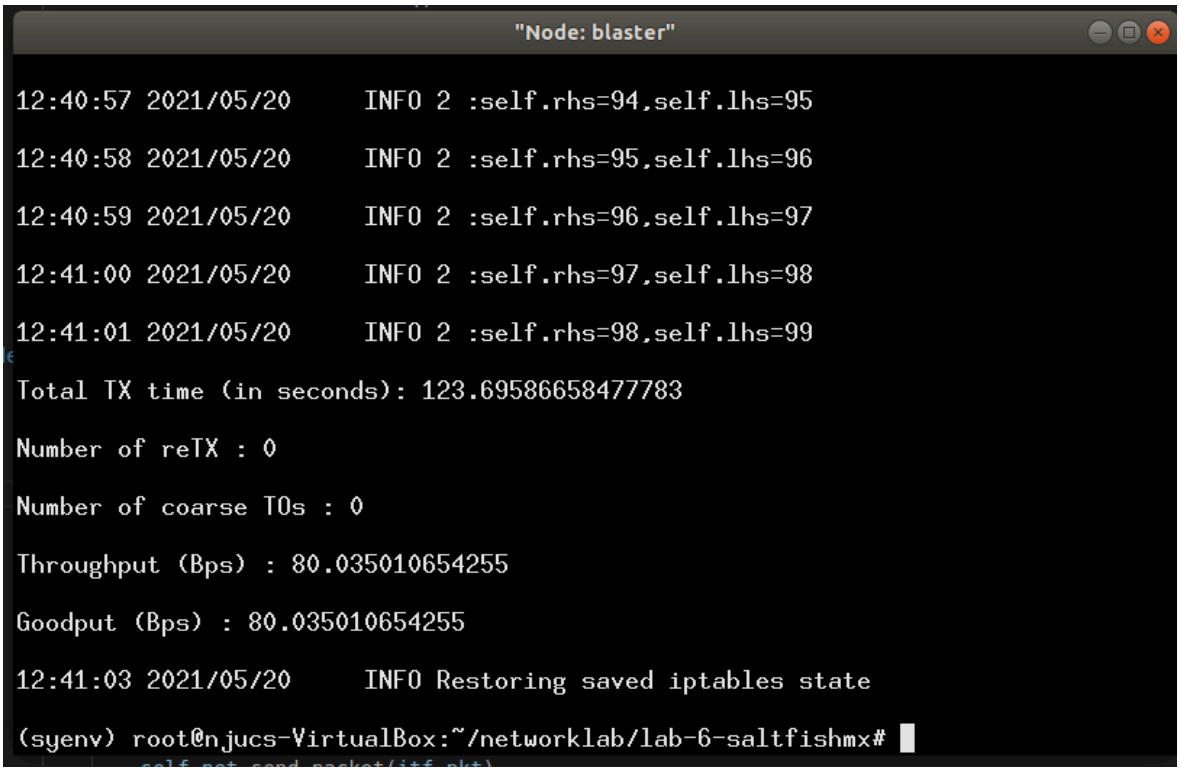
实验结果：



可以看到，这次由于丢包率的上升，corase timeout的次数虽变化不大，但是重传包的次数远远增加，而且用的时间也增加了。

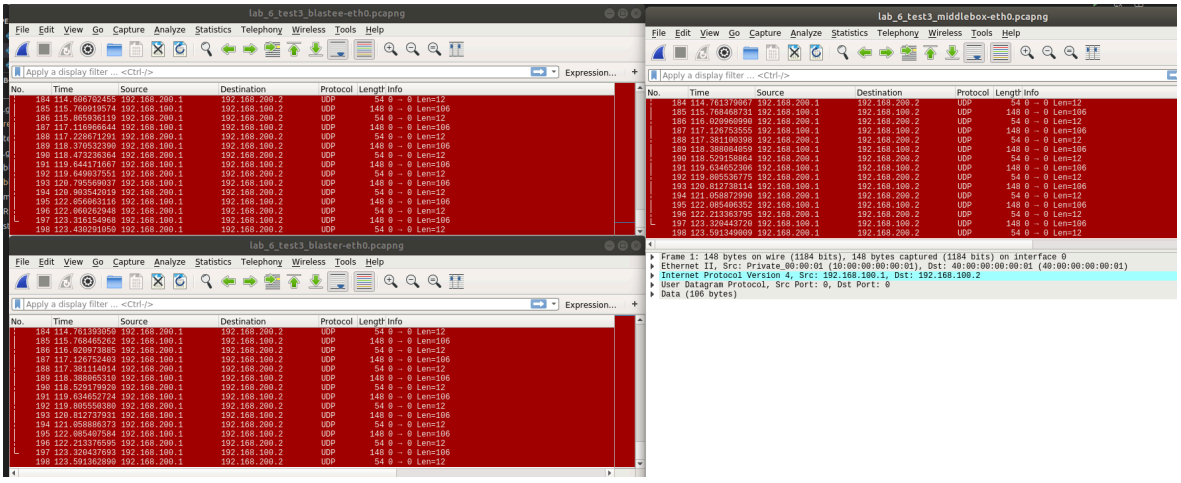
goodput占throughput的比例也下降了

3. 调整参数为droprate = 0，其他不变
   实验结果：



可以看到由于没有丢包，重传次数为0，而且goodput和throughput相等

抓包结果：



可以看到由于没有丢包，三个窗口观察到的包的数量都是一样的，因为没有发生重传。

综合上面的多个实验，可以看出经过一些bug的修改，实验成功

总结

感觉这次实验理思路的过程比较艰难，一开始没看懂`packet.get_header(RawPacketContents)`取的是哪部分，想的比较复杂
这次实验debug和测试的过程比较有意思，因为不是按照测试用例编程，一开始做完droprate=0.19以为没有问题了，结果改成droprate=0.5又出了问题，这种找问题再解决的过程还是挺好的。
完