南京大学本科生实验报告

181860097 计算机科学与技术系 王明骁
Email：1164151483@qq.com
开始日期：2020年5月16日 结束日期：2020年5月20日
lab6

实验名称

Lab 6: Reliable Communication

实验目的

- 理解TCP协议实现可靠传输的原理
- 在TCP协议的基础上进行简化得到本次实验的可靠传输原理，并将其实践，以达到可靠传输的目的

task1

1. **实验内容**

   task1为准备阶段，只需要按照实验手册上的说明，将相关文件copy到lab_6文件夹并做好命名即可

task2

1. 实验内容

   task2的内容为实现一个middlebox，由于mac地址已经硬编码在代码中，因此该middlebox只需要完成以下三个任务即可：

   - 读取文件，获取丢包率参数
   - 针对blaster发往blastee的包，按照丢包率随机丢包
   - 针对blastee发往blastee的包，不丢包

2. 实验结果

   等待所有内容完成后一起测试

3. 核心代码

```
if gotpkt:      You, 3 days ago • finish task1
    log_debug("I got a packet {}".format(pkt))
    #debugger()
    if dev == "middlebox-eth0":
        log_debug("Received from blaster")
        '''
        Received data packet
        Should I drop it?
        If not, modify headers & send to blastee
        '''
        ran = random.random()
        if ran < droprate:
            continue
        pkt[0].dst = blasteemac
        net.send_packet("middlebox-eth1", pkt)

    elif dev == "middlebox-eth1":
        log_debug("Received from blastee")
        '''
        Received ACK
        Modify headers & send to blaster. Not dropping ACK packets!

        '''
        pkt[0].dst = blastermac
        net.send_packet("middlebox-eth0", pkt)
```

收到一个包后，首先判断方向，如果是从blaster发往blastee的包，则生成随机数ran，然后用ran和从文件中读取到的丢包率droprate比较，如果ran较小，则丢包（与丢包率越大越容易丢包一致），否则则正常传递该包。如果是从blastee发往blaster的包，则直接传递，不丢包

task3

1. 实验内容

   task3要求实现blastee，blastee只需要根据收到的包回复ack即可，其中回复的包的序列号与收到的包的序列号相同，负载部分是收到的包的负载的前八个字节，如果收到的包不满八个字节，则不满的部分任意填充即可

2. 实验结果

   等待所有内容写完后一起测试

3. 核心代码

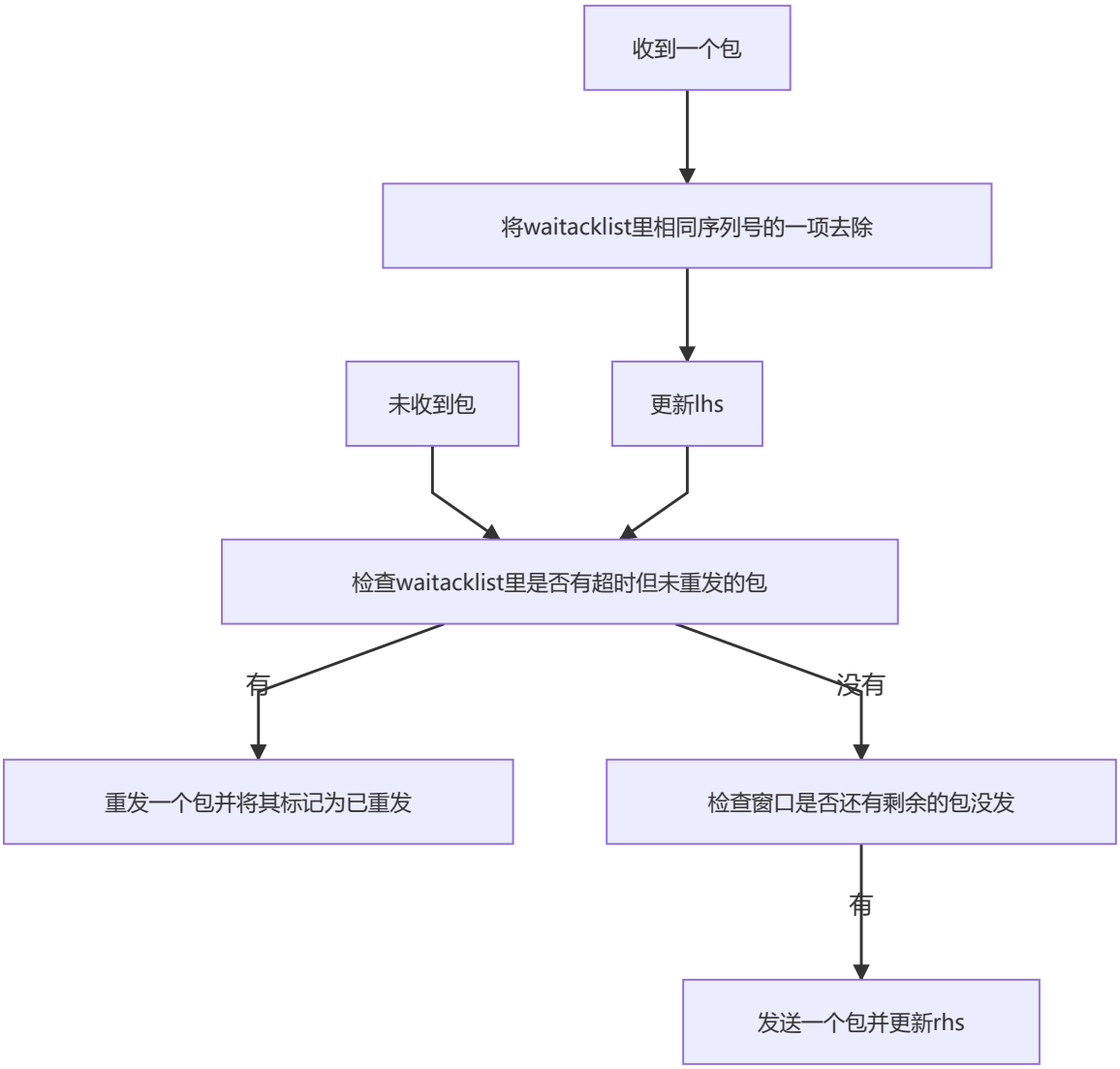
```
if gotpkt:
    #debugger()
    log_debug("I got a packet from {}".format(dev))
    log_debug("Pkt: {}".format(pkt))
    ipv4 = pkt.get_header(IPv4)
    mymac = mymacs[0]
    myip = '192.168.200.1'
    if ipv4 is None:
        continue
    if str(ipv4.src) != str(blasterip):
        continue
    temp1 = pkt[3].to_bytes()[:4]
    temp2 = pkt[3].to_bytes()[4:6]
    seqencenumber = int.from_bytes(temp1,byteorder = 'big')
    payloadlength = int.from_bytes(temp2,byteorder = 'big')
    if payloadlength > 8:
        payloadlength = 8
    #debugger()

    #payload = pkt[4].to_bytes()[:len(pkt[4])]
    payload = pkt[3].to_bytes()[6: 6 + payloadlength]
    while payloadlength < 8:
        payload += b' '
        payloadlength += 1

    seq = RawPacketContents(seqencenumber.to_bytes(4,byteorder = 'big'))
    pay = RawPacketContents(payload)
    e = Ethernet(src = mymac,dst = blastermac)
    ip = IPv4(src = myip,dst = blasterip ,protocol = IPProtocol.UDP,ttl = 8)
    udp = UDP(src = 2,dst = 1)
    sendpkt = e + ip + udp + seq + pay      You, 3 days ago • finish task3
    net.send_packet(dev, sendpkt)
```

收到包之后首先进行一些增加鲁棒性的判断，确保收到的包是blaster发来的，然后取出其序列号和负载长度，如果该负载长度大于8，则取出负载的前八个字节作为ack的负载，否则取出其所有的负载并在末尾添加若干的空格字节，直至达到负载长度为8，而序列号部分则根据手册上的要求，以发送包的序列号为ack的序列号，使用rawpacketcontents将其构造进一个包中，最后配置好相关信息之后即可发送

task4

1. 实验内容：

   task4要求实现一个blaster，blaster的实现是整个系统可靠传输的关键，具体实现逻辑如下：



其中，在更新lhs时，若如lhs发生变化，则说明窗口向右移动，则更新lhstime为当前时间。而在每次循环时，都会去检查当前时间减去lhstime是否大于timeout，若大于，则说明当前窗口已超时，更新lhstime为当前时间，并将waitacklist里的所有的包都标记为超时且未发送

2. 实验结果：

   在task5中验证

3. 核心代码：

   创建一个要发送的包：将其封装为一个函数如下

```
def createpkt(payloadlength,seq):

    blasteemac = '20:00:00:00:00:01'
    blastermac = '10:00:00:00:00:01'
    blasteeip = '192.168.200.1'
    blasterip = '192.168.100.1'

    pkt = Ethernet(src = blastermac,dst = blasteemac , ethertype = EtherType.IPv4) + IPv4(src = blasterip, dst = blas
    pkt31 = RawPacketContents(seq.to_bytes(4,byteorder = 'big'))
    pkt32 = RawPacketContents(payloadlength.to_bytes(2,byteorder = 'big'))
    pkt4 = RawPacketContents(seq.to_bytes(payloadlength,byteorder = 'big'))
    pkt += pkt31
    pkt += pkt32
    pkt += pkt4
    return pkt
```

   首先将ip，mac地址都硬编码在代码中，然后进行相关配置（mac地址，ip地址，protocol，ttl等），负载部分首先是序列号，按照大端存放的方式，然后是负载长度，最后是负载，由于负载可以是任意内容，因此这里为了方便使用序列号作为负载，长度为payloadlength，最后返回pkt得到一个可以发送的udp包

每次循环时的检查超时：

```
if time.time() - lhstime > timeout_time:
    timeoutcnt += 1
    lhstime = time.time()
    for index in waitacklist:
        index.istimeout = True
        index.isresend = False
```

每次循环时检查是否超时，若超时，则超时计数加一，lhstime更新为当前时间，同时对在waitacklist中的每一项都标记为已超时且未重传

收到一个ack时：

```
    prelhs = lhs
    temp1 = pkt[3].to_bytes()[:4]
    seqencenumber = int.from_bytes(temp1,byteorder = 'big')
    for index in waitacklist:
        if index.seq == seqencenumber:
            waitacklist.remove(index)
            break
    if len(waitacklist) == 0:
        lhs = rhs
        if lhs >= num:
            alltime = time.time()-starttime
            log_info("total tx time:{}\n".format(alltime))
            log_info("number of retx:{}\n".format(recnt))
            log_info("number of coarse tos:{}\n".format(timeoutcnt))
            log_info("throughput: {}\n".format(allbytes/alltime))
            log_info("goodput: {}\n".format(usebytes/alltime))
            break
    else:
        lhs = num + 1
        for index in waitacklist:
            if index.seq < lhs:
                lhs = index.seq      You, a day ago • fix blaster bug
    if lhs != prelhs:
        lhstime = time.time()
    for index in waitacklist:
        if index.isresend == False and index.istimeout == True:
            net.send_packet(my_intf[0].name,index.pkt)
            allbytes += payloadlength
            index.isresend = True
            recnt += 1
            break
```

收到一个ack时，首先保存现在的lhs，然后取出收到的ack的序列号，将waitacklist中序列号相同的那一项删除，之后进行更新lhs，更新时首先检查waitacklist是否为空，若为空，则说明没有已发送但未ack的包，因此直接把lhs更新为rhs，然后判断是否结束，若结束，则按照手册要求输出相关信息。若waitacklist不为空，则lhs为waitacklist中序列号最小的一项，更新完lhs之后，判断lhs是否发生变化，若发生变化，则说明窗口整体向右移动，故此时更新lhstime，最后检查waitacklist里是否有已超时但未重发的数据包，若有，则将其发送，并标记为已重发

未收到ack时：

```
else:
    log_debug("Didn't receive anything")

    flag = 1
    for index in waitacklist:
        if index.isresend == False and index.istimeout == True:
            #log_info("resend a packet:{}".format(index.seq))
            net.send_packet(my_intf[0].name,index.pkt)
            allbytes += payloadlength
            index.isresend = True
            flag = 0
            recnt += 1
            break
    if flag == 0: #sended a packet
        continue

    if rhs - lhs + 1 <= swsize and rhs <= num:
        sendpkt = createpkt(payloadlength,rhs)
        #log_info("send a packet{}".format(rhs))
        net.send_packet(my_intf[0].name,sendpkt)
        usebytes += payloadlength
        allbytes += payloadlength
        temp = waititem(sendpkt,False,False,rhs)
        waitacklist.append(temp)
        rhs += 1
```

检查是否有已超时但未重发的数据包，若有，则将其标记为已重发，并发送，发送过之后本次循环将不再发包。若未发送过，则判断窗口中是否还有未发送的包，若有，则使用createpkt构造一个新的包，序列号为rhs，并将其发送，同时将其添加到waitacklist中，然后rhs加一即可

task5

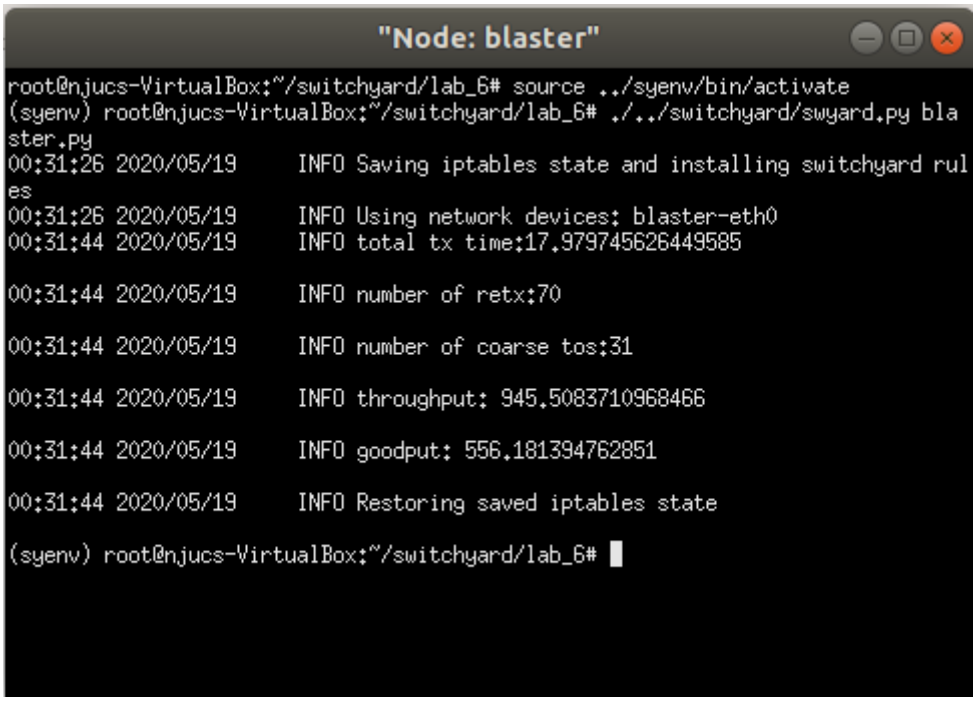
task5是对本次实验进行测试

首先配置middlebox的丢包率为0.19

blaster的配置如下：

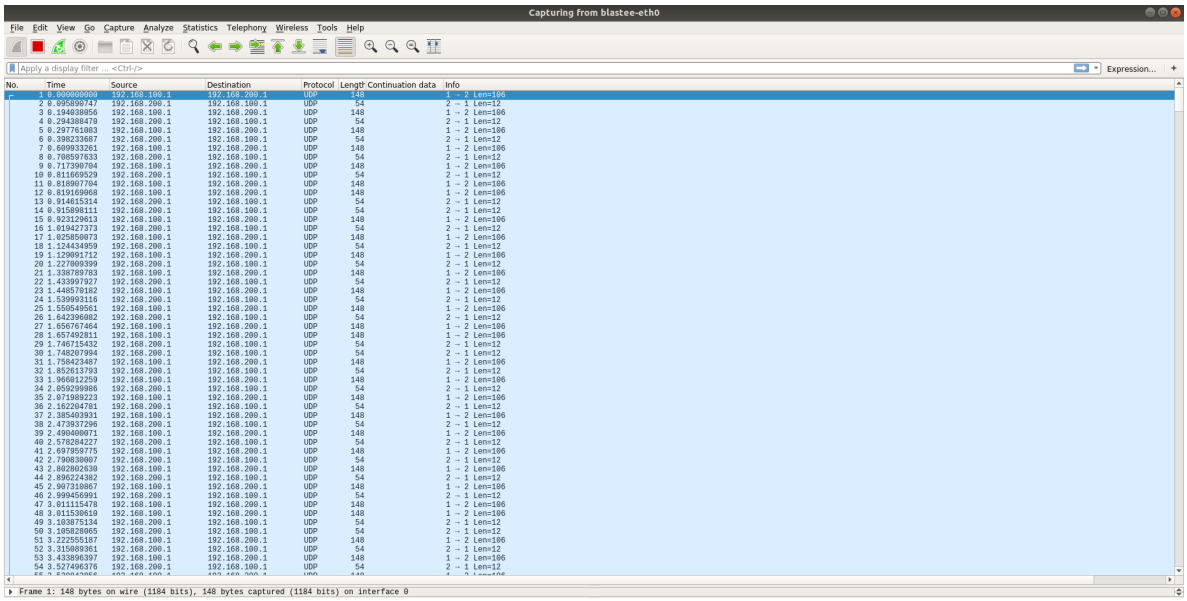
```
-b 192.168.200.1 -n 100 -l 100 -w 5 -t 300 -r 100
```

输出结果：

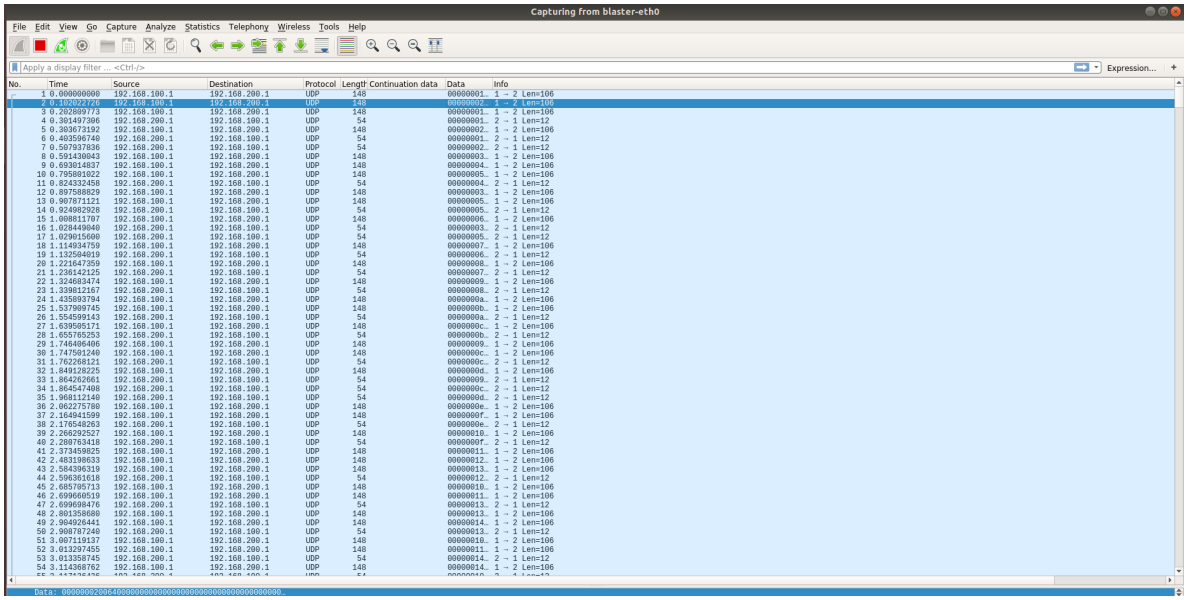


用wireshark抓包得到的结果如下：

blastee：



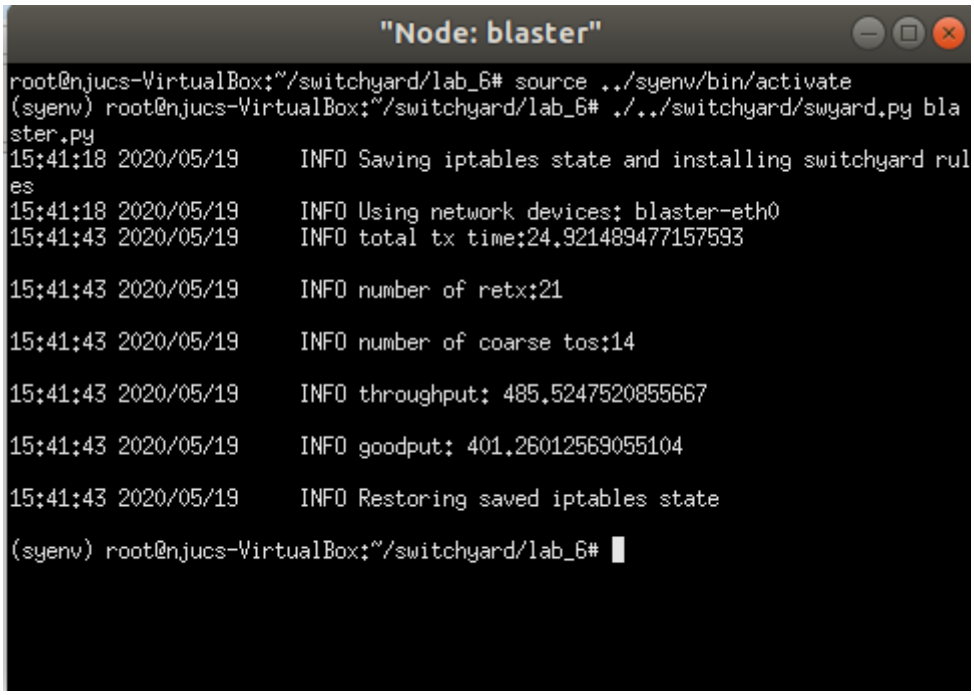
blaster：



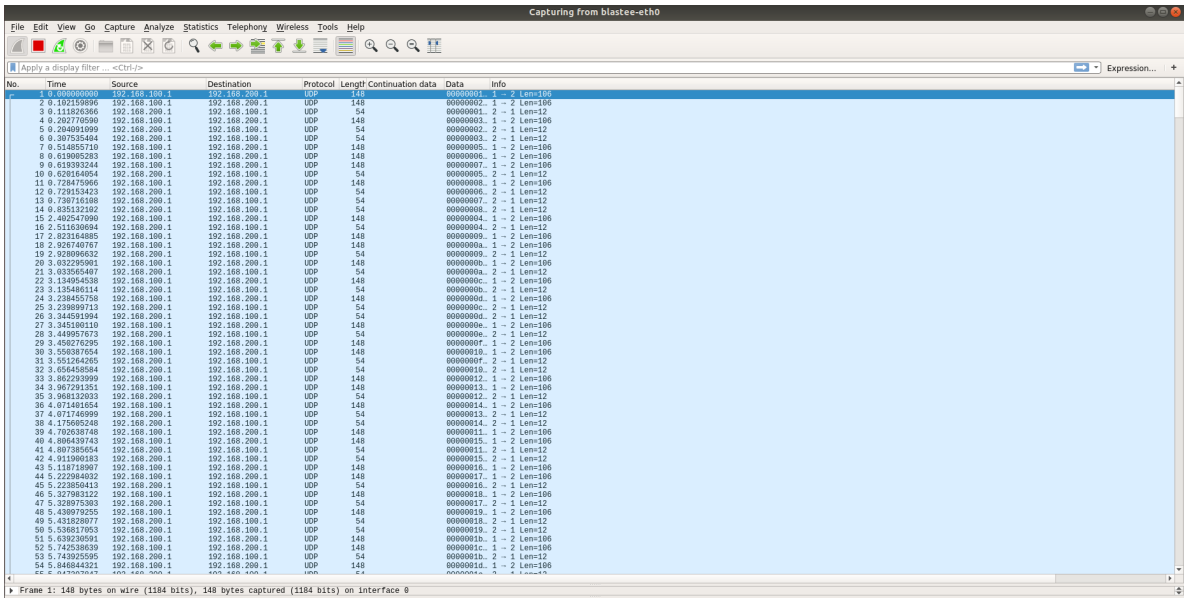
从输出结果中可以看出，尽管丢包率只有0.19，但是重传的包的数量达到70个之多，细看wireshark在blaster抓的包可以知道，这是因为超时的时间配置的太短，从wireshark中可以看出，blaster首先发了1和2，然后触发了超时，又发了1，然后收到了1的ack，重发超时的2，然后又收到了1和2的ack，开始发3，由此可以知道其实数据包1并没有被丢包（因为blastee发了两个ack）但是由于超时的时间太短，甚至时间短于从blaster发一个包再到blastee发送ack让blaster收到的时间，因此尽管丢包率只有0.19，但是重传了70个数据包

然后将超时时间增加到900，其余配置不变，进行第二次测试：

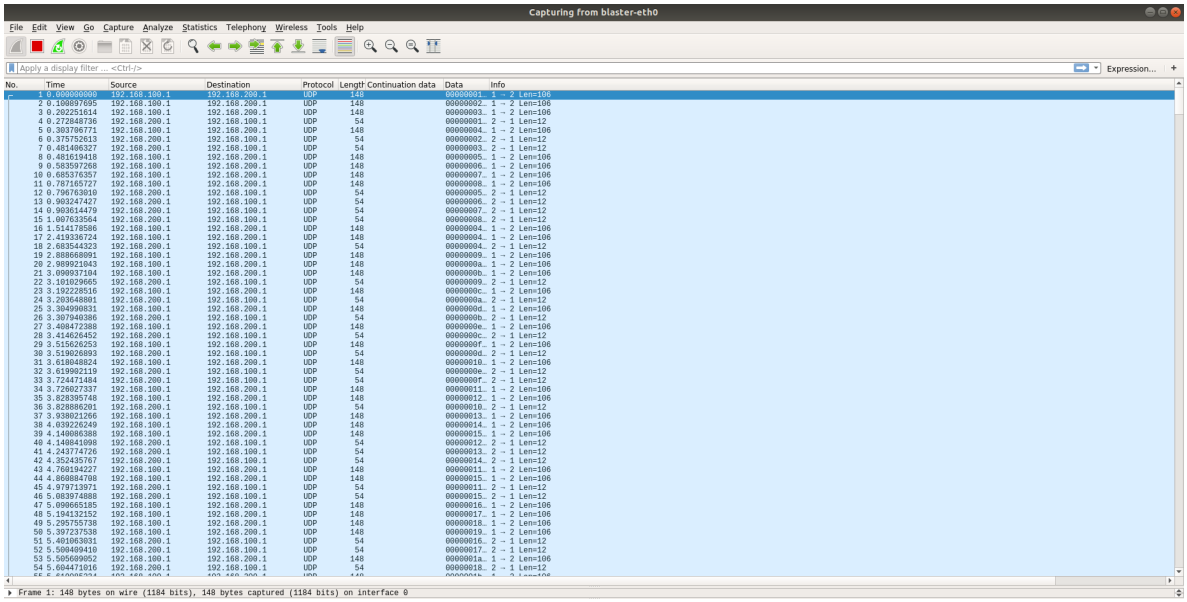
输出结果：



blastee：



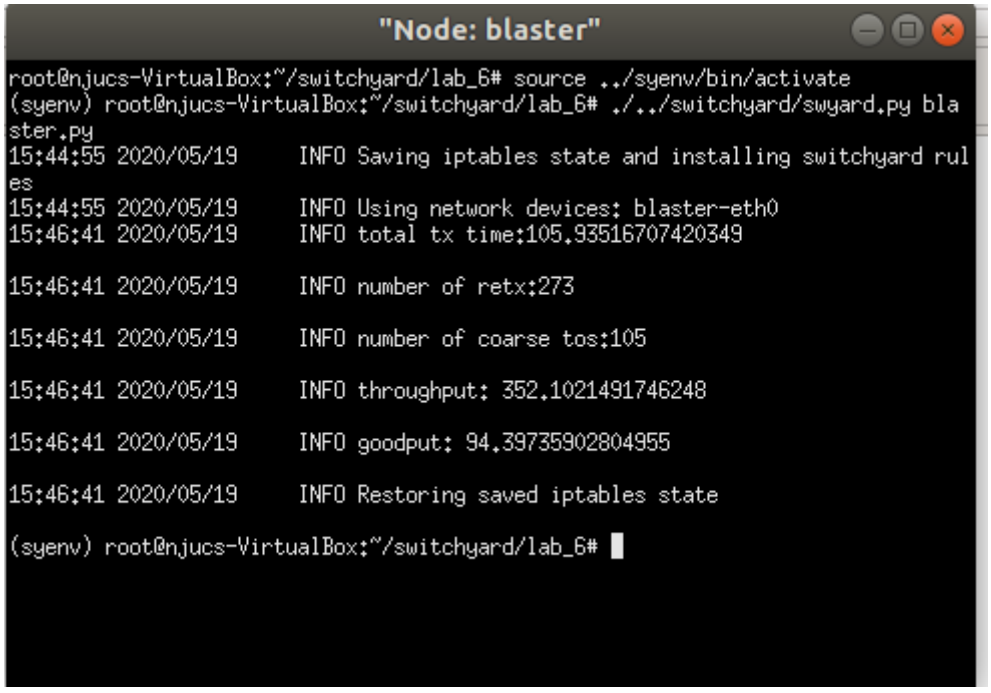
blaster：



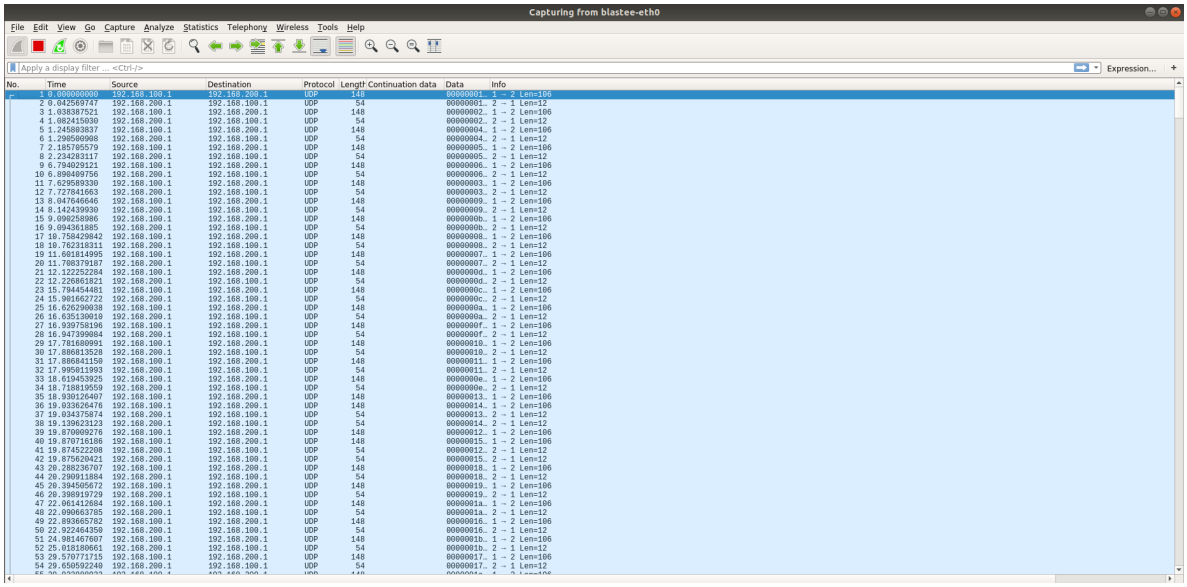
由于这次测试超时时间设置的较长，因此对于一个超时的数据包，基本上就是因为丢包，而不会是因为blastee还未来得及发ack，重传了21个包，与丢包率为0.19也比较吻合，而根据截图中列出的每个包的序列号，可以基本判断balster的逻辑是正确的，这里详细解释前18个包，首先发送123，然后收到1，发送4，收到23，发送5678，然后收到5678，此时由于4一直未收到，触发超时，而窗口45678只有4没收到，lhs不能向右移动，限制rhs也不能向右移动，因此在收到5678期间并未发送9，与理论结果符合，然后重发4，还是未收到ack，又重发4，终于收到了ack，此时lhs更新到9，rhs可以向右移动，因此此时开始发送数据包9，与理论结果符合

最后在第二次测试的基础上修改丢包率为0.7

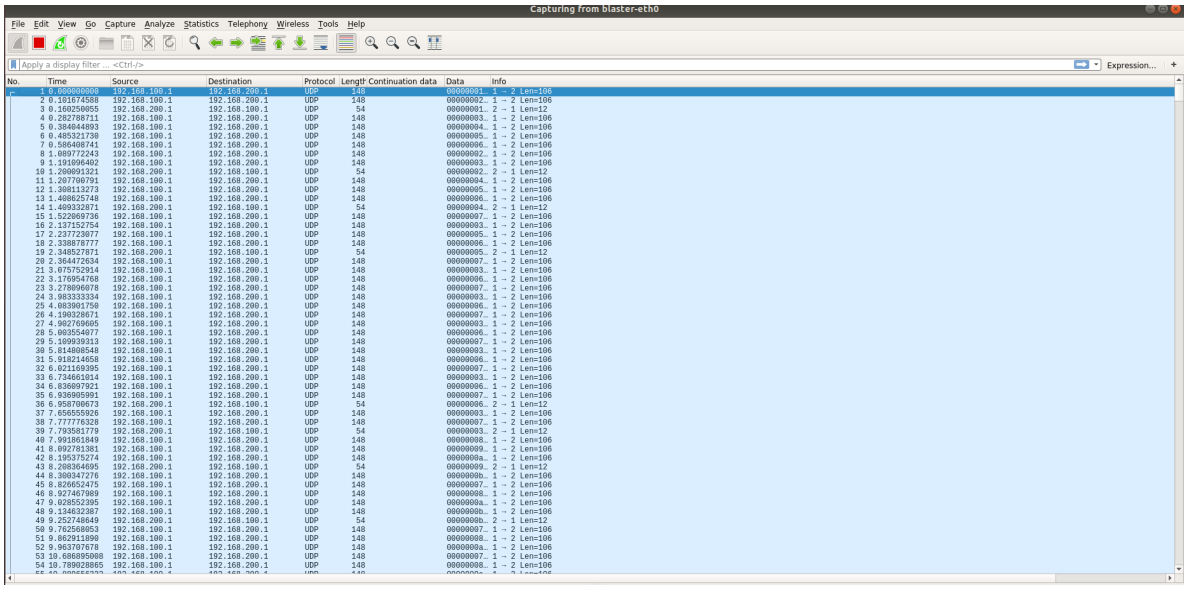
实验结果：



blatee：



blaster：



从输出结果可以看出，本次测试重发的包有273个，触发了105次超时，从wireshark中也可以看出，最开始时发送12，然后收到1的ack，然后发送3456，但是一个ack也没收到，窗口并未向右移动，等到超时之后，开始重发23，此时收到2的ack，重发4，可以看出此时虽然窗口还有未发的包7，但是仍然先发超时的包，与理论结果相符

通过以上三个测试，基本上可以判断该可靠传输得以正确实现

总结与感想

- 首先感谢曾晨凯同学教我怎样在wireshark中把数据包的序列号显示出来，以便验证是否正确
- 本次实验让我深刻理解了tcp协议的可靠传输原理
- 本次实验过程在调试时遇到了一些玄学bug，例如在blaster的if gotpkt分支中增加debugger()语句，作为断点设置，理论上说当第一次运行到该语句时，blaster应该是收到第一个包，但是实际上发现用wireshark抓包时，抓到blaster已经收到了很多个包，当时询问助教，助教也不知道为什么。现在想了一下有可能是因为缓冲区的问题，即每次收到包时将其放入缓冲区，此时wireshark就可以抓到包，然后使用recv_packet可能只是从缓冲区里取一个包，这样就可以解释wireshark抓到很多包的矛盾，但是也只是猜想，也不知道怎么验证