Lab6_Report

课程名称：**计算机网络** 任课教师：田臣/李文中 助教：

学院	计算机	专业（方向）	计算机
学号	181860077	姓名	佘帅杰
Email	3121416933@qq.com	开始完成日期	2020.5.26

实验名称

计算机网络试验6

实验目的

学习滑动窗口原理

实验内容

实现滑动窗口以及超时重发等功能

核心代码

MiddleBox

预处理

读入文件

后续判断的时候是生成一个1-100的数字，所以把c乘上了100

```
file = open("middlebox_params.txt")
line = file.readline()
line=line.strip('\n')
d=line.split(" ")
c=float(d[1])
c*=100
```

核心逻辑

收到了普通数据报，生成随机数判断大小，否则就修改Mac地址，发送

（由于调试的原因，这里还提取了数据报的Seq用于输出）

```
if dev == "middlebox-eth0":
    log_debug("Received from blaster")
    '''
    Received data packet
    Should I drop it?
    If not, modify headers & send to blastee
    '''
    temp=randint(1,100)
    if temp <= c:
        print("Packet drop")
    else:
        pkt[Ethernet].dst=EthAddr('20:00:00:00:00:01')
        seq2=pkt[3].to_bytes()[0:4]
        seq3=int.from_bytes(seq2, byteorder='big', signed=False)
        net.send_packet("middlebox-eth1", pkt)
        print("send pac"," ",seq3)
```

收到了Ack数据报

```
elif dev == "middlebox-eth1":
    log_debug("Received from blastee")
    '''
    Received ACK
    Modify headers & send to blaster. Not dropping ACK packets!
    net.send_packet("middlebox-eth0", pkt)
    '''
    pkt[Ethernet].dst=EthAddr('10:00:00:00:00:01')
    net.send_packet("middlebox-eth0", pkt)
    seq2=pkt[3].to_bytes()[0:4]
    seq3=int.from_bytes(seq2, byteorder='big', signed=False)
    print("send ACK ",seq3)
else:
    log_debug("Oops :))")
```

同理，只不过不进行丢弃判断，修改Mac后直接发送就可以了，同样为了调试提取Seq

Blastee

Blastee实际上是验证机器，收到了包就提取Seq，然后构造Ack就可以

Ack的结构由手册得到，然后抽取Seq的手法和之前一样

最后Hard Code地址等信息之后发出包即可

```
if gotpkt:
    #log_info("I got a packet from {}".format(dev))
    #log_info("Pkt: {}".format(pkt))
    #seq=int.from_bytes(pkt[3][0:4], byteorder='big', signed=False)
    #payload=int.from_bytes(pkt[3][6:14], byteorder='big', signed=False)
    #print(seq)
    #print(payload)
    seq2=pkt[3].to_bytes()[0:4]
    payload2=pkt[3].to_bytes()[6:14]
    #print(seq2)
    #print(payload2)
    seq=int.from_bytes(seq2, byteorder='big', signed=False)
    payload=int.from_bytes(payload2, byteorder='big', signed=False)
    print(seq)
    #print(payload)

    pkt = Ethernet() + IPv4() + UDP()
    pkt[1].protocol = IPProtocol.UDP
    pkt += seq2
    pkt += payload2
    pkt[Ethernet].src=EthAddr('20:00:00:00:00:01')
    pkt[IPv4].src=IPv4Address(ip)
    pkt[Ethernet].dst=EthAddr('10:00:00:00:00:01')
    pkt[IPv4].dst=IPv4Address('192.168.100.1')
    net.send_packet(my_interfaces[0],pkt)
```

Blaster

数据结构

这个结构是核心队列的基本元素结构

pac就是包，seq是序列号（方便使用，当然可以每次从包里重新取），标志位是否Ack

```
class Node():
    def __init__(self,pacs,seqs):
        self.pac=pacs
        self.seq=seqs
        self.flag=0
```

核心代码

初始化

读文件

对微秒进行处理（time模块用的是秒为单位）

设置初试序列号，设置LHS和RHS，设置初试计时器

```
file = open("blaster_params.txt")
line = file.readline()
line=line.strip('\n')
d=line.split(" ")
print(d)
ip=d[1]
num=d[3]
length=d[5]
window=d[7]
timeout=d[9]
timeout=float(timeout)/1000
timeout=1
print(timeout)
rev_time=d[11]
rev_time=float(rev_time)/1000
q=[]
seq=1
LHS=1
RHS=1
timer=time.time()
cur=time.time()
```

收到包之后处理Ack包

提取Seq，手法和之前的一样，然后在当前的窗口里查找对应的包，设置标志位

如果队列不空，又有当前队列的头部是已经Ack了的包，那就删除头部的包，然后调整LHS的值，刷新计时器，之后一直循环，会把头部元素以及后续可能在头部元素之前就Ack的包一起删除。

```
if gotpkt:
    seq2=pkt[3].to_bytes()[0:4]
    seq3=int.from_bytes(seq2, byteorder='big', signed=False)
    for i in q:
        if i.seq==seq3:
            print(seq3,"Has Acked")
            i.flag=1
    while len(q)>0:
        if q[0].flag==1:
            LHS = q[0].seq+1
            del(q[0])
            cur=time.time()
        else:
            break
```

如果没有收到包

那就看看是不是已经发完了所有的包同时等待Ack的队列已经空了

如果不满足上述条件说明工作没有结束

先看看超时，超时机制如前面的实验，使用两个变量，一个是当前的最新时间，一个是旧的标记时间，如果触发超时机制，那就在队列里找到第一个没有标记的发出（由于一轮只会发出一个包，所以发出后工作结束）

反之，如果没有超时，那就看看是不是满足发出条件，同时有可能是所有包发出但是没有有一些没有Ack，这种情况不再发包而是归入超时处理（如100个预设的包都发出了，但是97没有Ack，那就不再发101，而是等97超时触发重发）

重发就是构造包，放入seq，设置好地址。让RHS指向新发的包的序列号，然后刷新Seq，发出，并放入待Ack窗口。结束工作

```
'''
if seq>int(num) and len(q)==0:
    endtime=time.time()
    TotalTime=endtime-starttime
    print("Total time",TotalTime)
    print("Number of reTX: ",retrainsport)
    print("Number of coarse TOs: ",timeout_times)
    print("Throughput (Bps): ",(retrainsport+int(num))*int(length)/TotalTime)
    print("Goodput (Bps): ",int(num)*int(length)/TotalTime)
    exit()
timer=time.time()
print("timer",timer," ",cur)
if timer-cur>timeout:
    for i in q:
        if i.flag==0:
            print("Timeout resend ",i.seq)
            net.send_packet(my_intf[0],i.pac)
            timeout_times+=1
            retrainsport+=1
            break
else:
    if RHS-LHS+1 < int(window) and seq<=int(num):
        print("able to send  ",seq)
        pkt = Ethernet() + IPv4() + UDP()
        pkt[1].protocol = IPProtocol.UDP
        pkt += seq.to_bytes(4,byteorder='big', signed=False)
        pkt += int(length).to_bytes(2,byteorder='big', signed=False)
        payload = b'These are some application data bytes'
        payload=payload[0:int(length)-1]
        pkt+=payload
        pkt[Ethernet].dst=EthAddr('20:00:00:00:00:01')
        pkt[IPv4].dst=IPv4Address(ip)
        pkt[Ethernet].src=EthAddr('10:00:00:00:00:01')
        pkt[IPv4].src=IPv4Address('192.168.100.1')
        RHS=seq
        temp=Node(pkt,seq)
        q.append(temp)
        seq+=1
        net.send_packet(my_intf[0],pkt)
```

测试样例

Mininet测试

多次参数尝试后选择了比较有代表性的来测试

test1

先用简单的测试末尾为例说明分析过程

PS：（延时不能设置的太短，太短的话很快就重传了，很难出现因为丢包以及窗口大小限制导致的被迫重传。简单的说就是一丢包就被发现重传（RHS和LHS看起来贴的很近），不容易观察和验证）

pac num=100

drop rate=0.19

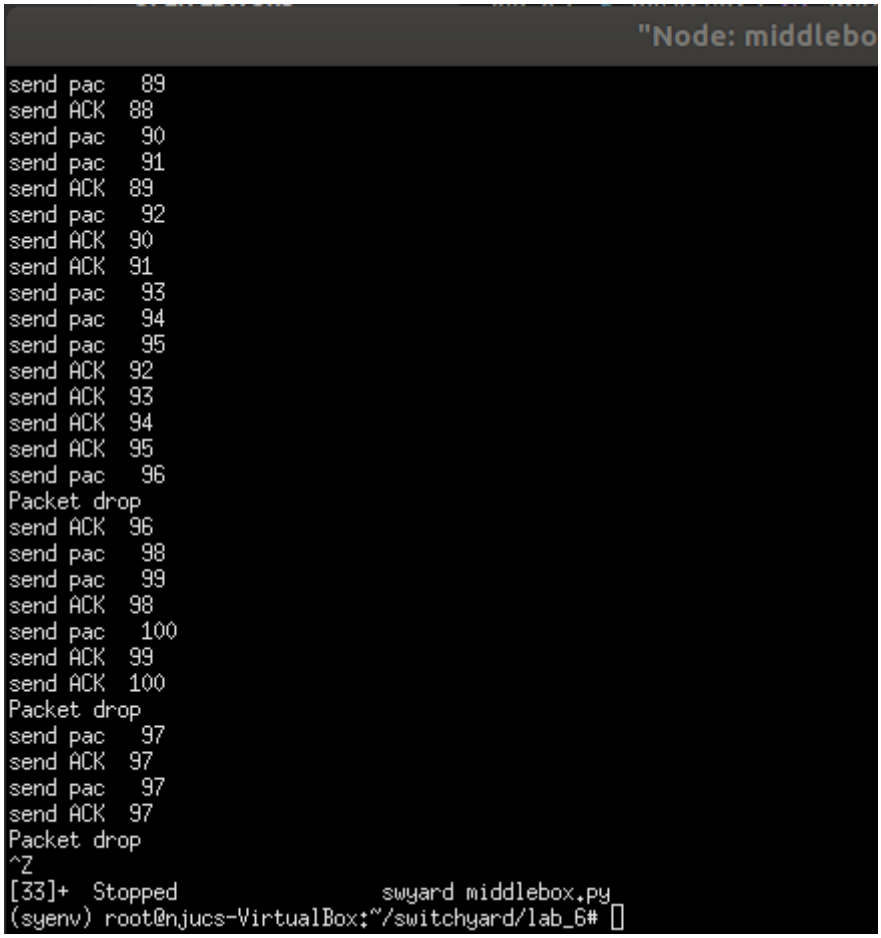
timeout val=1s=1000ms

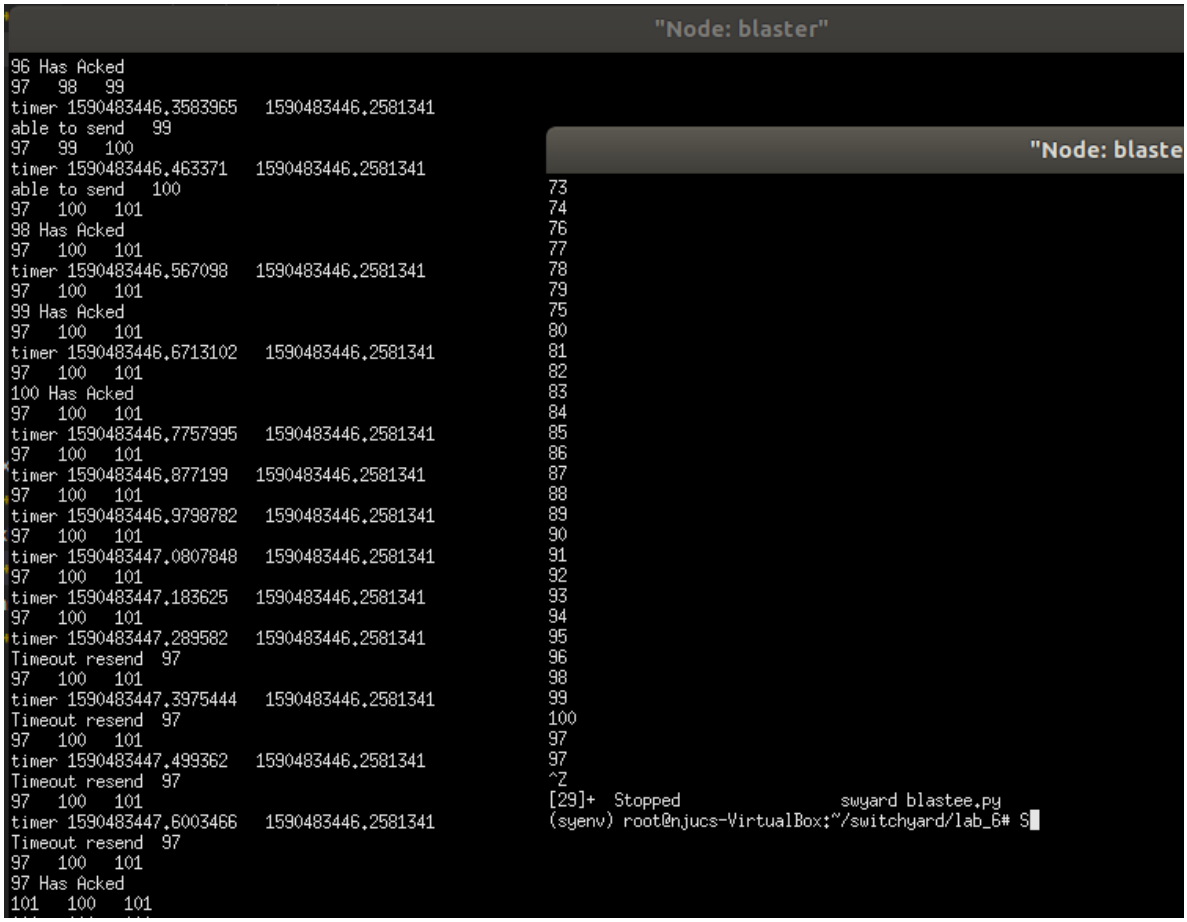
第一张图是middlebox的输出，middlebox在收到包之后要么输出是普通包还是ACK以及编号，或者是输出drop

这里分析就以图中96之后的Packet Drop为例

这里是把97号包丢弃

此时由上下文可以看到已经Ack了92，93，94，95





这一张图是blaster和blastee的输出，blaster输出三个数的行是在输出LHS，RHS和Seq。其他的输出显然

blastee的输出就是收到的包的编号

此时分析场景可以对应着blaster第一行，可以看到收到了96号包的ACk，分析第一张图的时候刚好是发出96号包，那么这个时候LHS指示97，代表97之前的都已经Ack了，符合设计逻辑。RHS指示的是98，对应的是发出了98号包，此时的Seq指向的99，也符合设计逻辑。

对应到Blastee就是倒数的第五行，此时收到了98号包

随后看到Blaster发出99号包，RHS和Seq后移，这个时候RHS-LHS+1=99-97+1=4，还可以发。

继续发出100，随后100被Ack，这个时候处理最后没有发完的97号包，触发了超时，一直发97号包一直到最后Ack结束

test2

pac num=100

drop rate=0.19

timeout val=1s=1000ms

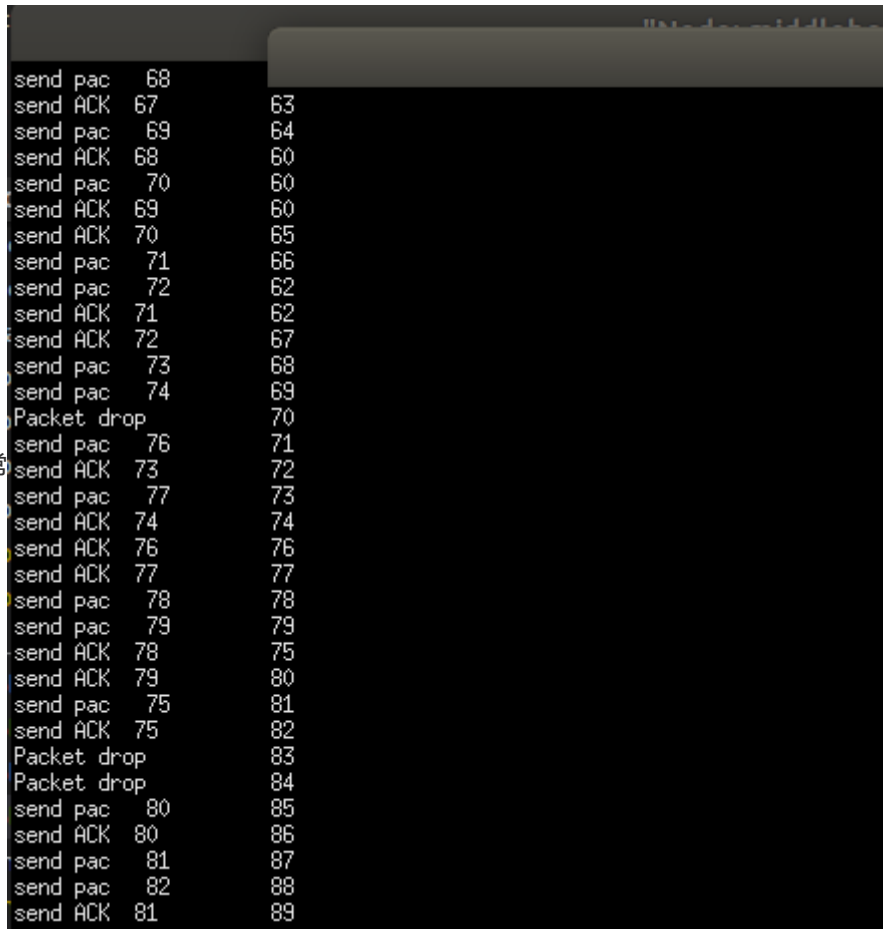
以如下为例子说明超时机制

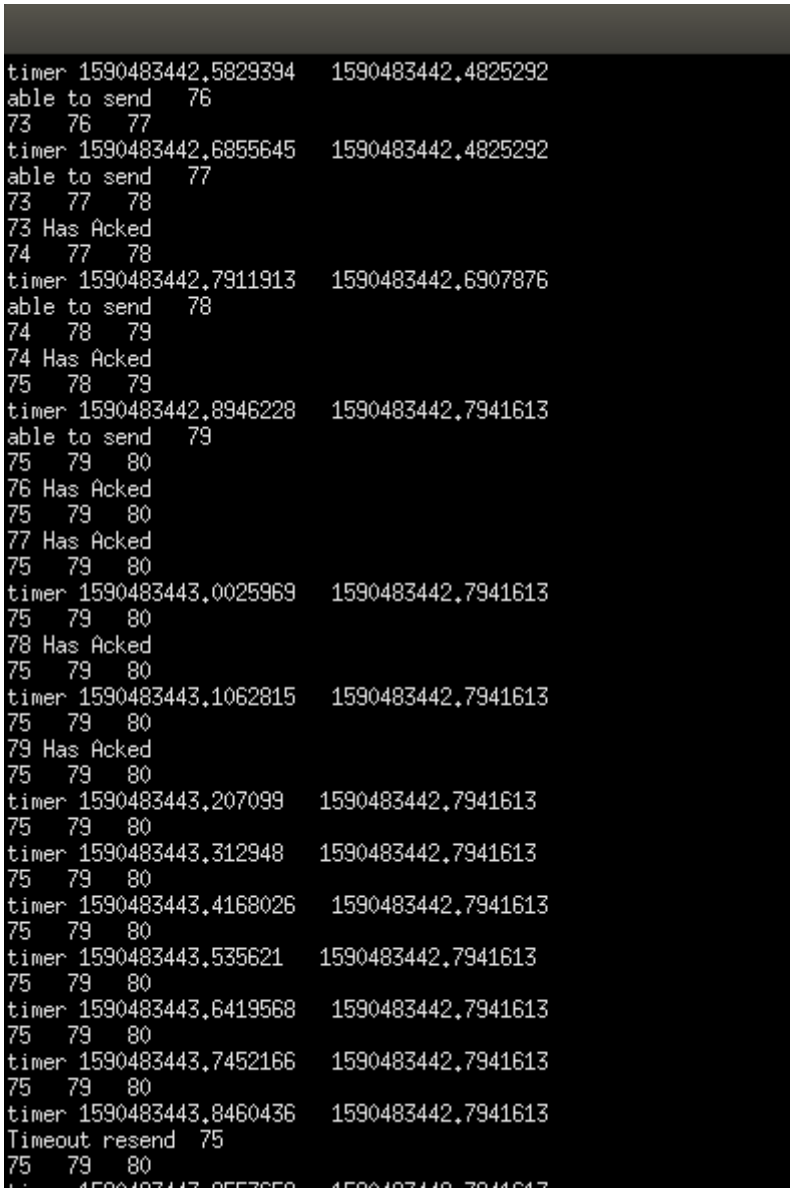
可以发现最开始的时候是发出76号包，这个时候LHS=73，也就是已经Ack了72以及之前的，RHS=76，对应的上述刚刚发出了76号，此时可以继续发，于是发出77，这个时候收到了73的Ack，于是RHS后移，LHS后移。可以发包，继续发出78号。收到了74的Ack。一收一发还可以发，发出79号，这个时候发现75号一直没有Ack

又有79-75+1=5，于是不再发包（再发就超过5了）

等候不多时触发了timeout，在这个期间收到了一直到79的Ack，触发之后重发没有Ack的75号。

程序恢复正常

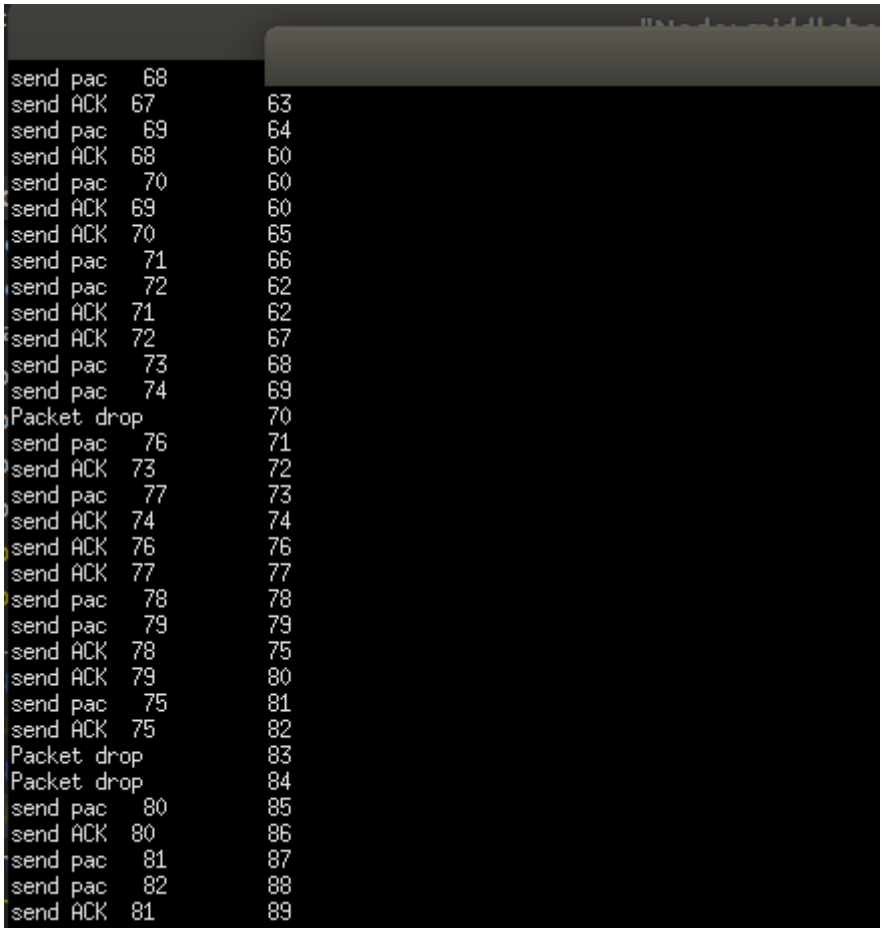




结合着Blastee和MiddleBox可以综和的校验

在MiddleBox里有74和76之间的Drop，以及在最后的后半部分的sendpac 75和send Ack75

在Blastee里有74和76之间的中断，75在后面才收到，收到75之后程序正常的工作

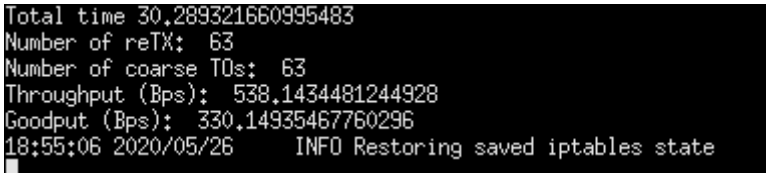


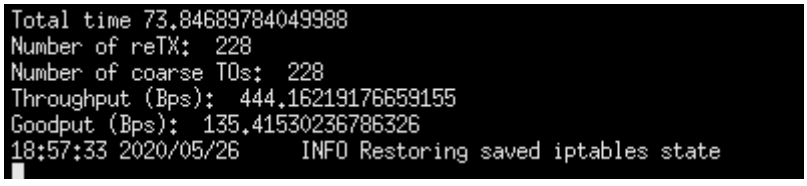
test3

参数对比

调大丢包率前后的测试结果

非常明显的超时次数增加以及带宽减小（网络不稳定造成的时延和带宽减小）





Wireshark

抓包结果分析

这个是在Blastee设置的抓包

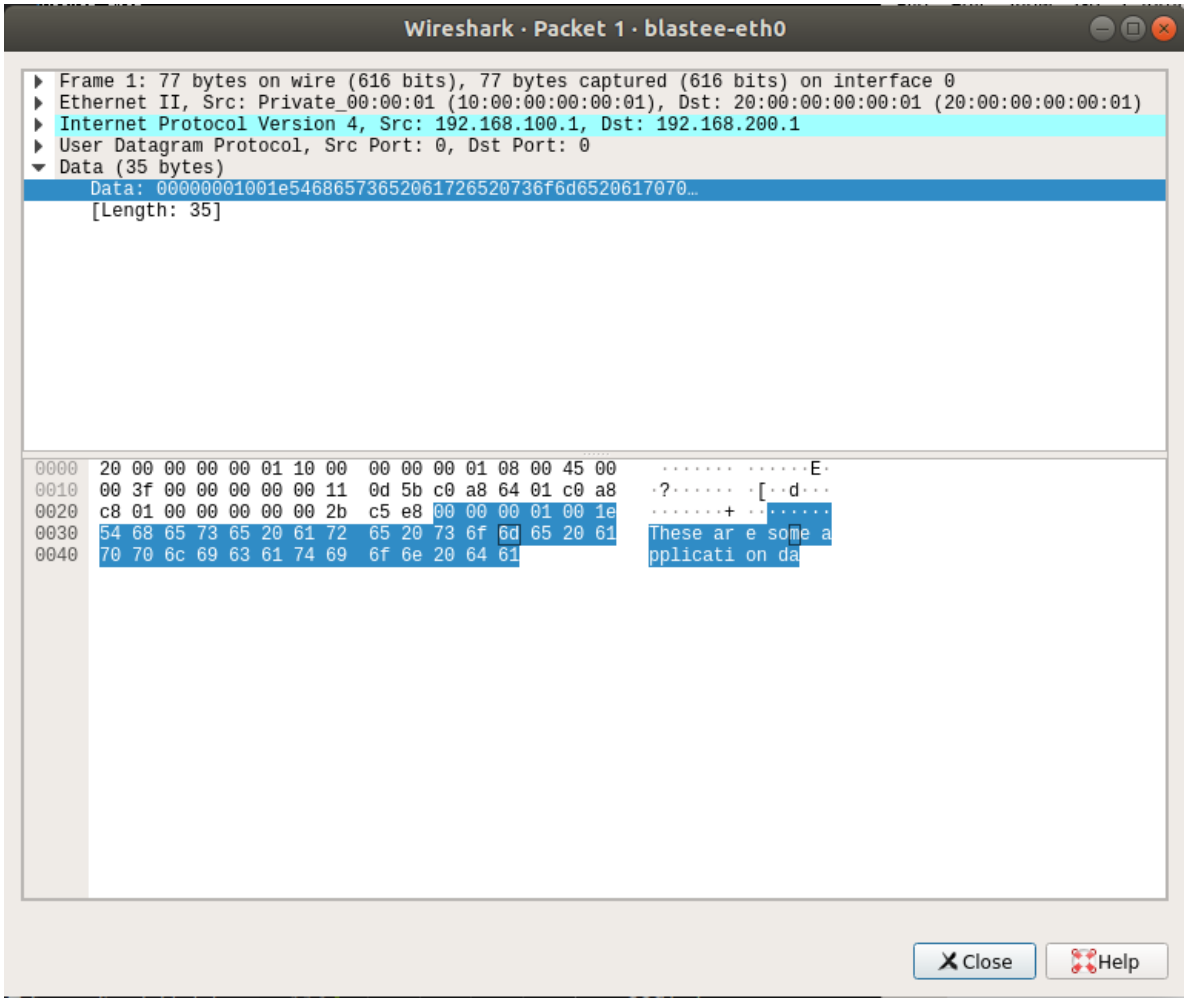
发现了Blaster发来的包，对照地址无误

看数据包里，可以得到前面的四个字节是Seq，就是1

然后是length，1e对应16进制的16+14=30，也就是预期设置的

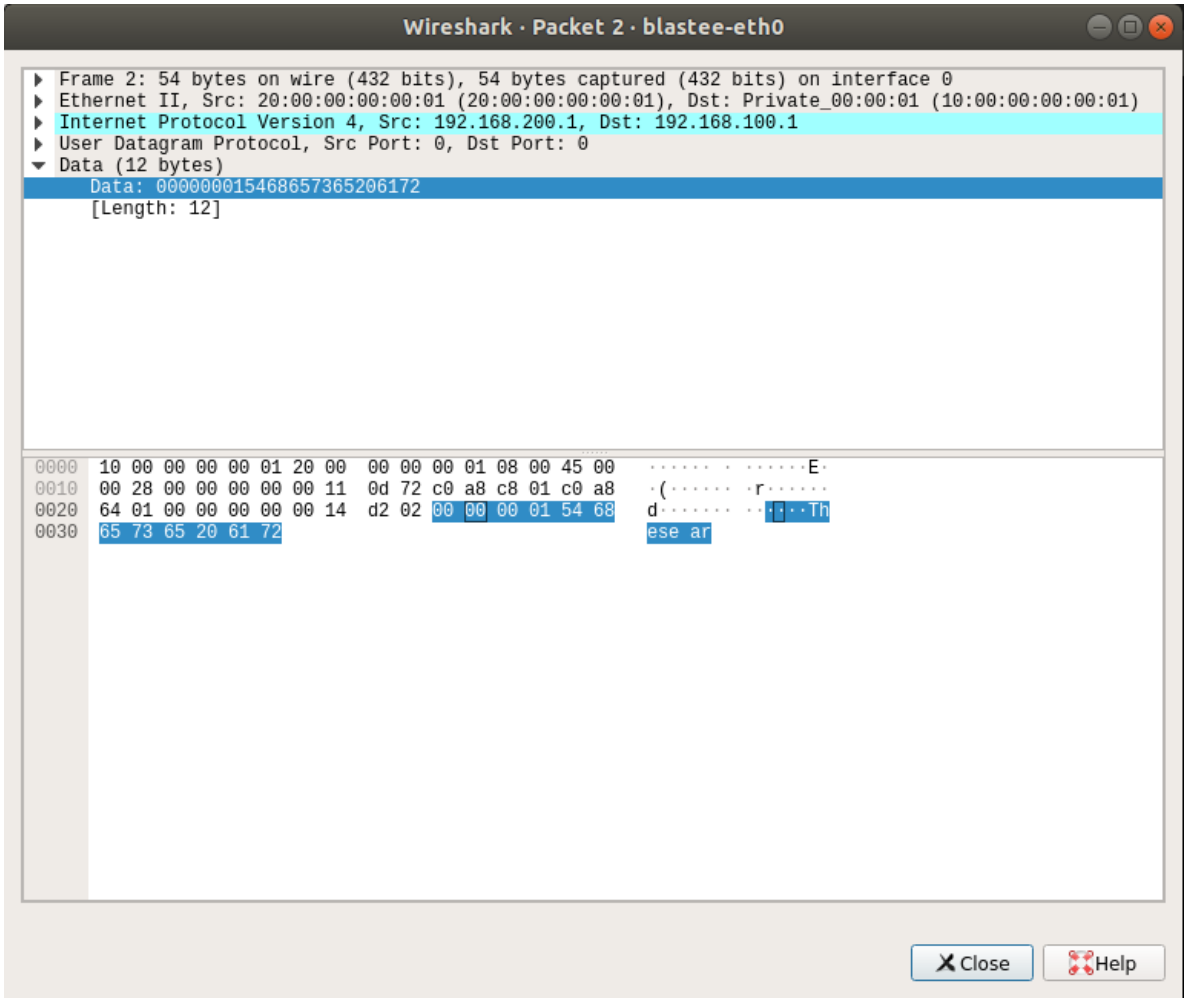
最后的是payload，一串字符串，同时根据30的长度进行了截断

上述部分一共是35字节



同理可以看到对于发出的Ack包，有着相似的结构，一样的Seq=1

同时没有了length部分，而是一个定长的payload，对应这就是一共12字节



实验感悟

把上课学习到的滑动窗口进行了实践，觉得很有意思

也对网络质量的影响因素有了直观了解