

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]
    sequencenumber = 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(sequencenumber.to_bytes(4,byteorder = 'big'))
    pay = RawPacketContents(payload)
    e = Ethernet(src = mymac,dst = blastermac)
    ip = IPv4(src = myip,dst = blasterip ,protocol = IPPROTO_UDP,ttl = 8)
    udp = UDP(src = 2,dst = 1)
    sendpkt = e + ip + udp + seq + pay
    net.send_packet(dev, sendpkt)

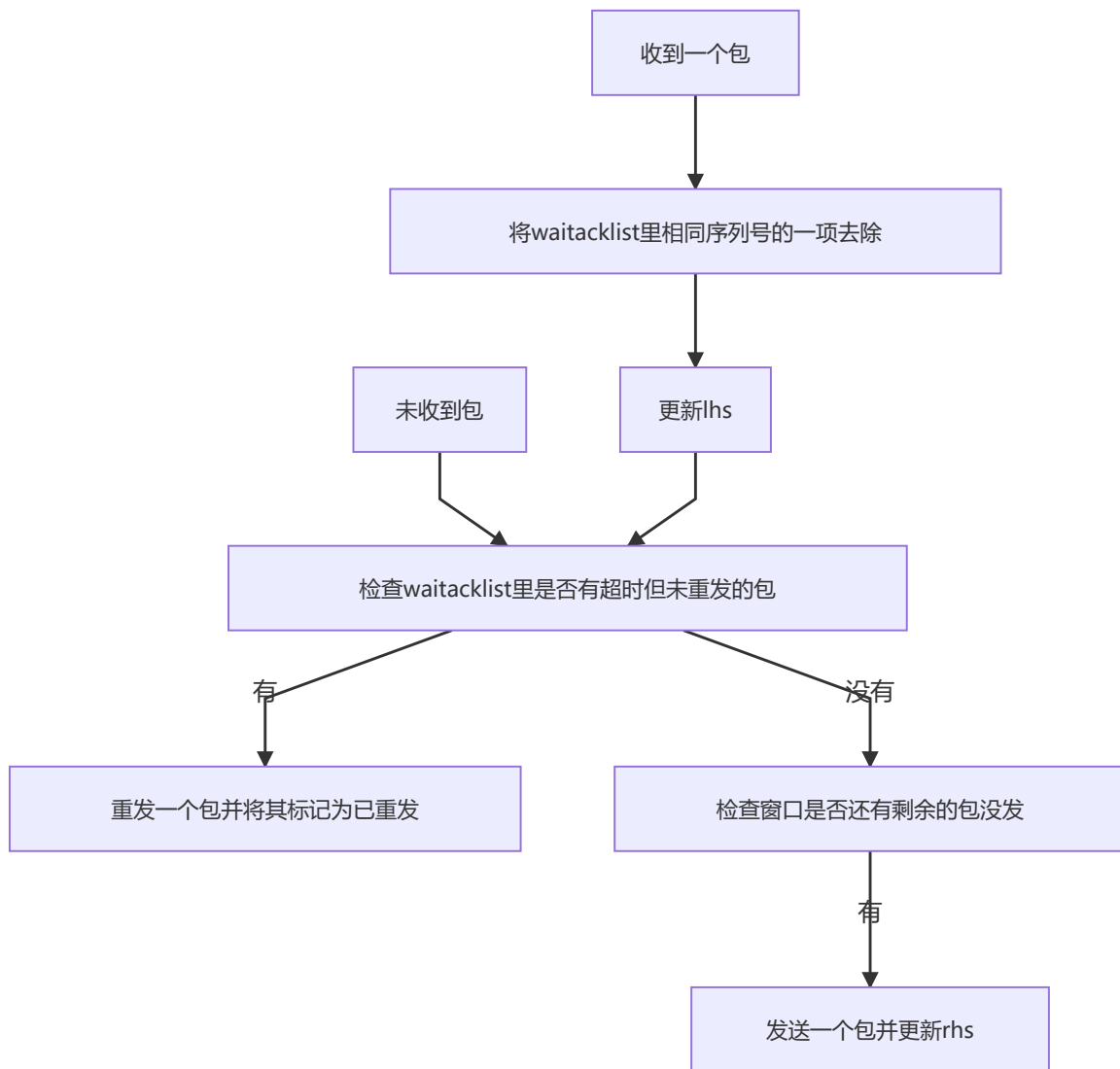
```

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'
    blasteaip = '192.168.200.1'
    blasterip = '192.168.100.1'

    pkt = Ethernet(src = blastermac,dst = blasteemac , etherType = EtherType.IPv4) + IPv4(src = blasterip, dst = blasteaip)
    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]
sequencenumber = int.from_bytes(temp1, byteorder = 'big')
for index in waitacklist:
    if index.seq == sequencenumber:
        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
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：

```
"Node: blaster"

root@njucs-VirtualBox:~/switchyard/lab_6# source ../syenv/bin/activate
(syenv) root@njucs-VirtualBox:~/switchyard/lab_6# ../switchyard/swyard.py blaster.py
00:31:26 2020/05/19 INFO Saving iptables state and installing switchyard rules
00:31:26 2020/05/19 INFO Using network devices: blaster-eth0
00:31:44 2020/05/19 INFO total tx time:17.979745626449585

00:31:44 2020/05/19 INFO number of retx:70

00:31:44 2020/05/19 INFO number of coarse tos:31

00:31:44 2020/05/19 INFO throughput: 945.5083710968466

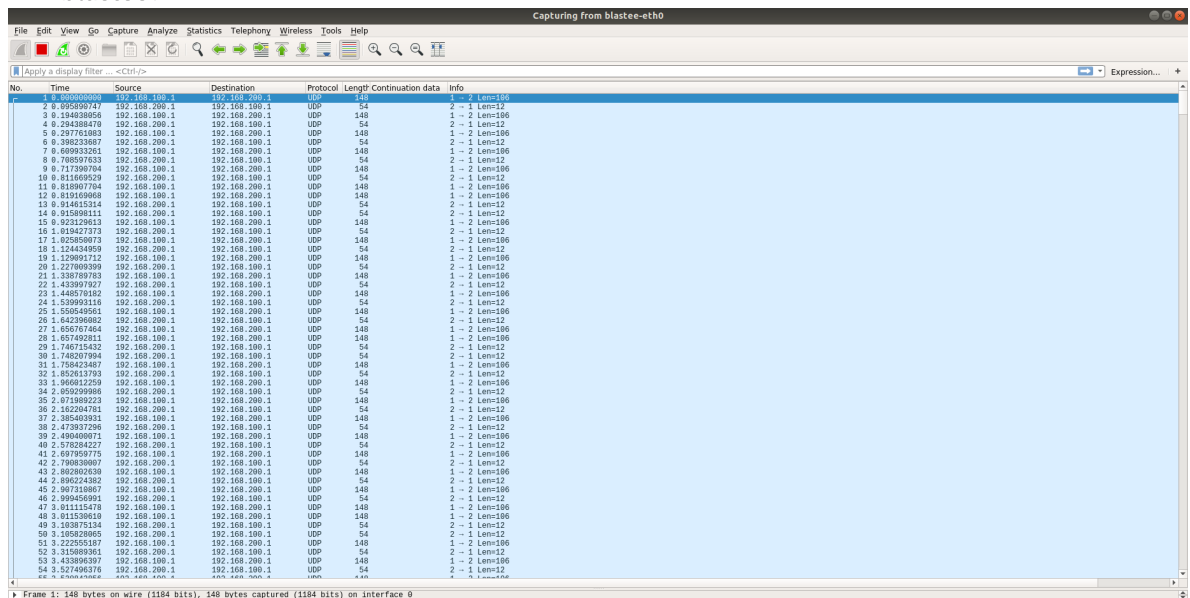
00:31:44 2020/05/19 INFO goodput: 556.181394762851

00:31:44 2020/05/19 INFO Restoring saved iptables state

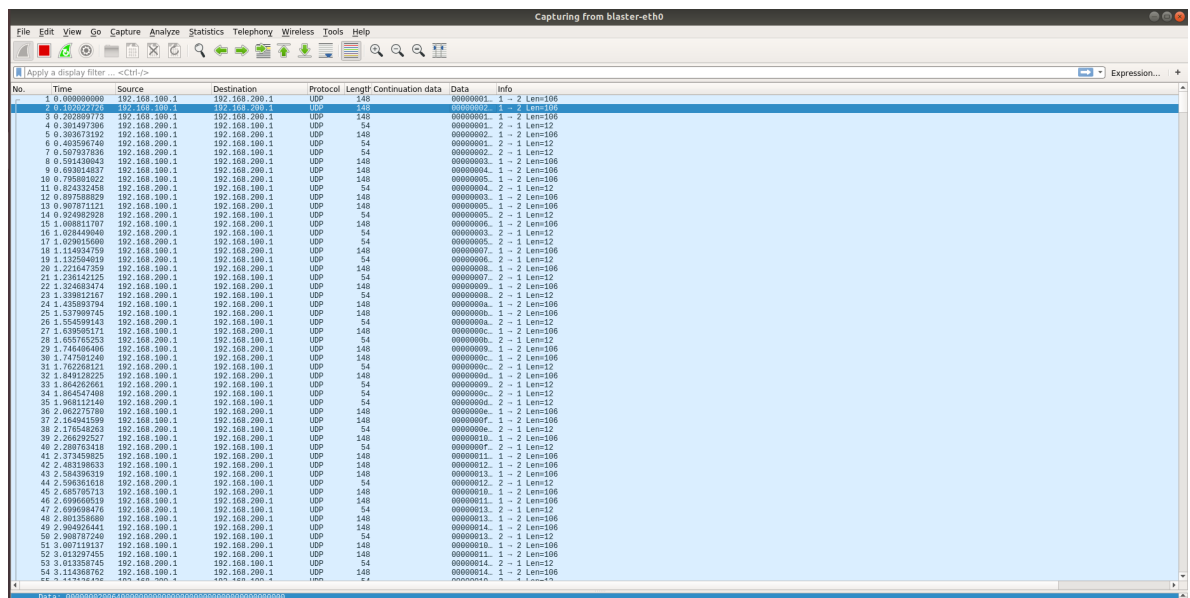
(syenv) root@njucs-VirtualBox:~/switchyard/lab_6#
```

用wireshark抓包得到的结果如下:

blasteer:



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：

```
"Node: blaster"
root@njucs-VirtualBox:~/switchyard/lab_6# source ../syenv/bin/activate
(syenv) root@njucs-VirtualBox:~/switchyard/lab_6# ./../switchyard/swyard.py bla
ster.py
15:41:18 2020/05/19      INFO Saving iptables state and installing switchyard rul
es
15:41:18 2020/05/19      INFO Using network devices: blaster-eth0
15:41:43 2020/05/19      INFO total tx time:24.921489477157593

15:41:43 2020/05/19      INFO number of retx:21

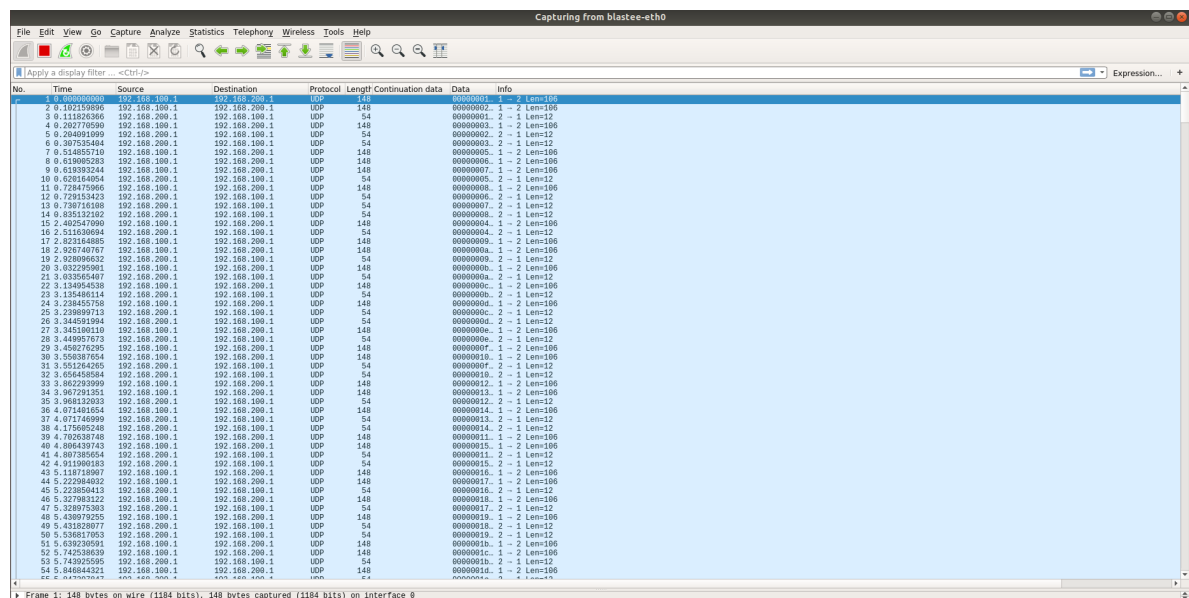
15:41:43 2020/05/19      INFO number of coarse tos:14

15:41:43 2020/05/19      INFO throughput: 485.5247520855667

15:41:43 2020/05/19      INFO goodput: 401.26012569055104

15:41:43 2020/05/19      INFO Restoring saved iptables state
(syenv) root@njucs-VirtualBox:~/switchyard/lab_6#
```

blastee:



blaster:

Capturing from blaster-eth0

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter... <Ctrl>F

No.	Time	Source	Destination	Protocol	Length	Continuation	Data	Info
1	0.000000000	192.168.100.1	192.168.200.1	UDP	148		00000001 1 - 2 Len=100	
2	0.010091909	192.168.100.1	192.168.200.1	UDP	148		00000002 1 - 2 Len=100	
3	0.202251814	192.168.100.1	192.168.200.1	UDP	148		00000003 1 - 2 Len=100	
4	0.272948706	192.168.100.1	192.168.200.1	UDP	148		00000004 1 - 2 Len=100	
5	0.300700771	192.168.100.1	192.168.200.1	UDP	148		00000005 1 - 2 Len=100	
6	0.375752613	192.168.200.1	192.168.100.1	UDP	54		00000002 2 - 1 Len=12	
7	0.482490237	192.168.200.1	192.168.100.1	UDP	54		00000003 2 - 1 Len=12	
8	0.481619418	192.168.100.1	192.168.200.1	UDP	148		00000005 1 - 2 Len=100	
9	0.505359708	192.168.100.1	192.168.200.1	UDP	148		00000006 1 - 2 Len=100	
10	0.685376357	192.168.100.1	192.168.200.1	UDP	148		00000007 1 - 2 Len=100	
11	0.707165727	192.168.100.1	192.168.200.1	UDP	148		00000008 1 - 2 Len=100	
12	0.796763910	192.168.200.1	192.168.100.1	UDP	54		00000005 2 - 1 Len=12	
13	0.903474277	192.168.200.1	192.168.100.1	UDP	54		00000006 2 - 1 Len=12	
14	0.903614479	192.168.200.1	192.168.100.1	UDP	54		00000007 2 - 1 Len=12	
15	1.007632564	192.168.200.1	192.168.100.1	UDP	54		00000008 2 - 1 Len=12	
16	1.514717086	192.168.100.1	192.168.200.1	UDP	148		00000004 1 - 2 Len=100	
17	2.419336724	192.168.100.1	192.168.200.1	UDP	148		00000004 1 - 2 Len=100	
18	2.685544553	192.168.100.1	192.168.200.1	UDP	148		00000004 1 - 2 Len=100	
19	2.888688091	192.168.100.1	192.168.200.1	UDP	148		00000005 1 - 2 Len=100	
20	2.890921843	192.168.100.1	192.168.200.1	UDP	148		00000006 1 - 2 Len=100	
21	3.090937104	192.168.100.1	192.168.200.1	UDP	148		00000006 1 - 2 Len=100	
22	3.102029605	192.168.200.1	192.168.100.1	UDP	54		00000009 2 - 1 Len=12	
23	3.192228516	192.168.100.1	192.168.200.1	UDP	148		00000006 1 - 2 Len=100	
24	3.203648801	192.168.200.1	192.168.100.1	UDP	54		00000009 2 - 1 Len=12	
25	3.384096031	192.168.100.1	192.168.200.1	UDP	148		00000006 1 - 2 Len=100	
26	3.397949386	192.168.200.1	192.168.100.1	UDP	54		00000009 2 - 1 Len=12	
27	3.408472388	192.168.100.1	192.168.200.1	UDP	148		00000006 1 - 2 Len=100	
28	3.414626452	192.168.200.1	192.168.100.1	UDP	54		00000009 2 - 1 Len=12	
29	3.515026252	192.168.100.1	192.168.200.1	UDP	148		00000007 1 - 2 Len=100	
30	3.519028893	192.168.200.1	192.168.100.1	UDP	54		00000009 2 - 1 Len=12	
31	3.618048824	192.168.100.1	192.168.200.1	UDP	148		00000010 1 - 2 Len=100	
32	3.619902119	192.168.200.1	192.168.100.1	UDP	54		00000009 2 - 1 Len=12	
33	3.724471484	192.168.200.1	192.168.100.1	UDP	54		00000007 2 - 1 Len=12	
34	3.728077037	192.168.100.1	192.168.200.1	UDP	148		00000011 1 - 2 Len=100	
35	3.828395748	192.168.100.1	192.168.200.1	UDP	148		00000012 1 - 2 Len=12	
36	3.828680291	192.168.200.1	192.168.100.1	UDP	54		00000012 1 - 2 Len=100	
37	3.938021266	192.168.100.1	192.168.200.1	UDP	148		00000013 1 - 2 Len=100	
38	4.039228249	192.168.100.1	192.168.200.1	UDP	148		00000014 1 - 2 Len=100	
39	4.140880388	192.168.100.1	192.168.200.1	UDP	148		00000015 1 - 2 Len=100	
40	4.148841098	192.168.200.1	192.168.100.1	UDP	54		00000012 2 - 1 Len=12	
41	4.243717476	192.168.100.1	192.168.200.1	UDP	148		00000015 2 - 1 Len=12	
42	4.352435767	192.168.200.1	192.168.100.1	UDP	54		00000014 2 - 1 Len=12	
43	4.768184227	192.168.100.1	192.168.200.1	UDP	148		00000015 1 - 2 Len=100	
44	4.868884709	192.168.100.1	192.168.200.1	UDP	148		00000015 1 - 2 Len=100	
45	4.978713971	192.168.200.1	192.168.100.1	UDP	54		00000015 2 - 1 Len=100	
46	5.083974888	192.168.200.1	192.168.100.1	UDP	54		00000015 2 - 1 Len=12	
47	5.090605185	192.168.100.1	192.168.200.1	UDP	148		00000016 1 - 2 Len=100	
48	5.194132152	192.168.100.1	192.168.200.1	UDP	148		00000017 1 - 2 Len=100	
49	5.295755738	192.168.100.1	192.168.200.1	UDP	148		00000018 1 - 2 Len=100	
50	5.397237038	192.168.100.1	192.168.200.1	UDP	148		00000019 1 - 2 Len=100	
51	5.491863031	192.168.200.1	192.168.100.1	UDP	54		00000016 2 - 1 Len=12	
52	5.508499410	192.168.100.1	192.168.200.1	UDP	148		00000017 2 - 1 Len=12	
53	5.595699052	192.168.100.1	192.168.200.1	UDP	148		00000018 1 - 2 Len=100	
54	5.604471016	192.168.200.1	192.168.100.1	UDP	54		00000018 2 - 1 Len=12	
55	5.604471016	192.168.200.1	192.168.100.1	UDP	54		00000018 2 - 1 Len=12	

Frame 1: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface 0

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blaste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：

```

"Node: blaster"
root@njucs-VirtualBox:~/switchyard/lab_6# source ../syenv/bin/activate
(syenv) root@njucs-VirtualBox:~/switchyard/lab_6# ../../switchyard/swyard.py bla
ster.py
15:44:55 2020/05/19      INFO Saving iptables state and installing switchyard rul
es
15:44:55 2020/05/19      INFO Using network devices: blaster-eth0
15:46:41 2020/05/19      INFO total tx time:105.93516707420349

15:46:41 2020/05/19      INFO number of retx:273

15:46:41 2020/05/19      INFO number of coarse tos:105

15:46:41 2020/05/19      INFO throughput: 352.1021491746248

15:46:41 2020/05/19      INFO goodput: 94.39735902804955

15:46:41 2020/05/19      INFO Restoring saved iptables state

(syenv) root@njucs-VirtualBox:~/switchyard/lab_6#

```

blatee:

Capturing from blaster-eth0

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl>-> Expression...

No.	Time	Source	Destination	Protocol	Length	Continuation	Data	Info
1	0.000000000	192.168.100.1	192.168.200.1	UDP	148		00000001 1 - 2 Len=106	
2	0.142541749	192.168.100.1	192.168.200.1	UDP	54		00000002 1 - 2 Len=12	
3	0.038387521	192.168.100.1	192.168.200.1	UDP	148		00000002 1 - 2 Len=106	
4	0.004415030	192.168.100.1	192.168.200.1	UDP	54		00000002 1 - 2 Len=12	
5	0.245883837	192.168.100.1	192.168.200.1	UDP	148		00000004 1 - 2 Len=106	
6	0.296509098	192.168.200.1	192.168.100.1	UDP	54		00000004 2 - 1 Len=12	
7	0.185786579	192.168.100.1	192.168.200.1	UDP	148		00000005 1 - 2 Len=106	
8	0.234283117	192.168.200.1	192.168.100.1	UDP	54		00000005 2 - 1 Len=12	
9	0.754020121	192.168.100.1	192.168.200.1	UDP	148		00000005 1 - 2 Len=106	
10	0.898489756	192.168.200.1	192.168.100.1	UDP	54		00000006 2 - 1 Len=12	
11	0.629500330	192.168.100.1	192.168.200.1	UDP	148		00000006 1 - 2 Len=106	
12	0.727941663	192.168.200.1	192.168.100.1	UDP	54		00000006 2 - 1 Len=12	
13	0.047646448	192.168.100.1	192.168.200.1	UDP	148		00000006 1 - 2 Len=106	
14	0.142439930	192.168.200.1	192.168.100.1	UDP	54		00000006 2 - 1 Len=12	
15	0.090258886	192.168.100.1	192.168.200.1	UDP	148		00000006 1 - 2 Len=106	
16	0.494361885	192.168.100.1	192.168.200.1	UDP	54		00000006 1 - 2 Len=12	
17	0.758420842	192.168.100.1	192.168.200.1	UDP	148		00000006 1 - 2 Len=106	
18	0.762318311	192.168.200.1	192.168.100.1	UDP	54		00000006 2 - 1 Len=12	
19	0.168181495	192.168.100.1	192.168.200.1	UDP	148		00000007 1 - 2 Len=106	
20	0.170807918	192.168.100.1	192.168.200.1	UDP	54		00000007 2 - 1 Len=12	
21	0.122252284	192.168.100.1	192.168.200.1	UDP	148		00000007 1 - 2 Len=106	
22	0.226618822	192.168.200.1	192.168.100.1	UDP	54		00000007 2 - 1 Len=12	
23	0.784544851	192.168.100.1	192.168.200.1	UDP	148		00000007 1 - 2 Len=106	
24	0.061662722	192.168.200.1	192.168.100.1	UDP	54		00000007 2 - 1 Len=12	
25	0.625206038	192.168.100.1	192.168.200.1	UDP	148		00000007 1 - 2 Len=106	
26	0.635138018	192.168.200.1	192.168.100.1	UDP	54		00000007 2 - 1 Len=12	
27	0.308755108	192.168.100.1	192.168.200.1	UDP	148		00000007 1 - 2 Len=106	
28	0.947390084	192.168.200.1	192.168.100.1	UDP	54		00000007 2 - 1 Len=12	
29	0.718400991	192.168.100.1	192.168.200.1	UDP	148		00000007 1 - 2 Len=106	
30	0.886813528	192.168.200.1	192.168.100.1	UDP	54		00000010 2 - 1 Len=12	
31	0.898411159	192.168.100.1	192.168.200.1	UDP	148		00000011 1 - 2 Len=106	
32	0.995811993	192.168.200.1	192.168.100.1	UDP	54		00000011 2 - 1 Len=12	
33	0.614539325	192.168.100.1	192.168.200.1	UDP	148		00000006 1 - 2 Len=106	
34	0.718819509	192.168.100.1	192.168.200.1	UDP	54		00000006 2 - 1 Len=12	
35	0.938126407	192.168.100.1	192.168.200.1	UDP	148		00000013 1 - 2 Len=106	
36	0.839204478	192.168.100.1	192.168.200.1	UDP	148		00000014 1 - 2 Len=106	
37	0.604375874	192.168.200.1	192.168.100.1	UDP	54		00000013 2 - 1 Len=12	
38	0.339621121	192.168.100.1	192.168.200.1	UDP	148		00000014 1 - 2 Len=106	
39	0.870909276	192.168.100.1	192.168.200.1	UDP	148		00000012 1 - 2 Len=106	
40	0.870716108	192.168.100.1	192.168.200.1	UDP	148		00000015 1 - 2 Len=106	
41	0.874522208	192.168.200.1	192.168.100.1	UDP	54		00000012 2 - 1 Len=12	
42	0.878282421	192.168.200.1	192.168.100.1	UDP	54		00000015 2 - 1 Len=12	
43	0.288282707	192.168.100.1	192.168.200.1	UDP	148		00000016 1 - 2 Len=106	
44	0.290911884	192.168.200.1	192.168.100.1	UDP	54		00000016 2 - 1 Len=12	
45	0.384505072	192.168.100.1	192.168.200.1	UDP	148		00000016 1 - 2 Len=106	
46	0.388919729	192.168.200.1	192.168.100.1	UDP	54		00000016 2 - 1 Len=12	
47	0.014312084	192.168.100.1	192.168.200.1	UDP	148		00000016 1 - 2 Len=106	
48	0.090663785	192.168.200.1	192.168.100.1	UDP	54		00000014 2 - 1 Len=12	
49	0.036637182	192.168.100.1	192.168.200.1	UDP	148		00000014 1 - 2 Len=106	
50	0.922464358	192.168.200.1	192.168.100.1	UDP	54		00000016 2 - 1 Len=12	
51	0.081467607	192.168.100.1	192.168.200.1	UDP	148		00000016 1 - 2 Len=106	
52	0.013889661	192.168.200.1	192.168.100.1	UDP	54		00000016 2 - 1 Len=12	
53	0.576771715	192.168.100.1	192.168.200.1	UDP	148		00000017 1 - 2 Len=106	
54	0.650922248	192.168.200.1	192.168.100.1	UDP	54		00000017 2 - 1 Len=12	

blaster:

Capturing from blaster-eth0

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl>-> Expression...

No.	Time	Source	Destination	Protocol	Length	Continuation	Data	Info
1	0.000000000	192.168.100.1	192.168.200.1	UDP	148		00000001 1 - 2 Len=106	
2	0.191674588	192.168.100.1	192.168.200.1	UDP	148		00000002 1 - 2 Len=106	
3	0.160280055	192.168.200.1	192.168.100.1	UDP	54		00000002 2 - 1 Len=12	
4	0.282788711	192.168.100.1	192.168.200.1	UDP	148		00000003 1 - 2 Len=106	
5	0.384044953	192.168.100.1	192.168.200.1	UDP	148		00000003 1 - 2 Len=106	
6	0.485321738	192.168.100.1	192.168.200.1	UDP	148		00000005 1 - 2 Len=106	
7	0.588406741	192.168.100.1	192.168.200.1	UDP	148		00000005 1 - 2 Len=106	
8	0.089772243	192.168.100.1	192.168.200.1	UDP	148		00000002 1 - 2 Len=106	
9	0.130496402	192.168.100.1	192.168.200.1	UDP	148		00000003 1 - 2 Len=106	
10	0.200991321	192.168.200.1	192.168.100.1	UDP	54		00000002 2 - 1 Len=12	
11	0.207700791	192.168.100.1	192.168.200.1	UDP	148		00000004 1 - 2 Len=106	
12	0.308113773	192.168.100.1	192.168.200.1	UDP	148		00000005 1 - 2 Len=106	
13	0.408625748	192.168.100.1	192.168.200.1	UDP	148		00000006 1 - 2 Len=106	
14	0.409332871	192.168.100.1	192.168.200.1	UDP	148		00000006 1 - 2 Len=106	
15	0.522699736	192.168.100.1	192.168.200.1	UDP	148		00000007 1 - 2 Len=106	
16	0.137152754	192.168.100.1	192.168.200.1	UDP	148		00000003 1 - 2 Len=106	
17	0.237720877	192.168.100.1	192.168.200.1	UDP	148		00000005 1 - 2 Len=106	
18	0.338878777	192.168.100.1	192.168.200.1	UDP	148		00000006 1 - 2 Len=106	
19	0.348527871	192.168.200.1	192.168.100.1	UDP	54		00000005 2 - 1 Len=12	
20	0.364472634	192.168.100.1	192.168.200.1	UDP	148		00000007 1 - 2 Len=106	
21	0.07575254	192.168.100.1	192.168.200.1	UDP	148		00000003 1 - 2 Len=106	
22	0.176954768	192.168.100.1	192.168.200.1	UDP	148		00000006 1 - 2 Len=106	
23	0.270966878	192.168.100.1	192.168.200.1	UDP	148		00000006 1 - 2 Len=106	
24	0.083333334	192.168.100.1	192.168.200.1	UDP	148		00000003 1 - 2 Len=106	
25	0.083991758	192.168.100.1	192.168.200.1	UDP	148		00000006 1 - 2 Len=106	
26	0.198328671	192.168.100.1	192.168.200.1	UDP	148		00000007 1 - 2 Len=106	
27	0.002769695	192.168.100.1	192.168.200.1	UDP	148		00000003 1 - 2 Len=106	
28	0.002546977	192.168.100.1	192.168.200.1	UDP	148		00000006 1 - 2 Len=106	
29	0.109393133	192.168.100.1	192.168.200.1	UDP	148		00000007 1 - 2 Len=106	
30	0.014086548	192.168.100.1	192.168.200.1	UDP	148		00000003 1 - 2 Len=106	
31	0.018214658	192.168.100.1	192.168.200.1	UDP	148		00000006 1 - 2 Len=106	
32	0.021168095	192.168.100.1	192.168.200.1	UDP	148		00000007 1 - 2 Len=106	
33	0.734616014	192.168.100.1	192.168.200.1	UDP	148		00000003 1 - 2 Len=106	
34	0.038997924	192.168.100.1	192.168.200.1	UDP	148		00000006 1 - 2 Len=106	
35	0.938059991	192.168.100.1	192.168.200.1	UDP	148		00000007 1 - 2 Len=106	
36	0.950706673	192.168.200.1	192.168.100.1	UDP	54		00000006 2 - 1 Len=12	
37	0.050550326	192.168.100.1	192.168.200.1	UDP	148		00000003 1 - 2 Len=106	
38	0.777776328	192.168.100.1	192.168.200.1	UDP	148		00000007 1 - 2 Len=106	
39	0.750501778	192.168.100.1	192.168.200.1	UDP	148		00000003 2 - 1 Len=12	
40	0.991861849	192.168.100.1	192.168.200.1	UDP	148		00000008 1 - 2 Len=106	
41	0.092791392	192.168.100.1	192.168.200.1	UDP	148		00000005 1 - 2 Len=106	
42	0.195375274	192.168.100.1	192.168.200.1	UDP	148		00000006 1 - 2 Len=106	
43	0.206364695	192.168.200.1	192.168.100.1	UDP	54		00000006 2 - 1 Len=12	
44	0.380347276	192.168.100.1	192.168.200.1	UDP	148		00000006 1 - 2 Len=106	
45	0.820652475	192.168.100.1	192.168.200.1	UDP	148		00000007 1 - 2 Len=106	
46	0.927467989	192.168.100.1	192.168.200.1	UDP	148		00000008 1 - 2 Len=106	
47	0.928552395	192.168.100.1	192.168.200.1	UDP	148		00000008 1 - 2 Len=106	
48	0.134632087	192.168.100.1	192.168.200.1	UDP	148		00000008 1 - 2 Len=106	
49	0.252748649	192.168.200.1	192.168.100.1	UDP	54		00000008 2 - 1 Len=12	
50	0.702580523	192.168.100.1	192.168.200.1	UDP	148		00000007 1 - 2 Len=106	
51	0.862911899	192.168.100.1	192.168.200.1	UDP	148		00000008 1 - 2 Len=106	
52	0.963707878	192.168.100.1	192.168.200.1	UDP	148		00000008 1 - 2 Len=106	
53	0.688890808	192.168.100.1	192.168.200.1	UDP	148		00000007 1 - 2 Len=106	
54	0.789928855	192.168.100.1	192.168.200.1	UDP	148		00000008 1 - 2 Len=106	

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中把数据包序列号显示出来，以便验证是否正确
- 本次实验让我深刻