南京大学本科生实验报告

课程名称：**计算机网络**　　任课教师：田臣/李文中　　助教：lzh、lsp、wcx

学院	**计算机科学与技术系**	专业（方向）	计算机科学与技术系
学号	211220027	姓名	王秋博
Email	211220027@nju.edu.cn	开始/完成日期	11.23/12.7

1. 实验名称

Lab6: **Reliable Communication**

2. 实验目的

构建一个可靠通信库，实现以下功能：

1. 针对每个成功接收的数据包，在 blastee 上执行 ACK 机制
2. 冲击波上的固定尺寸滑动窗口。
3. 冲击波上的粗略超时，用于重新发送非 ACK 数据包

3. 实验内容

Task 2：**MiddleBox**

Step 1: Coding

middlebox 的逻辑如图：

```
def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
    _, fromIface, packet = recv
    if fromIface == "middlebox-eth0":
        log_debug("Received from blaster")
        '''
        Received data packet
        Should I drop it?
        If not, modify headers & send to blastee
        '''
        if float(randint(1,100))/100>self.dropRate:
            eth_idx=packet.get_header_index(Ethernet)
            packet[eth_idx].src=intf1_mac
            packet[eth_idx].dst=blastee_mac
            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)
        '''
        eth_idx=packet.get_header_index(Ethernet)
        packet[eth_idx].src=intf0_mac
        packet[eth_idx].dst=blaster_mac
        self.net.send_packet("middlebox-eth0", packet)
    else:
        log_debug("Oops :))")
```

重新构建表头并从另一端强制转发，对于来自 blastee 的非 ACK 加入丢弃机制，其实也就是加入随机数，随机不转发

Task 3：**Blastee**

Step 1: Coding

Blastee 的逻辑如下图：

```
    self.net = net
    # TODO: store the parameters
    ...
    self.recv_pac=[]
    self.count=0
    self.blasterIp=blasterIp
    self.num=int(num)
```

其中 recv_pac 和 count 分别用于存储和计数收到的独特序列号数据包，当 count==num 时表示 blastee 的逻辑结束

```
ack=Packet()
ether=Ethernet()
ether.ethertype=EtherType.IP
ether.src=blastee_mac
ether.dst=intf1_mac

ip=IPv4()
ip.src=self.net.interfaces()[0].ipaddr
ip.dst=self.blasterIp
ip.protocol=IPProtocol.UDP
ip.ttl=32

udp=UDP()
udp.src=4444
udp.dst=514
ack=ether+ip+udp

seq=packet[3].to_bytes()[:4]
ack+=RawPacketContents(seq)
seq_num=int.from_bytes(packet[3].to_bytes()[:4],'big')
if seq_num not in self.recv_pac:
    self.recv_pac.append(seq_num)
    self.count+=1
pl_len=int.from_bytes(packet[3].to_bytes()[4:6],'big')
if pl_len>=8:
    pl=packet[3].to_bytes()[6:14]
else:
    pl=packet[3].to_bytes()[6:]+(0).to_bytes(8-pl_len,'big')
assert(len(seq+pl)==12)
```

上图为构建 ack 过程，重点在复制 pkt 的序列号和负载长度

其中我给 udp 的 dst 赋值为 514 是由于这样抓包时 protocol 为 syslog 可以更清晰的显示序列号

Task 4：**Blaster**

Step 1: Coding

Blaster 的逻辑如下图：

```
        self.net = net
        # TODO: store the parameters
        ...
        self.blasteeIp=blasteeIp
        self.num=int(num)
        self.senderWindow=int(senderWindow)
        self.LHS=1
        #self.RHS=self.LHS+self.senderWindow-1
        self.RHS=0
        self.recv_ack=[]
        self.wait_pktseq=[]
        self.ackcount=0
        self.sended_pkt=[]
        self.length=int(length)
        self.timeout=float(timeout)/1000
        self.recvTimeout=float(recvTimeout)/1000
        self.sended_bytes=0
        self.start_time=0
        self.end_time=0
        self.timeout_count=0
        self.resend_count=0
        self.timer=time.time()
```

其中 recv_ack 列表判断是否收到 ack；

Wait_pktseq 存储准备发的数据包序列号

Sended_pkt 用来判断是否为重传

剩余大概均为手册中所提或计数量

```
def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
    _, fromIface, packet = recv
    log_debug("I got a packet")
    if packet.has_header(Ethernet) == False or packet.has_header(IPv4) == False or pack
        return
    ack_seq_num=int.from_bytes(packet[3].to_bytes()[:4],'big')
    if ack_seq_num not in self.recv_ack:
        self.recv_ack.append(ack_seq_num)
        self.ackcount+=1
    if self.ackcount==self.num:
        self.print_info()
        self.shutdown()

    for i in self.wait_pktseq[:]:
        if i==ack_seq_num:
            self.wait_pktseq.remove(i)
```

Handle_pkt 即为收到 ack，此时将队列中准备重发的序列号相同数据包删除

同时计数，若 count==num 则说明全部接收 ack，程序结束

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

    while(self.LHS in self.recv_ack and self.LHS<=self.RHS):
        self.LHS+=1
        self.timer=time.time()

    tmp=self.RHS
    self.RHS=min(self.LHS+self.senderWindow-1,self.num)
    for i in range(tmp+1,self.RHS+1):
        assert i not in self.recv_ack
        self.wait_pktseq.append(i)

    if(time.time()-self.timer>self.timeout):
        self.timeout_count+=1
        self.timer=time.time()
        for i in range(self.LHS,self.RHS+1):
            if i in self.recv_ack:
                continue
            else:
                self.wait_pktseq.append(i)

    if len(self.wait_pktseq)==0:
        return
```

Handle_no_packet 部分首先更新窗口，若窗口出现改变则更新计时器，优先发新窗口的包而不是重传，可以使抓包重传的概率降低，接近丢包率

```
pkt = Ethernet() + IPv4() + UDP()
pkt[1].protocol = IPProtocol.UDP
pkt[0].src=blaster_mac
pkt[0].dst=intf0_mac
pkt[0].ethertype=EtherType.IPv4
pkt[1].src=self.net.interfaces()[0].ipaddr
pkt[1].dst=self.blasteeIp
pkt[1].ttl=32
pkt[2].src=514
pkt[2].dst=4444
# Do other things here and send packet
...
new_seqnum=self.wait_pktseq.pop(0)
if new_seqnum in self.sended_pkt:
    self.resend_count+=1
else:
    self.sended_pkt.append(new_seqnum)
seq=RawPacketContents(new_seqnum.to_bytes(4,'big'))
pl=RawPacketContents(self.length.to_bytes(2,'big')+(0).to_bytes(self.length,'big'))
self.sended_bytes+=self.length
pkt=pkt+seq+pl
self.net.send_packet(self.net.interfaces()[0],pkt)
```

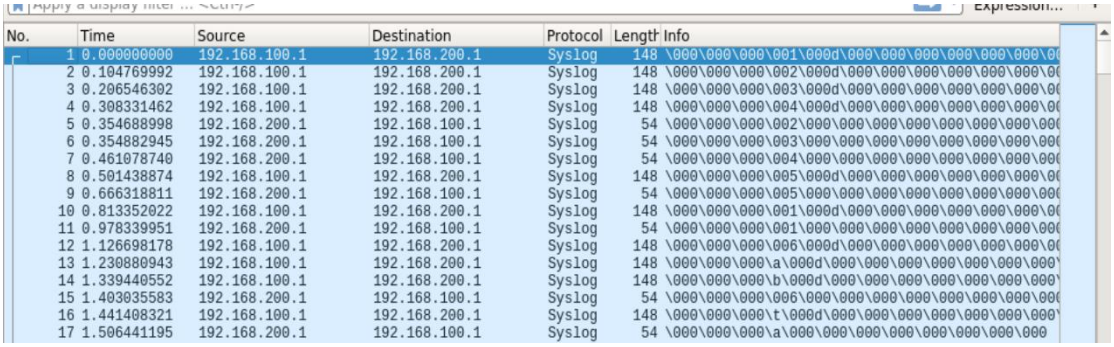
以上为构建数据包过程，与 blastee 中大致相同

Task 5：Running your code

对 Blaster 抓包：

```
01:47:38 2023/12/08     INFO Saving iptables state and installing switchyard rul
es
01:47:38 2023/12/08     INFO Using network devices: blaster-eth0
01:47:59 2023/12/08     INFO Total TX time: 20.301246881484985
01:47:59 2023/12/08     INFO Number of reTX: 27
01:47:59 2023/12/08     INFO Number of coarse TOs: 19
01:47:59 2023/12/08     INFO Throughput (Bps): 625.5773388767847
01:47:59 2023/12/08     INFO Goodput (Bps): 492.5805817927438
01:47:59 2023/12/08     INFO Restoring saved iptables state
```

（抓了好多次抓了个 19）



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.100.1	192.168.200.1	Syslog	148	\000\000\000\001\000d\000\000\000\000\000\000\00
2	0.104769992	192.168.100.1	192.168.200.1	Syslog	148	\000\000\000\002\000d\000\000\000\000\000\000\00
3	0.206546302	192.168.100.1	192.168.200.1	Syslog	148	\000\000\000\003\000d\000\000\000\000\000\000\00
4	0.308331462	192.168.100.1	192.168.200.1	Syslog	148	\000\000\000\004\000d\000\000\000\000\000\000\00
5	0.354688998	192.168.200.1	192.168.100.1	Syslog	54	\000\000\000\002\000\000\000\000\000\000\000\00
6	0.354882945	192.168.200.1	192.168.100.1	Syslog	54	\000\000\000\003\000\000\000\000\000\000\000\00
7	0.461078740	192.168.200.1	192.168.100.1	Syslog	54	\000\000\000\004\000\000\000\000\000\000\000\00
8	0.501438874	192.168.100.1	192.168.200.1	Syslog	148	\000\000\000\005\000d\000\000\000\000\000\000\00
9	0.666318811	192.168.200.1	192.168.100.1	Syslog	54	\000\000\000\005\000\000\000\000\000\000\000\00
10	0.813352022	192.168.100.1	192.168.200.1	Syslog	148	\000\000\000\001\000d\000\000\000\000\000\000\00
11	0.978339951	192.168.200.1	192.168.100.1	Syslog	54	\000\000\000\001\000\000\000\000\000\000\000\00
12	1.126698178	192.168.100.1	192.168.200.1	Syslog	148	\000\000\000\006\000d\000\000\000\000\000\000\00
13	1.230880943	192.168.100.1	192.168.200.1	Syslog	148	\000\000\000\a\000d\000\000\000\000\000\000\000
14	1.339440552	192.168.100.1	192.168.200.1	Syslog	148	\000\000\000\b\000d\000\000\000\000\000\000\000
15	1.403035583	192.168.200.1	192.168.100.1	Syslog	54	\000\000\000\006\000\000\000\000\000\000\000\00
16	1.441408321	192.168.100.1	192.168.200.1	Syslog	148	\000\000\000\t\000d\000\000\000\000\000\000\000
17	1.506441195	192.168.200.1	192.168.100.1	Syslog	54	\000\000\000\a\000\000\000\000\000\000\000\000

Info 中第四列为序列号，可以看到，LSH 没被 ack 时不会移动，且会在粗略超时时间后重传。如第 10 行序列号为 1，在 5 号已发过窗口卡住后重发

4. 实验结果

本节实验结果基本于实验过程中阐述，不再赘述

5. 核心代码

同实验结果

6. 总结与感想

实验将通过一种简单的方式模拟了现实世界的传输仍令我头痛不已，学习的时候也不免感慨现实的复杂