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：洋溢

学院	**计算机**	专业（方向）	**计算机科学与技术**
学号	**221220142**	姓名	**欧阳瑞泽**
Email	**221220142@smail.nju.edu.cn**	开始/完成日期	**2024.5.18**

Lab 6: Reliable Communication

1. 实验目的

本 Lab 实现运输层的可靠数据通信。分别完成三个部分：Middlebox，Blastee 和 Blaster。由于 Middlebox 随机丢包，需要在 Blaster 维护一个滑动窗口，对超时的数据包进行重发，Blastee 对每一个数据包发送 ACK 进行回应。

2. 实验内容

Step 1: Middlebox

Middlebox 类似于管道，实现双向的传输数据包，不过两端的主机固定，因而不需要 ARP 表或者发送 ARP 请求。初始化时硬解码 Blastee 和 Blaster 的 Mac 地址，在接收到包时生成随机数来模拟随机丢包，有一定概率丢失数据包，否则修改数据包的以太头，分别修改为 blastee 和 blaster 的 Mac 地址。然后从一端发送往另一端。

Step 2:Blastee

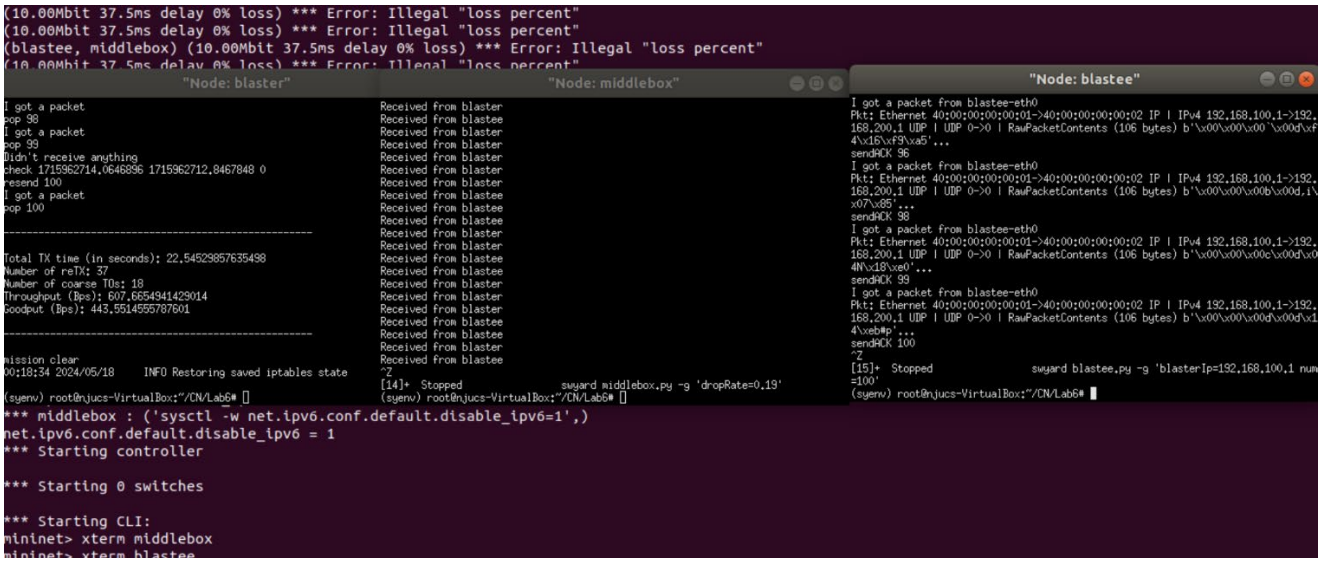
初始化时硬解码 Middlebox 的 Mac 地址，作为数据包以太头的目的地址。每收到一个合法的数据包后，生成 ACK 包并进行回应。按照手册上的格式，复制

RawPacketContext 的部分内容。Sequence number 需要使用 **big-endian** 格式，payload 部分要补足 8 个字节。

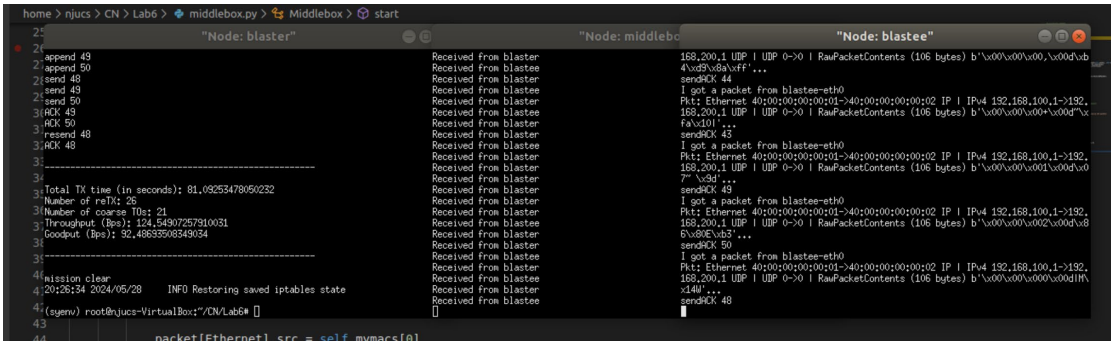
Step 3: Blaster

维护一个滑动窗口来记录已经发送但未 ACK 确认的数据包，遵守两个规则：1 滑动窗口右段减去左端需要小于等于长度限制，2 只有当左端数据包收到 ACK 确认时移动左端点到下一个未确认的数据包，在此之前若有不是最左端的数据包被 ACK 确认，需要缓存信息。若最早一次发送的数据包没有被 ACK 确认且超时过后，需要重发窗口中所有没有被 ACK 确认的数据包，然后更新最早一次发送的数据包。

3. 实验结果



发送 100 个数据包，超时时限设置为 0.5s。可见 blaster 经过多次重发后，发送完 100 个数据包，输出统计结果。

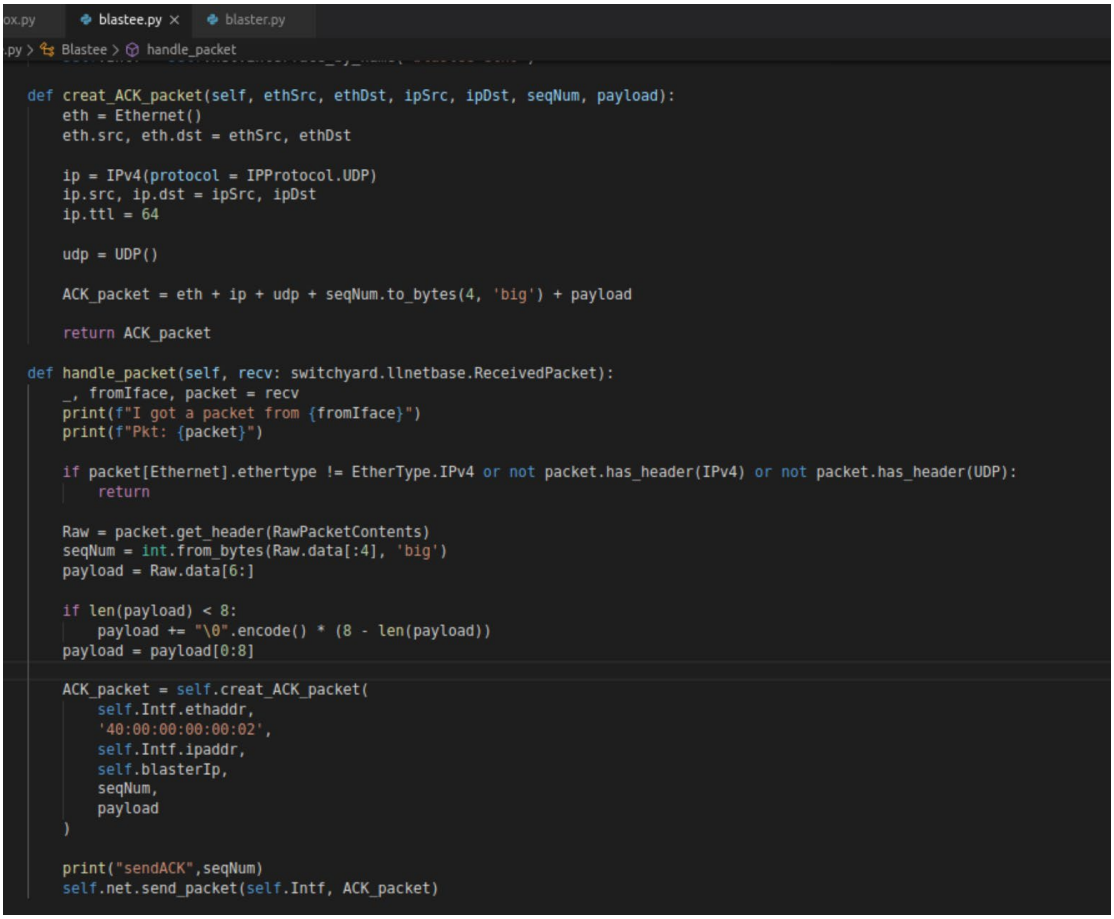


修改参数，middlebox 丢包率改为 0.5，发送 50 个数据包后，可见 blaster 的重发次数为 26，接近理论值。

4. 核心代码

middlebox.py

```python
        self.net = net
        self.dropRate = float(dropRate)
        self.intf1 = "middlebox-eth0"
        self.intf2 = "middlebox-eth1"
        self.my_interfaces = [self.net.interface_by_name(self.intf1), self.net.interface_by_name(self.intf2)]
        self.mymacs = [intf.ethaddr for intf in self.my_interfaces]
        self.myip = [intf.ipaddr for intf in self.my_interfaces]
        print("\n-----------------------\n")
        print(self.mymacs[0], self.mymacs[1])
        print("\n-----------------------\n")


    def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
        _, fromIface, packet = recv
        if fromIface == self.intf1:
            print("Received from blaster")
            '''
            Received data packet
            Should I drop it?
            If not, modify headers & send to blastee
            '''
            if random.random() <= self.dropRate:
                return

            packet[Ethernet].src = self.mymacs[0]
            packet[Ethernet].dst = self.mymacs[1]

            self.net.send_packet(self.intf2, packet)
            #self.net.send_packet("middlebox-eth1", packet)

        elif fromIface == "middlebox-eth1":
            print("Received from blastee")
            '''
            Received ACK
            Modify headers & send to blaster. Not dropping ACK packets!
            net.send_packet("middlebox-eth0", pkt)
            '''

            packet[Ethernet].src = self.mymacs[1]
            packet[Ethernet].dst = self.mymacs[0]

            self.net.send_packet(self.intf1, packet)
        else:
```


ox.py
blastee.py
blaster.py
.py > Blastee > handle_packet
def creat_ACK_packet(self, ethSrc, ethDst, ipSrc, ipDst, seqNum, payload):
eth = Ethernet()
eth.src, eth.dst = ethSrc, ethDst
ip = IPv4(protocol = IPProtocol.UDP)
ip.src, ip.dst = ipSrc, ipDst
ip.ttl = 64
udp = UDP()
ACK_packet = eth + ip + udp + seqNum.to_bytes(4, 'big') + payload
return ACK_packet
def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
_, fromIface, packet = recv
print(f"I got a packet from {fromIface}")
print(f"Pkt: {packet}")
if packet[Ethernet].ethertype != EtherType.IPv4 or not packet.has_header(IPv4) or not packet.has_header(UDP):
return
Raw = packet.get_header(RawPacketContents)
seqNum = int.from_bytes(Raw.data[:4], 'big')
payload = Raw.data[6:]
if len(payload) < 8:
payload += "\0".encode() * (8 - len(payload))
payload = payload[0:8]
ACK_packet = self.creat_ACK_packet(
self.Intf.ethaddr,
'40:00:00:00:00:02',
self.Intf.ipaddr,
self.blasterIp,
seqNum,
payload
)
print("sendACK",seqNum)
self.net.send_packet(self.Intf, ACK_packet)

```
def send_pkt(self):
    curTime = time.time() #save time now ,do not use time.time() in for currsion
    if curTime - self.recvTimeout / 1000 > self.lastsendtime:
        if len(self.sendqueue) != 0:
            entry = self.sendqueue.pop(0)
            packet = entry.packet
            self.lastsendtime = curTime
            self.Throughput += 1
            if entry.resendNum == 0:
                self.Goodput += 1
                print("send", entry.seqNum)
            else:
                self.reTX += 1
                print("resend", entry.seqNum)

            if self.firstpacketTime == -1:
                self.firstpacketTime = time.time()
            self.net.send_packet(self.Intf.name, packet)

        else:
            update = 0
            while len(self.sendWindow) and self.sendWindow[0].isACK == True:
                print("pop", self.sendWindow[0].seqNum)
                self.sendWindow.popleft()
                self.timestamp = time.time()
                update = 1
            if self.timestamp == -1:
                self.timestamp = time.time()
                update = 1

            curTime = time.time()

            while self.num != 0 and len(self.sendWindow) < self.limit:
```

5. 总结与感想

最开始没有注意到接收和发送的时延 rev_timeout，写了一种对每个包维护时间戳的方法。后来发现过后修改了做法，改成对每一组维护时间戳，并新增了发送和接收的队列。Blaster 处理滑动窗口的操作顺序要注意，要特判滑动窗口清空（初始）的情况。对 RawPacketContext 的操作要注意，特别是 payload 补齐 8 个字节和使用 API 进行大端数据的转换。