南京大学本科生实验报告

课程名称：**计算机网络**　　任课教师：黄程远　　助教：刘松岳

学院	**人工智能学院**	专业（方向）	**人工智能**
学号	**211300022**	姓名	**刘梦杰**
Email	**2757400745@qq.com**	开始/完成日期	**2025.1.10/2024.1.14**

1. 实验名称 计网 lab6

2. 实验目的 实现数据包的确认与重传

3. 实验内容与核心代码

a) **Middlebox：根据概率 droprate 对数据包进行转发或者丢弃以及转发 ack 即可；**

```
    r = random.random()
    if r > self.dropRate:
        packet[Ethernet].src = '40:00:00:00:00:02'
        packet[Ethernet].dst = '20:00:00:00:00:01'
        self.net.send_packet("middlebox-eth1", packet)
        log_info("Receive data packet")
    else:
        log_info("Drop data 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)
    '''
    packet[Ethernet].src = '40:00:00:00:00:01'
    packet[Ethernet].dst = '10:00:00:00:00:01'
    self.net.send_packet("middlebox-eth0", packet)
    log_info("Receive ACK")
else:
    log_debug("Oops :))")
```

b) **Balstee：构造 ack 包并根据情况填充负载；**

```
def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
    _, fromIface, packet = recv
    log_debug(f"I got a packet from {fromIface}")
    log_debug(f"Pkt: {packet}")

    eth_header = Ethernet(src="20:00:00:00:00:01", dst="40:00:00:00:00:02", ethertype=EtherType.IP)
    ipv4_header = IPv4(protocol=IPProtocol.UDP, src=IPv4Address('192.168.200.1'), dst=self.blasterIp)
    udp_header = UDP(src=12345, dst=54321)
    ack = eth_header + ipv4_header + udp_header
    payload_len = int.from_bytes(packet[3].to_bytes()[4:6], "big")
    if payload_len >= 8:
        ack = ack + packet[3].to_bytes()[:4] + packet[3].to_bytes()[6:14]
    else:
        ack = ack + packet[3].to_bytes()[:4] + packet[3].to_bytes()[6:] + (0).to_bytes(8 - payload_len, "big")
    self.net.send_packet(fromIface, ack)
    log_info(f"send ack: {ack}")
```

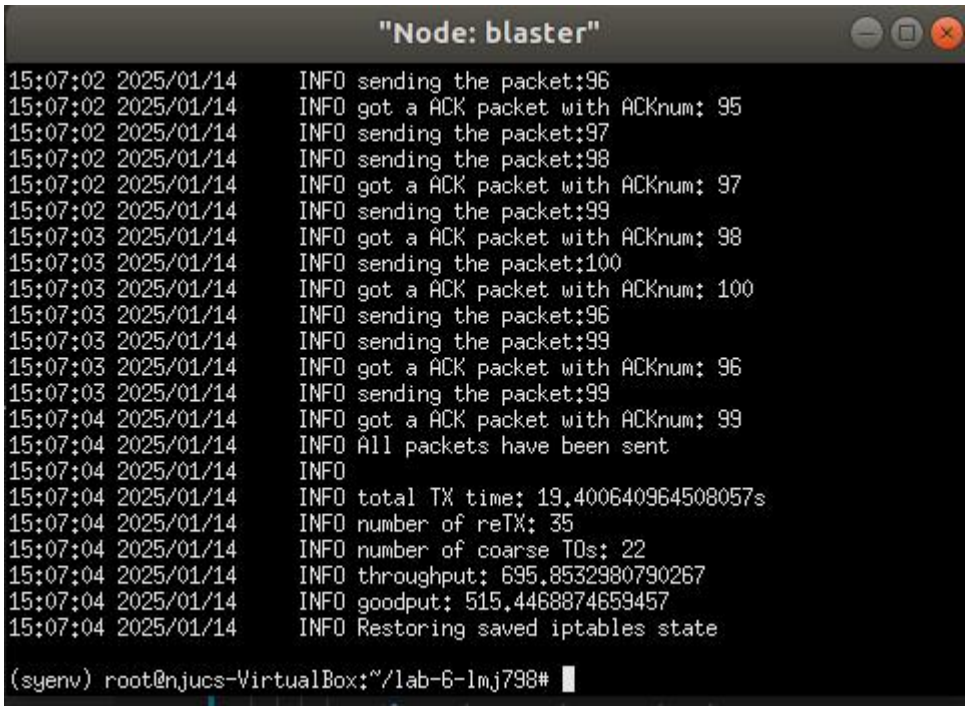
c) **Blaster：实现滑动窗口，输出一些变量，通过列表 self.acks 中的布尔值来确认包是否被 ack，是则将 LHS 右移，使用列表 self.transmit_seq 来存储窗口中待发送的包的序号，使用 transmit_single_packet()函数来依次发送列表中的包，再没有收到包的情况下发送，在发送包时累加吞吐量和有效吞吐量，其中用 self.count_of_send 记录发送次数，等于一时记录第一次发送包的时间，在 start 函数中收包时更新最后一次更新时间 self.last_send_time，从而计算总时间。**

```
def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
    _, fromIface, packet = recv
    self.time=time.time()
    seq = int.from_bytes(packet[3].to_bytes()[:4], 'big')
    log_info(f"got a ACK packet with ACKnum: {seq}")

    if seq in self.transmit_seq:
        self.transmit_seq.remove(seq)

    self.acks[seq]=True

    while self.acks[self.LHS] and self.LHS<=self.RHS:
        self.LHS+=1
        if self.LHS==self.num+1:
            break


    while self.RHS<self.num and self.RHS-self.LHS+1 < self.senderWindow:
        self.RHS+=1
        self.transmit_seq.append(self.RHS)

    if self.LHS==self.num+1:
        log_info("All packets have been sent")
        self.shutdown()

def handle_no_packet(self):
    if time.time()-self.time>self.timeout:
        self.number_of_coarse_TOs += 1
        self.time = time.time()
        for i in range(self.LHS,self.RHS+1):
            if not self.acks[i]:
                self.transmit_seq.append(i)
        self.transmit_single_packet()
    else:
        self.transmit_single_packet()
```

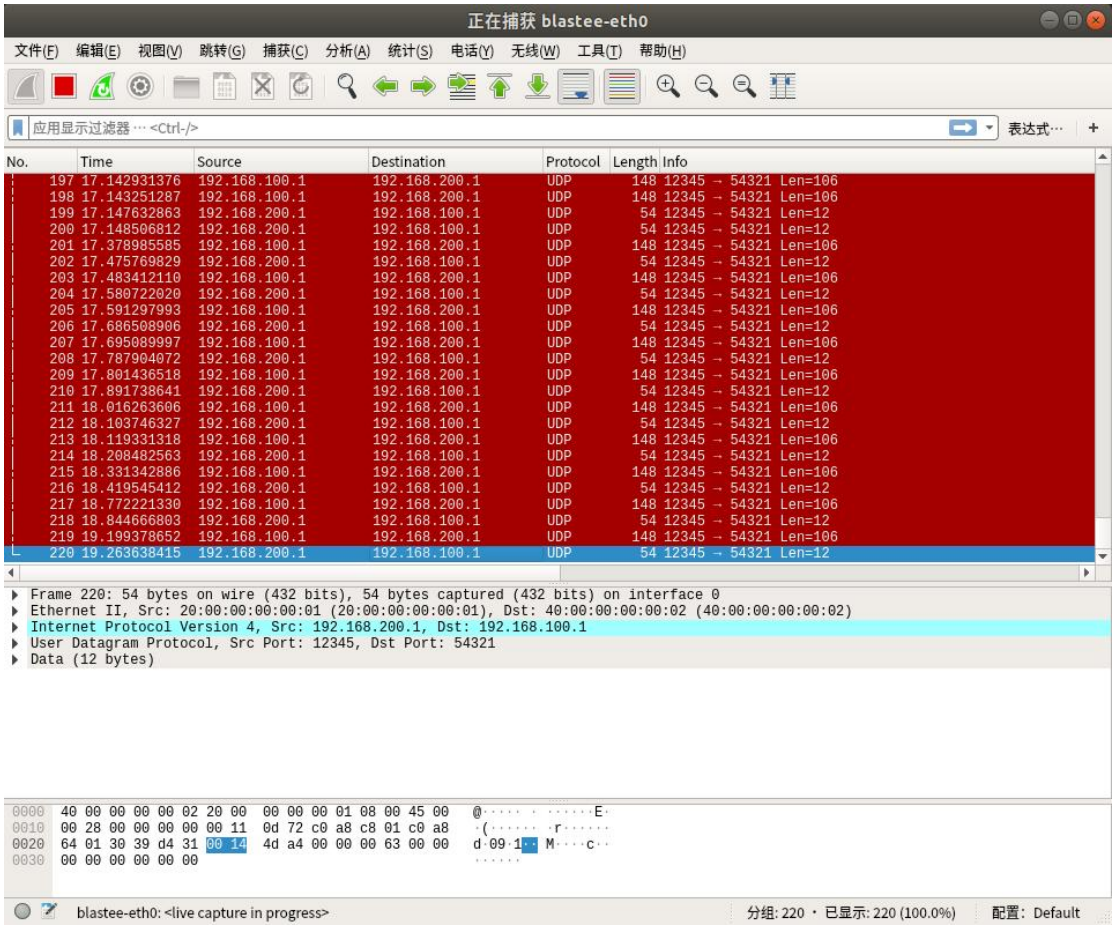
```
def transmit_single_packet(self):
    if len(self.transmit_seq) != 0:
        current_num = self.transmit_seq.pop(0)
        SequencePart = RawPacketContents(current_num.to_bytes(4,'big')+self.length.to_bytes(2,'big'))
        Payload = RawPacketContents(self.payloads[current_num])
        eth_header = Ethernet(src="10:00:00:00:00:01", dst="40:00:00:00:00:01")
        ipv4_header = IPv4(protocol=IPProtocol.UDP, src='192.168.100.1', dst=self.blasteeIp)
        udp_header = UDP(src=12345, dst=54321)
        pkt = eth_header+ipv4_header+udp_header+SequencePart+Payload
        seq_num = int.from_bytes(SequencePart.to_bytes()[:4],'big')
        log_info(f"sending the packet:{seq_num}")
        self.net.send_packet('blaster-eth0',pkt)

        self.count_of_send += 1
        if self.count_of_send == 1:
            self.first_send_time = time.time()

        if self.FirstSend[current_num] == True:
            self.goodput += len(Payload)
            self.FirstSend[current_num] = False
        else:
            self.retransmit_count += 1
        self.throughput += len(Payload)
        return

def start(self):
    '''
    A running daemon of the blaster.
    Receive packets until the end of time.
    '''
    while True:
        try:
            recv = self.net.recv_packet(timeout=self.recvTimeout)
            self.last_update_time = time.time()
        except NoPackets:
            self.handle_no_packet()
            continue
        except Shutdown:
            break
```

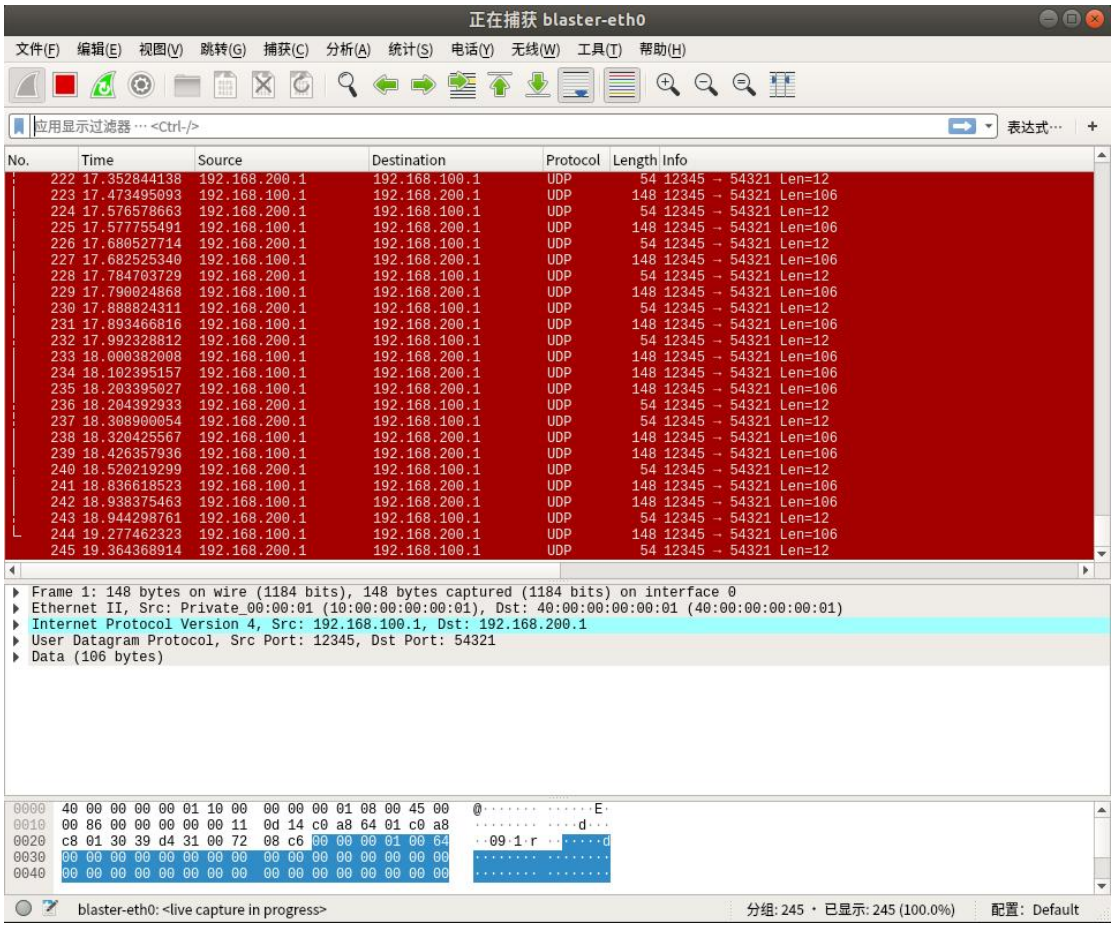
4. 实验结果：

a) Xterm 输出如下：



b) Wireshark 输出如下：




正在捕获 blaster-eth0
文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)
应用显示过滤器 … <Ctrl-/>
表达式…
No. Time Source Destination Protocol Length Info
222 17.352844138 192.168.200.1 192.168.100.1 UDP 54 12345 → 54321 Len=12
223 17.473495093 192.168.100.1 192.168.200.1 UDP 148 12345 → 54321 Len=106
224 17.576578663 192.168.200.1 192.168.100.1 UDP 54 12345 → 54321 Len=12
225 17.577755491 192.168.100.1 192.168.200.1 UDP 148 12345 → 54321 Len=106
226 17.680527714 192.168.200.1 192.168.100.1 UDP 54 12345 → 54321 Len=12
227 17.682525340 192.168.100.1 192.168.200.1 UDP 148 12345 → 54321 Len=106
228 17.784703729 192.168.200.1 192.168.100.1 UDP 54 12345 → 54321 Len=12
229 17.790024868 192.168.100.1 192.168.200.1 UDP 148 12345 → 54321 Len=106
230 17.888824311 192.168.200.1 192.168.100.1 UDP 54 12345 → 54321 Len=12
231 17.893466816 192.168.100.1 192.168.200.1 UDP 148 12345 → 54321 Len=106
232 17.992328812 192.168.200.1 192.168.100.1 UDP 54 12345 → 54321 Len=12
233 18.000382008 192.168.100.1 192.168.200.1 UDP 148 12345 → 54321 Len=106
234 18.102395157 192.168.100.1 192.168.200.1 UDP 148 12345 → 54321 Len=106
235 18.203395027 192.168.100.1 192.168.200.1 UDP 148 12345 → 54321 Len=106
236 18.204392933 192.168.200.1 192.168.100.1 UDP 54 12345 → 54321 Len=12
237 18.308900054 192.168.200.1 192.168.100.1 UDP 54 12345 → 54321 Len=12
238 18.320425567 192.168.100.1 192.168.200.1 UDP 148 12345 → 54321 Len=106
239 18.426357936 192.168.100.1 192.168.200.1 UDP 148 12345 → 54321 Len=106
240 18.520219299 192.168.200.1 192.168.100.1 UDP 54 12345 → 54321 Len=12
241 18.836618523 192.168.100.1 192.168.200.1 UDP 148 12345 → 54321 Len=106
242 18.938375463 192.168.100.1 192.168.200.1 UDP 148 12345 → 54321 Len=106
243 18.944298761 192.168.200.1 192.168.100.1 UDP 54 12345 → 54321 Len=12
244 19.277462323 192.168.100.1 192.168.200.1 UDP 148 12345 → 54321 Len=106
245 19.364368914 192.168.200.1 192.168.100.1 UDP 54 12345 → 54321 Len=12
Frame 1: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface 0
Ethernet II, Src: Private_00:00:01 (10:00:00:00:00:01), Dst: 40:00:00:00:00:01 (40:00:00:00:00:01)
Internet Protocol Version 4, Src: 192.168.100.1, Dst: 192.168.200.1
User Datagram Protocol, Src Port: 12345, Dst Port: 54321
Data (106 bytes)
0000 40 00 00 00 00 01 10 00 00 00 00 01 08 00 45 00
0010 00 86 00 00 00 00 00 11 0d 14 c0 a8 64 01 c0 a8
0020 c8 01 30 39 d4 31 00 72 08 c6 00 00 00 01 00 64
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
blaster-eth0: <live capture in progress>
分组: 245 · 已显示: 245 (100.0%)
配置: Default