# 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：

| | | | |
|---|---|---|---|
| 学院 | **工程管理院系** | 专业（方向） | **金融工程** |
| 学号 | **211275041** | 姓名 | **杨晨毅** |
| Email | | 开始/完成日期 | **5.27-5.29** |

## 1. 实验名称

**Lab-6 实现可靠通信**

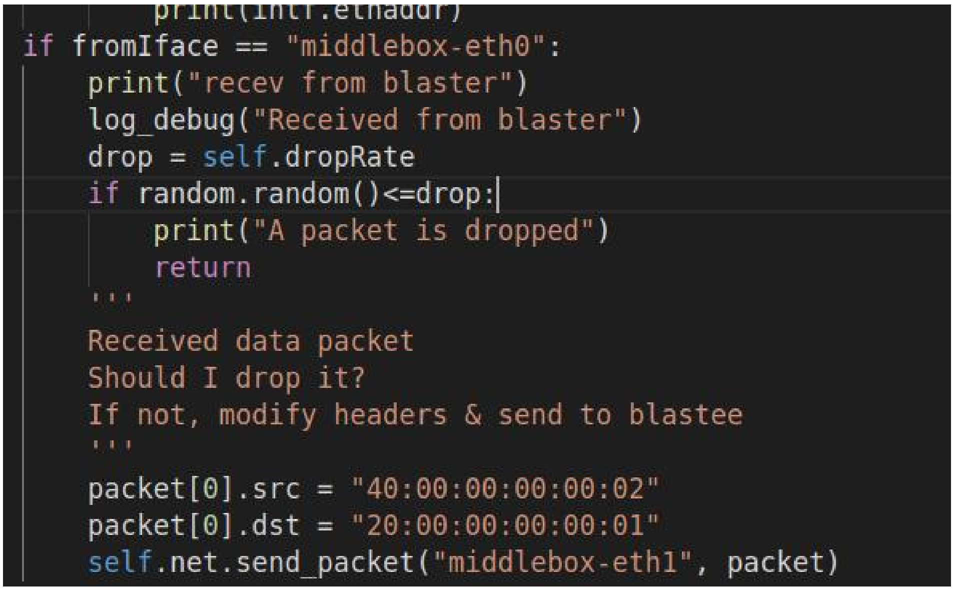
## 2. 实验目的

**实现 blaster、middlebox、blastee 功能**

## 3. 实验内容与核心代码

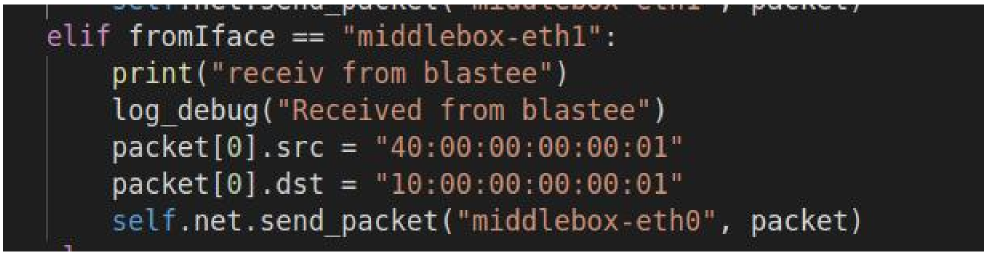
### 1. 实现 middlebox 的转发功能

**若从 blaster 发来，则概率丢包**



```
        print(intf.ethaddr)
if fromIface == "middlebox-eth0":
    print("recev from blaster")
    log_debug("Received from blaster")
    drop = self.dropRate
    if random.random()<=drop:
        print("A packet is dropped")
        return
    '''
    Received data packet
    Should I drop it?
    If not, modify headers & send to blastee
    '''
    packet[0].src = "40:00:00:00:00:02"
    packet[0].dst = "20:00:00:00:00:01"
    self.net.send_packet("middlebox-eth1", packet)
```

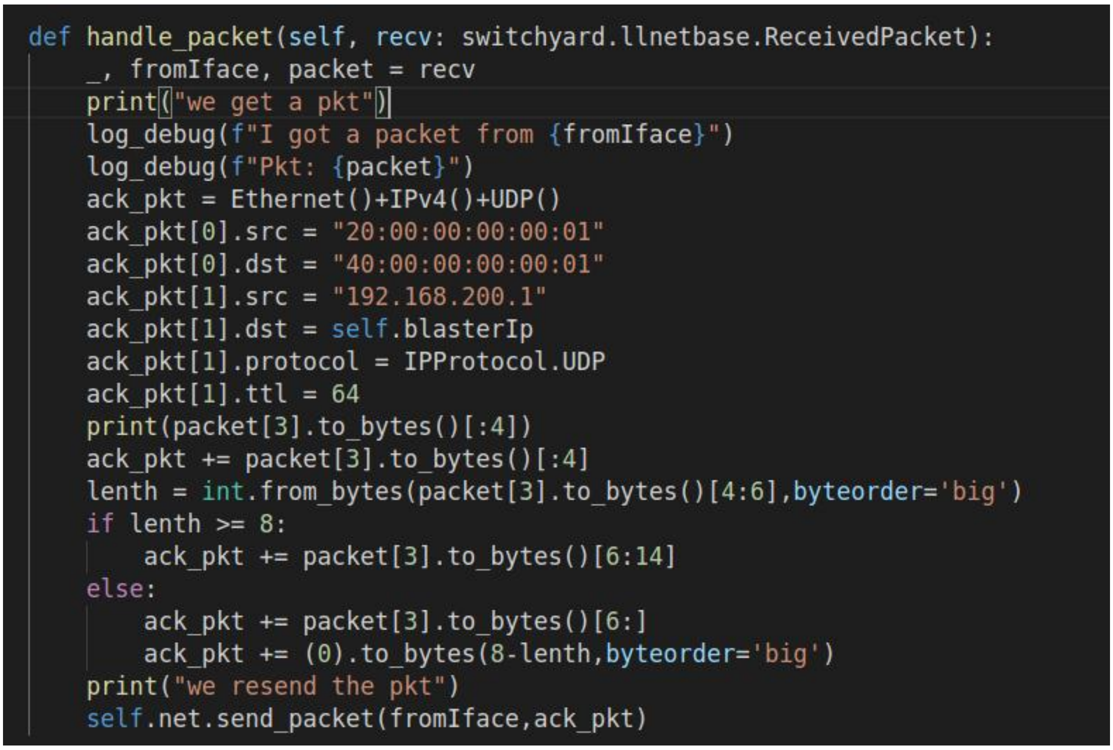
**若从 blastee 发来，则直接转发**



```
    self.net.send_packet("middlebox-eth1", packet)
elif fromIface == "middlebox-eth1":
    print("receiv from blastee")
    log_debug("Received from blastee")
    packet[0].src = "40:00:00:00:00:01"
    packet[0].dst = "10:00:00:00:00:01"
    self.net.send_packet("middlebox-eth0", packet)
```

## 2. 实现 blastee 的回复功能

首先复制序列号在前 32 比特，之后判断 payload 长度是否足够 8byte，如果不够就补 0；

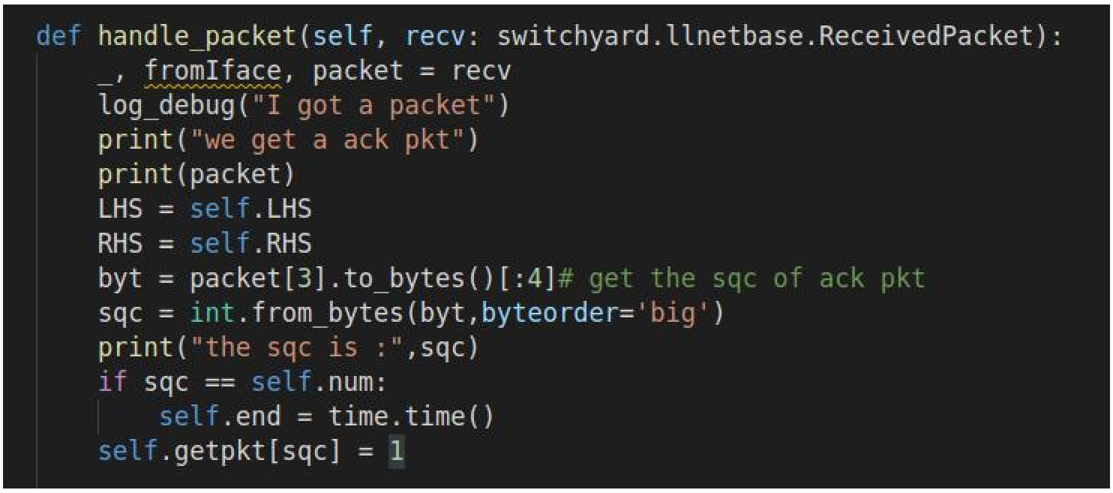


```python
def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
    _, fromIface, packet = recv
    print("we get a pkt")
    log_debug(f"I got a packet from {fromIface}")
    log_debug(f"Pkt: {packet}")
    ack_pkt = Ethernet()+IPv4()+UDP()
    ack_pkt[0].src = "20:00:00:00:00:01"
    ack_pkt[0].dst = "40:00:00:00:00:01"
    ack_pkt[1].src = "192.168.200.1"
    ack_pkt[1].dst = self.blasterIp
    ack_pkt[1].protocol = IPProtocol.UDP
    ack_pkt[1].ttl = 64
    print(packet[3].to_bytes()[:4])
    ack_pkt += packet[3].to_bytes()[:4]
    lenth = int.from_bytes(packet[3].to_bytes()[4:6],byteorder='big')
    if lenth >= 8:
        ack_pkt += packet[3].to_bytes()[6:14]
    else:
        ack_pkt += packet[3].to_bytes()[6:]
        ack_pkt += (0).to_bytes(8-lenth,byteorder='big')
    print("we resend the pkt")
    self.net.send_packet(fromIface,ack_pkt)
```

## 3. 实现 blaster 的发包、sendwind、重发包功能

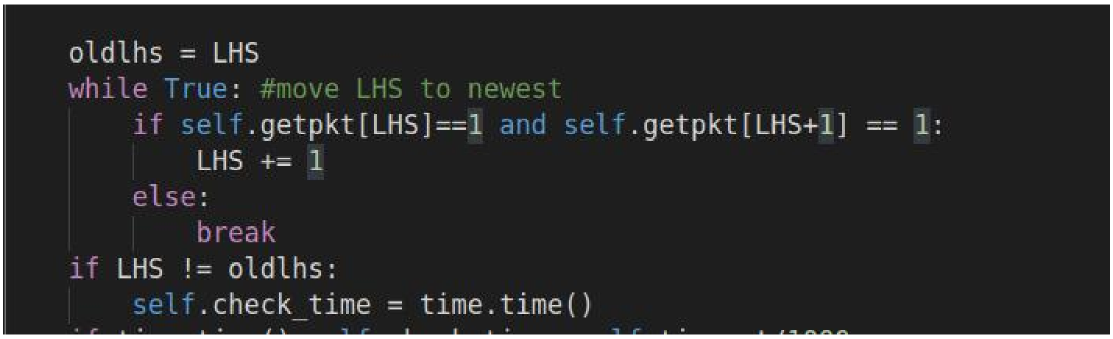
### （1）handle_packet

首先在 handle 下取出包的序列号 sqc 然后对相应数据赋 1 表示已接到相应序列的包



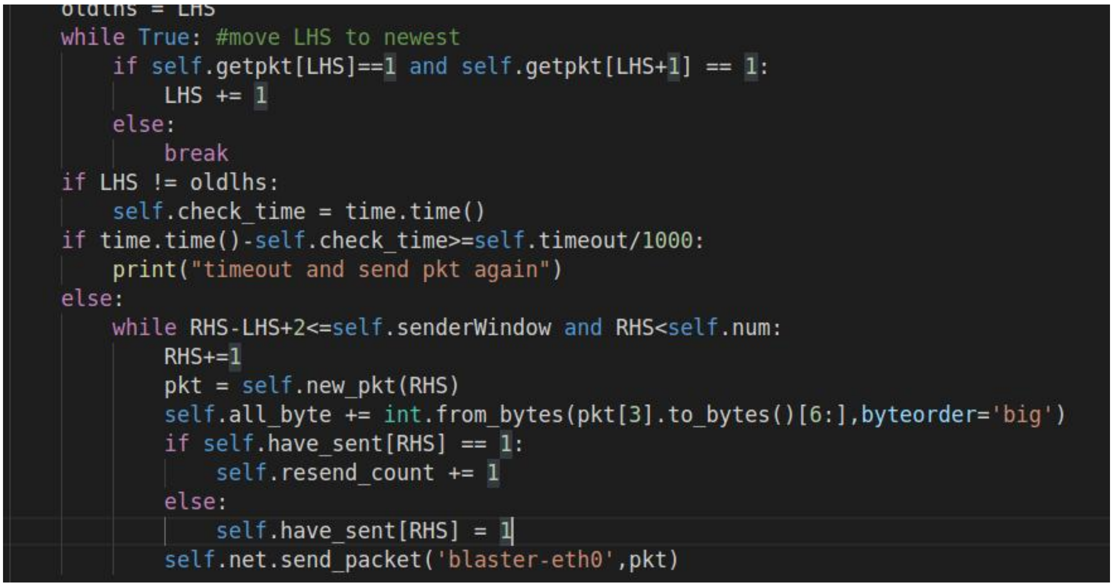
```python
def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
    _, fromIface, packet = recv
    log_debug("I got a packet")
    print("we get a ack pkt")
    print(packet)
    LHS = self.LHS
    RHS = self.RHS
    byt = packet[3].to_bytes()[:4]# get the sqc of ack pkt
    sqc = int.from_bytes(byt,byteorder='big')
    print("the sqc is :",sqc)
    if sqc == self.num:
        self.end = time.time()
    self.getpkt[sqc] = 1
```

接着更新 **LHS** 的指向，若成功则重设重传时间



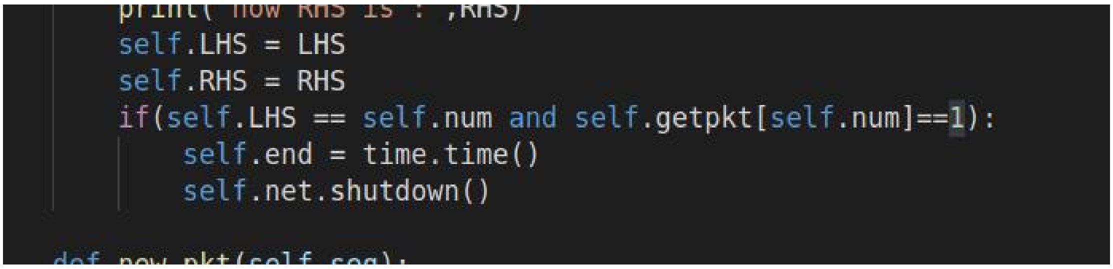
```
oldlhs = LHS
while True: #move LHS to newest
    if self.getpkt[LHS]==1 and self.getpkt[LHS+1] == 1:
        LHS += 1
    else:
        break
if LHS != oldlhs:
    self.check_time = time.time()
```

之后对是否超时进行判断，若超时则跳过让 **handle_no** 做，若不超时则进行发包



```
oldlhs = LHS
while True: #move LHS to newest
    if self.getpkt[LHS]==1 and self.getpkt[LHS+1] == 1:
        LHS += 1
    else:
        break
if LHS != oldlhs:
    self.check_time = time.time()
if time.time()-self.check_time>=self.timeout/1000:
    print("timeout and send pkt again")
else:
    while RHS-LHS+2<=self.senderWindow and RHS<self.num:
        RHS+=1
        pkt = self.new_pkt(RHS)
        self.all_byte += int.from_bytes(pkt[3].to_bytes()[6:],byteorder='big')
        if self.have_sent[RHS] == 1:
            self.resend_count += 1
        else:
            self.have_sent[RHS] = 1
        self.net.send_packet('blaster-eth0',pkt)
```

最后进行结束条件的判断

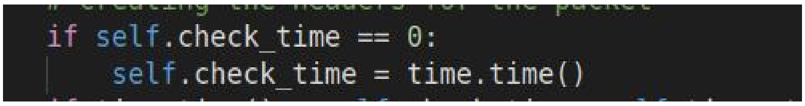


```
    print("now RHS is :",RHS)
    self.LHS = LHS
    self.RHS = RHS
    if(self.LHS == self.num and self.getpkt[self.num]==1):
        self.end = time.time()
        self.net.shutdown()

def new_pkt(self,seq):
```

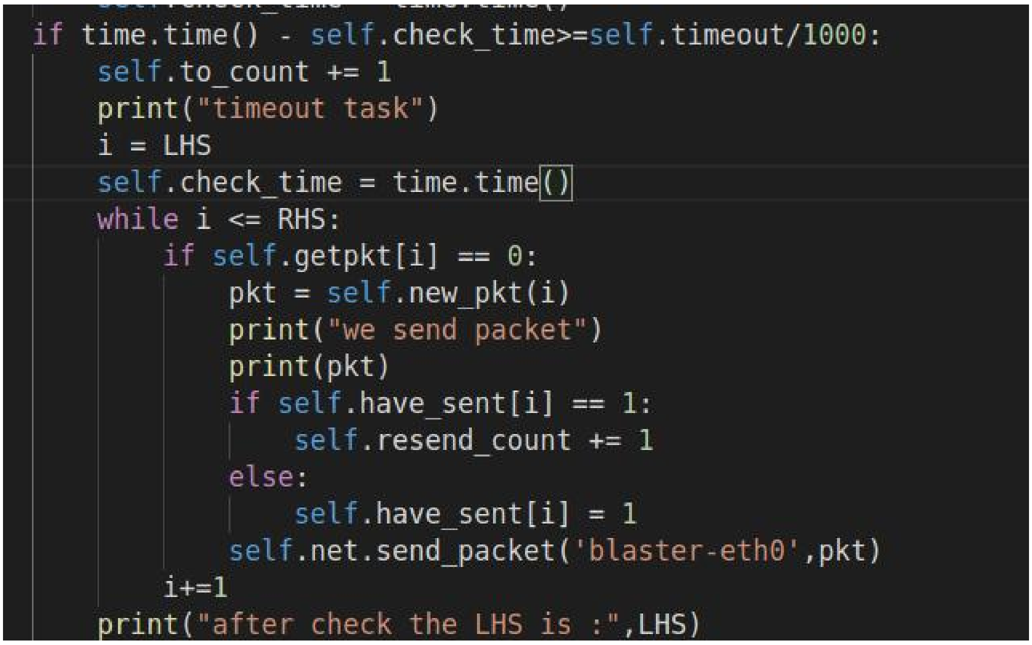
（2）**handle_no_packet**

进行开始时间的设置



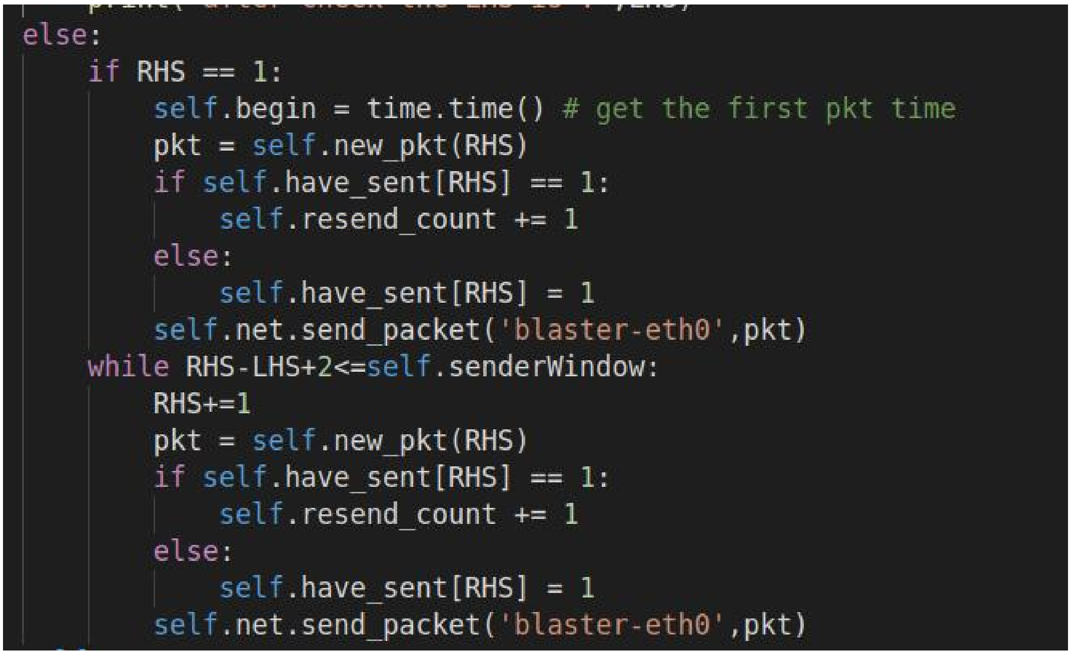
```
# creating the headers for the packet
if self.check_time == 0:
    self.check_time = time.time()
```

之后进行超时重传从 **LHS** 开始，若 **getpkt[i]**不为 **1**（没收到包）则进行记录后重传。



```
if time.time() - self.check_time>=self.timeout/1000:
    self.to_count += 1
    print("timeout task")
    i = LHS
    self.check_time = time.time()
    while i <= RHS:
        if self.getpkt[i] == 0:
            pkt = self.new_pkt(i)
            print("we send packet")
            print(pkt)
            if self.have_sent[i] == 1:
                self.resend_count += 1
            else:
                self.have_sent[i] = 1
            self.net.send_packet('blaster-eth0',pkt)
        i+=1
    print("after check the LHS is :",LHS)
```

之后进行未超时的处理以 **sendwindow** 为标准判断是否应接下去发包。

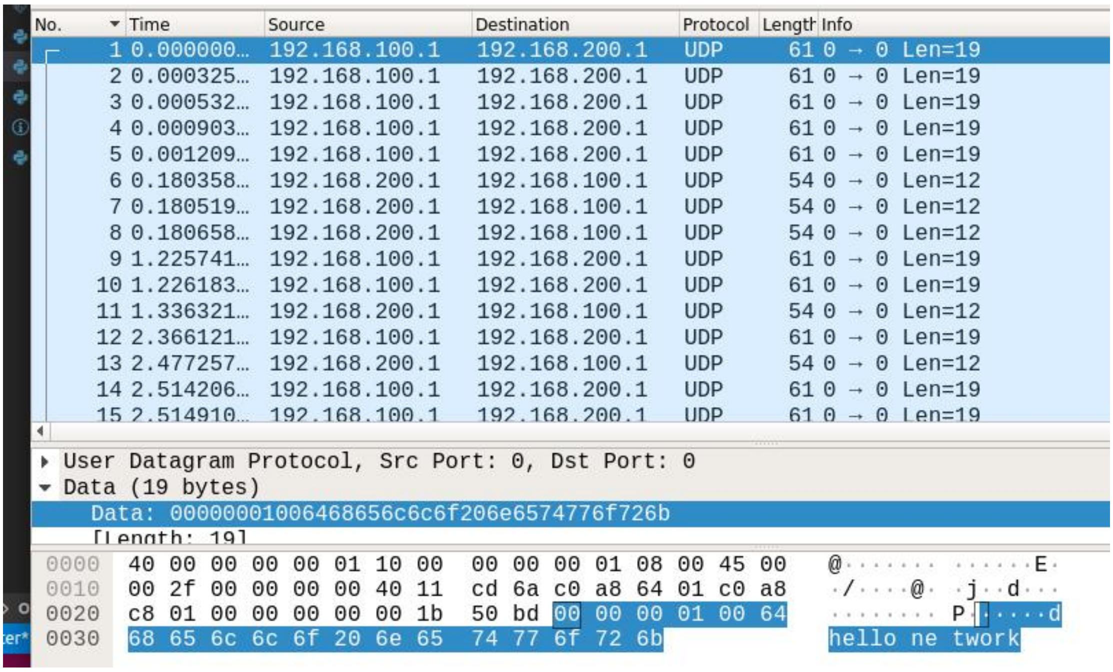


```
else:
    if RHS == 1:
        self.begin = time.time() # get the first pkt time
        pkt = self.new_pkt(RHS)
        if self.have_sent[RHS] == 1:
            self.resend_count += 1
        else:
            self.have_sent[RHS] = 1
        self.net.send_packet('blaster-eth0',pkt)
    while RHS-LHS+2<=self.senderWindow:
        RHS+=1
        pkt = self.new_pkt(RHS)
        if self.have_sent[RHS] == 1:
            self.resend_count += 1
        else:
            self.have_sent[RHS] = 1
        self.net.send_packet('blaster-eth0',pkt)
```

最后抓包：

对第一个 **blaster** 发出的包，**16** 进制下，蓝色前 **4** 个为序列号，后两

个为 length，最后为 payload



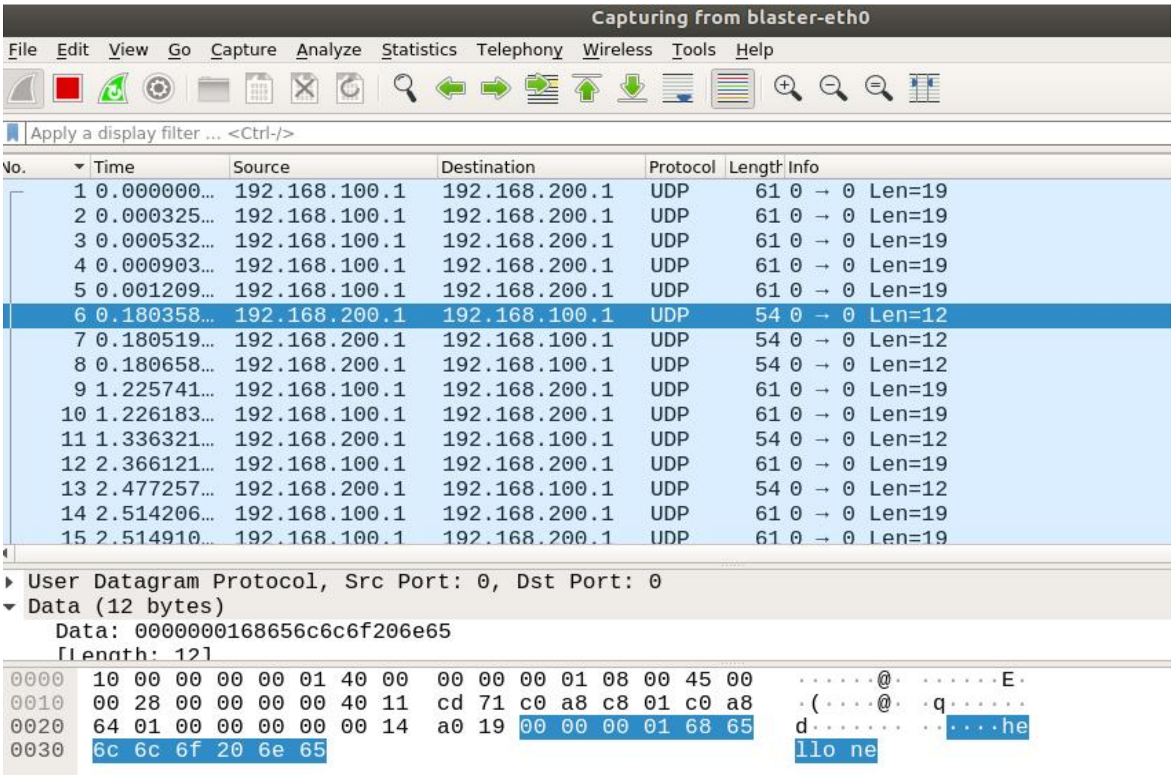
| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 0.000000… | 192.168.100.1 | 192.168.200.1 | UDP | 61 | 0 → 0 Len=19 |
| 2 | 0.000325… | 192.168.100.1 | 192.168.200.1 | UDP | 61 | 0 → 0 Len=19 |
| 3 | 0.000532… | 192.168.100.1 | 192.168.200.1 | UDP | 61 | 0 → 0 Len=19 |
| 4 | 0.000903… | 192.168.100.1 | 192.168.200.1 | UDP | 61 | 0 → 0 Len=19 |
| 5 | 0.001209… | 192.168.100.1 | 192.168.200.1 | UDP | 61 | 0 → 0 Len=19 |
| 6 | 0.180358… | 192.168.200.1 | 192.168.100.1 | UDP | 54 | 0 → 0 Len=12 |
| 7 | 0.180519… | 192.168.200.1 | 192.168.100.1 | UDP | 54 | 0 → 0 Len=12 |
| 8 | 0.180658… | 192.168.200.1 | 192.168.100.1 | UDP | 54 | 0 → 0 Len=12 |
| 9 | 1.225741… | 192.168.100.1 | 192.168.200.1 | UDP | 61 | 0 → 0 Len=19 |
| 10 | 1.226183… | 192.168.100.1 | 192.168.200.1 | UDP | 61 | 0 → 0 Len=19 |
| 11 | 1.336321… | 192.168.200.1 | 192.168.100.1 | UDP | 54 | 0 → 0 Len=12 |
| 12 | 2.366121… | 192.168.100.1 | 192.168.200.1 | UDP | 61 | 0 → 0 Len=19 |
| 13 | 2.477257… | 192.168.200.1 | 192.168.100.1 | UDP | 54 | 0 → 0 Len=12 |
| 14 | 2.514206… | 192.168.100.1 | 192.168.200.1 | UDP | 61 | 0 → 0 Len=19 |
| 15 | 2.514910… | 192.168.100.1 | 192.168.200.1 | UDP | 61 | 0 → 0 Len=19 |

```
▸ User Datagram Protocol, Src Port: 0, Dst Port: 0
▾ Data (19 bytes)
    Data: 00000001006468656c6c6f206e6574776f726b
    [Length: 19]
```

```
0000  40 00 00 00 00 01 10 00  00 00 00 01 08 00 45 00   @·······  ······E·
0010  00 2f 00 00 00 00 40 11  cd 6a c0 a8 64 01 c0 a8   ·/····@·  ·j··d···
0020  c8 01 00 00 00 00 00 1b  50 bd 00 00 00 01 00 64   ········  P·····d
0030  68 65 6c 6c 6f 20 6e 65  74 77 6f 72 6b            hello ne  twork
```

对从 blastee 的回复包分析：

16 进制下，前四个为序列号，后 8 个为 payload 的前部截取校验



| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 0.000000… | 192.168.100.1 | 192.168.200.1 | UDP | 61 | 0 → 0 Len=19 |
| 2 | 0.000325… | 192.168.100.1 | 192.168.200.1 | UDP | 61 | 0 → 0 Len=19 |
| 3 | 0.000532… | 192.168.100.1 | 192.168.200.1 | UDP | 61 | 0 → 0 Len=19 |
| 4 | 0.000903… | 192.168.100.1 | 192.168.200.1 | UDP | 61 | 0 → 0 Len=19 |
| 5 | 0.001209… | 192.168.100.1 | 192.168.200.1 | UDP | 61 | 0 → 0 Len=19 |
| 6 | 0.180358… | 192.168.200.1 | 192.168.100.1 | UDP | 54 | 0 → 0 Len=12 |
| 7 | 0.180519… | 192.168.200.1 | 192.168.100.1 | UDP | 54 | 0 → 0 Len=12 |
| 8 | 0.180658… | 192.168.200.1 | 192.168.100.1 | UDP | 54 | 0 → 0 Len=12 |
| 9 | 1.225741… | 192.168.100.1 | 192.168.200.1 | UDP | 61 | 0 → 0 Len=19 |
| 10 | 1.226183… | 192.168.100.1 | 192.168.200.1 | UDP | 61 | 0 → 0 Len=19 |
| 11 | 1.336321… | 192.168.200.1 | 192.168.100.1 | UDP | 54 | 0 → 0 Len=12 |
| 12 | 2.366121… | 192.168.100.1 | 192.168.200.1 | UDP | 61 | 0 → 0 Len=19 |
| 13 | 2.477257… | 192.168.200.1 | 192.168.100.1 | UDP | 54 | 0 → 0 Len=12 |
| 14 | 2.514206… | 192.168.100.1 | 192.168.200.1 | UDP | 61 | 0 → 0 Len=19 |
| 15 | 2.514910… | 192.168.100.1 | 192.168.200.1 | UDP | 61 | 0 → 0 Len=19 |

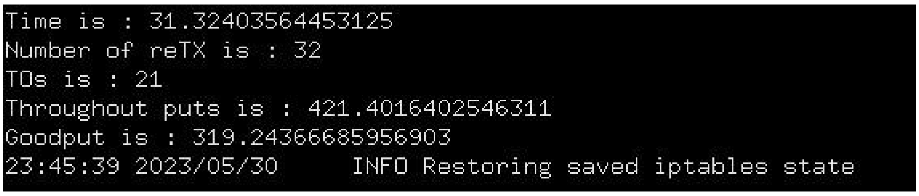
```
▸ User Datagram Protocol, Src Port: 0, Dst Port: 0
▾ Data (12 bytes)
    Data: 0000000168656c6c6f206e65
    [Length: 12]
```

```
0000  10 00 00 00 00 01 40 00  00 00 00 01 08 00 45 00   ······@·  ······E·
0010  00 28 00 00 00 00 40 11  cd 71 c0 a8 c8 01 c0 a8   ·(····@·  ·q······
0020  64 01 00 00 00 00 00 14  a0 19 00 00 00 01 68 65   d·······  ······he
0030  6c 6c 6f 20 6e 65                                  llo ne
```

包的结构正确

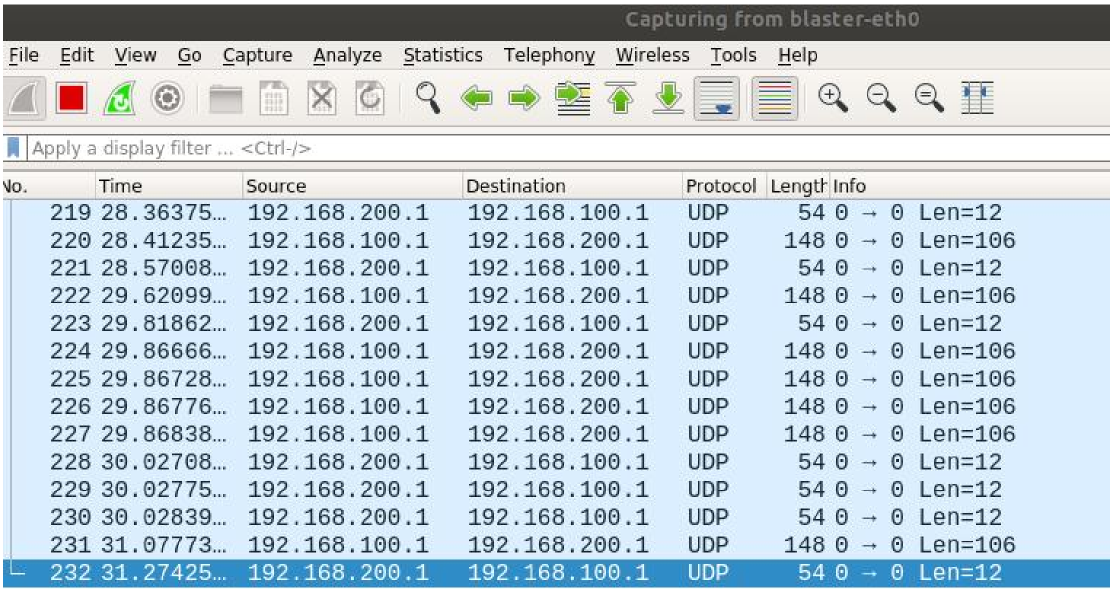
## 4. 实验结果

以下是输出截图



```
Time is : 31.32403564453125
Number of reTX is : 32
TOs is : 21
Throughout puts is : 421.4016402546311
Goodput is : 319.24366685956903
23:45:39 2023/05/30      INFO Restoring saved iptables state
```

与 **blaster** 最后一个收包对比可知时间记录、重传次数正确



| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 219 | 28.36375… | 192.168.200.1 | 192.168.100.1 | UDP | 54 | 0 → 0 Len=12 |
| 220 | 28.41235… | 192.168.100.1 | 192.168.200.1 | UDP | 148 | 0 → 0 Len=106 |
| 221 | 28.57008… | 192.168.200.1 | 192.168.100.1 | UDP | 54 | 0 → 0 Len=12 |
| 222 | 29.62099… | 192.168.100.1 | 192.168.200.1 | UDP | 148 | 0 → 0 Len=106 |
| 223 | 29.81862… | 192.168.200.1 | 192.168.100.1 | UDP | 54 | 0 → 0 Len=12 |
| 224 | 29.86666… | 192.168.100.1 | 192.168.200.1 | UDP | 148 | 0 → 0 Len=106 |
| 225 | 29.86728… | 192.168.100.1 | 192.168.200.1 | UDP | 148 | 0 → 0 Len=106 |
| 226 | 29.86776… | 192.168.100.1 | 192.168.200.1 | UDP | 148 | 0 → 0 Len=106 |
| 227 | 29.86838… | 192.168.100.1 | 192.168.200.1 | UDP | 148 | 0 → 0 Len=106 |
| 228 | 30.02708… | 192.168.200.1 | 192.168.100.1 | UDP | 54 | 0 → 0 Len=12 |
| 229 | 30.02775… | 192.168.200.1 | 192.168.100.1 | UDP | 54 | 0 → 0 Len=12 |
| 230 | 30.02839… | 192.168.200.1 | 192.168.100.1 | UDP | 54 | 0 → 0 Len=12 |
| 231 | 31.07773… | 192.168.100.1 | 192.168.200.1 | UDP | 148 | 0 → 0 Len=106 |
| 232 | 31.27425… | 192.168.200.1 | 192.168.100.1 | UDP | 54 | 0 → 0 Len=12 |

同时 **Goodput/Throughput =0.757 = 100/132** 数据正确

## 5. 总结与感想

进一步加深理解可靠通信