Computer Networking-Lab-Repot

课程名称：**计算机网络** 任课教师：**田臣/李文中**

学院	**Dept. of Computer Science and Technology**	专业（方向）	**CS**
学号	**181830044**	姓名	**董宸郅**
Email	**pdon#foxmail.com**	开始/完成日期	**5.6/5.27**

实验名称： **Lab 6: Reliable Communication**

实验目的

- 实现一个简易的可靠传输机制
- 加深对于网络层与传输层的理解
- 深入理解UDP协议的工作机制

实验内容

理论知识

UDP

UDP是OSI参考模型中一种无连接的传输层协议，它主要用于**不要求分组顺序到达**的传输中，分组传输顺序的检查与排序由应用层完成，提供面向事务的简单不可靠信息传送服务。UDP 协议基本上是IP协议与上层协议的接口，是一个**无连接协议**，不需要维护连接状态。UDP报头由4个域组成，其中每个域各占用2个字节，具体包括源端口号、目标端口号、数据报长度、校验值。

实验步骤（含测试结果与关键代码）

`FYI`：代码逻辑细节见注释

Task 2: Middlebox

`Coding`

设计一个类`MiddleBox`用于记录关键信息：

`drop_now`：判断是否要丢弃当前的包

```
class MiddleBox:
    def __init__(self):
        '''
        init some useful information
        '''
        # number of total pkts received from blaster
        self.total_pkt = 0
        # number of pkts forwarded to blastee
        self.fwd_pkt = 0
        input_file = open('middlebox_params.txt', 'r')
        self.drop_rate = float(input_file.readline().split()[1])

        self.macs = {}
        self.macs['blaster'] = '10:00:00:00:00:01'
        self.macs['blastee'] = '20:00:00:00:00:01'
        self.macs['mb2blaster'] = '40:00:00:00:00:01'
        self.macs['mb2blastee'] = '40:00:00:00:00:02'

        self.ips = {}
        self.ips['blaster'] = '192.168.100.1'
        self.ips['blastee'] = '192.168.200.1'
        self.ips['mb2blaster'] = '192.168.100.2'
        self.ips['mb2blastee'] = '192.168.200.2'

    def drop_now(self):
        '''
        decide whether to drop the pkt
        '''
        if random() < self.drop_rate:
            return True
        else:
            return False
```

根据所收到包的来源，作相应的简易转发；如果是`blaster`发到`blastee`的，判断是否丢包：

```
if dev == "middlebox-eth0":
    mb.total_pkt += 1
    log_debug("Received from blaster")
    '''
    Received data packet
    Should I drop it?
    If not, modify headers & send to blastee
    '''
    if not mb.drop_now():
        mb.fwd_pkt += 1
        pkt[Ethernet].src = mb.macs['mb2blastee']
        pkt[Ethernet].dst = mb.macs['blastee']
        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[Ethernet].src = mb.macs['mb2blaster']
    pkt[Ethernet].dst = mb.macs['blaster']
    net.send_packet("middlebox-eth0", pkt)
```

Task 3: Blastee

`Coding`

设计一个类`Blastee`记录关键信息：

`mk_pkt`：从来自blaster的pkt中提取信息并生成相应的ACK包

`safe_exit`：判断当前是否能终止blastee

```python
class Blastee:
    def __init__(self):
        '''
        init some useful information
        '''
        self.pkt_cnt = 0
        self.acked = []

        input_file = open('blastee_params.txt', 'r')
        params = input_file.readline().split()
        self.blaster_IP = str(params[1])  # useless actually
        self.num = int(params[3])

        self.macs = {}
        self.macs['blaster'] = '10:00:00:00:00:01'
        self.macs['blastee'] = '20:00:00:00:00:01'
        self.macs['mb2blaster'] = '40:00:00:00:00:01'
        self.macs['mb2blastee'] = '40:00:00:00:00:02'

        self.ips = {}
        self.ips['blaster'] = '192.168.100.1'
        self.ips['blastee'] = '192.168.200.1'
        self.ips['mb2blaster'] = '192.168.100.2'
        self.ips['mb2blastee'] = '192.168.200.2'

    def mk_ack(self, pkt):
        '''
        create ACK for received pkts
        '''
        hdr = Ethernet() + IPv4(protocol=IPProtocol.UDP) + UDP()
        hdr[Ethernet].src = self.macs['blastee']
        hdr[Ethernet].dst = self.macs['mb2blastee']
        hdr[IPv4].src = self.ips['blastee']
        hdr[IPv4].dst = self.ips['blaster']
        # extract 'seq_num' from the pkt
        seq_num_raw = (pkt[RawPacketContents].to_bytes())[:4]
        seq_num = int.from_bytes(seq_num_raw, 'big')
        # inorder to end blastee properly
        # only non-acked pkt should be recorded
        if seq_num not in self.acked:
            self.acked.append(seq_num)
            self.pkt_cnt += 1
        # extract 'length' from the pkt
        len = int.from_bytes((pkt[RawPacketContents].to_bytes())[4:6], 'big')
        if len < 8:
            # stuff the empty space
            payload = (pkt[RawPacketContents].to_bytes())[6:] + bytes(8 - len)
        else:
            payload = (pkt[RawPacketContents].to_bytes())[6:14]
        # add up the 3 parts above
        return hdr + seq_num_raw + payload

    def safe_exit(self):
        '''
        decide whether it's OK to end blastee
        '''
        if self.pkt_cnt < self.num:
            return False
        # no longer necessary since blaster won't send pkt with seq_num>num
        # for x in range(1, self.num):
        #     if x not in self.acked:
        #         return False
        return True
```

回复ACK：

```
new_pkt = blastee.mk_ack(pkt)
net.send_packet(dev, new_pkt)
```

Task 4: Blaster

`Coding`

设计一个类`Blaster`记录关键信息：

`mk_pkt`：生成含有对应`seq_num`的包

```
class Blaster:
    def __init__(self):
        '''
        init some useful information
        '''
        self.lhs = 1
        self.rhs = 0
        # acked pkts' seq_num
        self.acked = []
        # the last timestamp when lhs is sent
        self.lhs_send_time = 0.0
        # the timestamp when first pkt is sent
        self.start = 0.0
        # the total cnt of pkts resent
        self.retrans = 0
        # the total cnt of timeouts
        self.to_times = 0

        input_file = open('blaster_params.txt', 'r')
        params = input_file.readline().split()
        self.blastee_IP = str(params[1])  # useless actually
        self.num = int(params[3])
        self.len = int(params[5])
        self.sw = int(params[7])
        self.to = float(params[9]) / 1000
        self.recv_to = float(params[11]) / 1000

        self.macs = {}
        self.macs['blaster'] = '10:00:00:00:00:01'
        self.macs['blastee'] = '20:00:00:00:00:01'
        self.macs['mb2blaster'] = '40:00:00:00:00:01'
```

```python
        self.macs['mb2blastee'] = '40:00:00:00:00:02'

        self.ips = {}
        self.ips['blaster'] = '192.168.100.1'
        self.ips['blastee'] = '192.168.200.1'
        self.ips['mb2blaster'] = '192.168.100.2'
        self.ips['mb2blastee'] = '192.168.200.2'

    def mk_pkt(self, seq_num):
        '''
        create ACK for received pkts
        '''
        hdr = Ethernet() + IPv4(protocol=IPProtocol.UDP) + UDP()
        hdr[Ethernet].src = self.macs['blaster']
        hdr[Ethernet].dst = self.macs['mb2blaster']
        hdr[IPv4].src = self.ips['blaster']
        hdr[IPv4].dst = self.ips['blastee']
        # transform data into rawbyte format
        seq_num = seq_num.to_bytes(4, 'big')
        length = self.len.to_bytes(2, 'big')
        payload = bytes(self.len)
        # add up the 3 parts above
        return hdr + seq_num + length + payload
```

- 收到ACK：

```python
ack_seq = int.from_bytes((pkt[RawPacketContents].to_bytes())[:4], 'big')
# add new 'seq_num' to acked[]
if ack_seq not in blaster.acked:
    blaster.acked.append(ack_seq)

if ack_seq == blaster.lhs:
    '''
    1. change 'lhs' to the most right postion
        where all pkts to the left have been acked
    2. decide whether to end blaster according to 'num'
    '''
    blaster.lhs += 1
    if blaster.lhs - 1 == blaster.num:
        break
    while blaster.lhs in blaster.acked:
        blaster.lhs += 1
        if blaster.lhs - 1 == blaster.num:
            break
```

- 未收到ACK：

```python
# send new pkt
if blaster.rhs - blaster.lhs + 1 < blaster.sw:
    blaster.rhs += 1
    if blaster.rhs == 1:
        blaster.lhs_send_time = time.time()
        blaster.start = blaster.lhs_send_time
    if blaster.rhs <= blaster.num:
        net.send_packet('blaster-eth0', blaster.mk_pkt(blaster.rhs))

```

```
# check timeout for lhs
if time.time() - blaster.lhs_send_time > blaster.to:
    # update key info
    blaster.to_times += 1
    blaster.lhs_send_time = time.time()
    # all pkts within sender window and non-acked should be resent
    for x in range(blaster.lhs, min(blaster.rhs, blaster.num) + 1):
        if x not in blaster.acked:
            blaster.retrans += 1
            net.send_packet('blaster-eth0', blaster.mk_pkt(x))
```

Task 5: Running your code

各项参数取值：

middlebox：-d 0.23

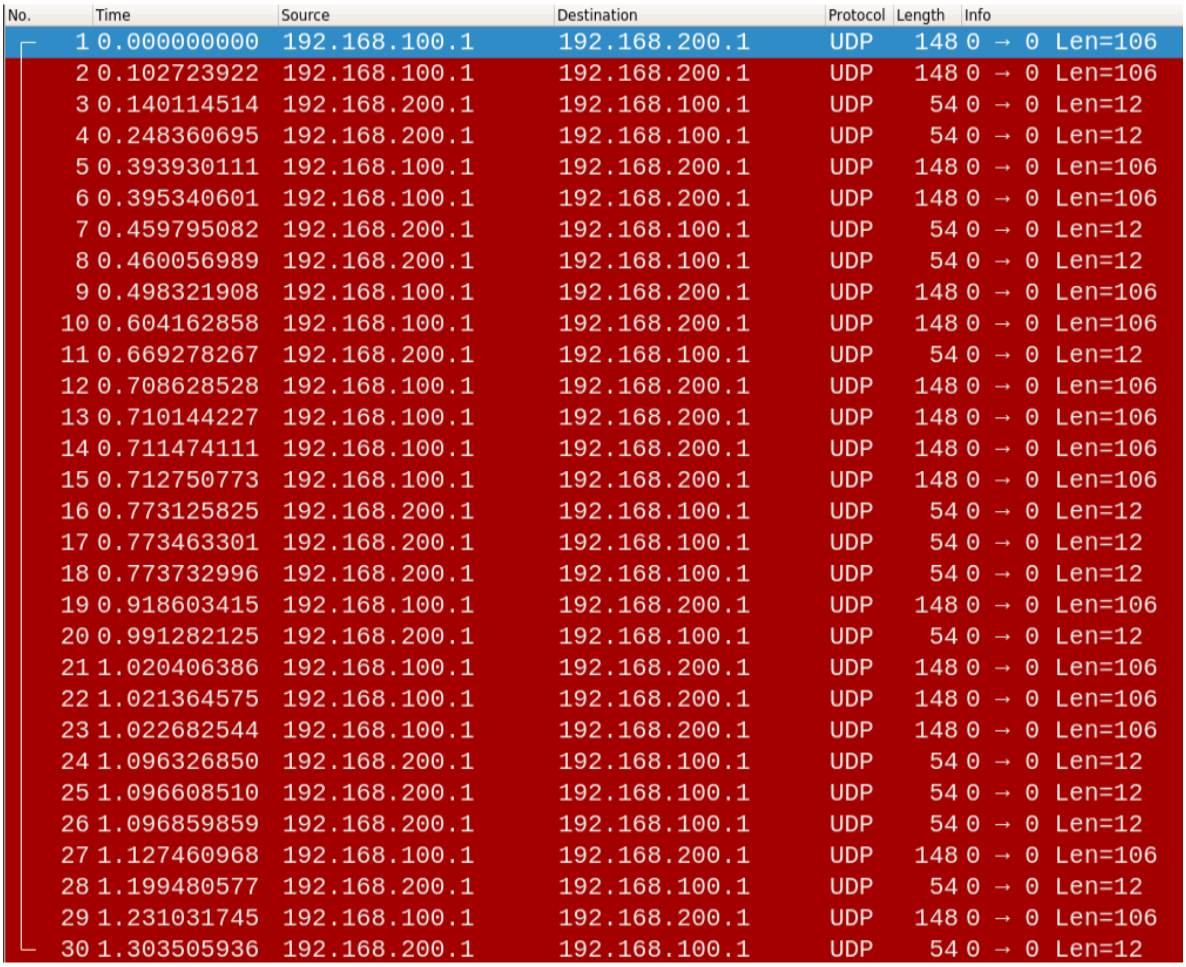
blaster：-b 192.168.200.1 -n 10 -l 100 -w 3 -t 300 -r 100

blastee：-b 192.168.100.1 -n 10

运行结果截图（包含**xterm**中输出的调试信息与**wireshark**抓包）：

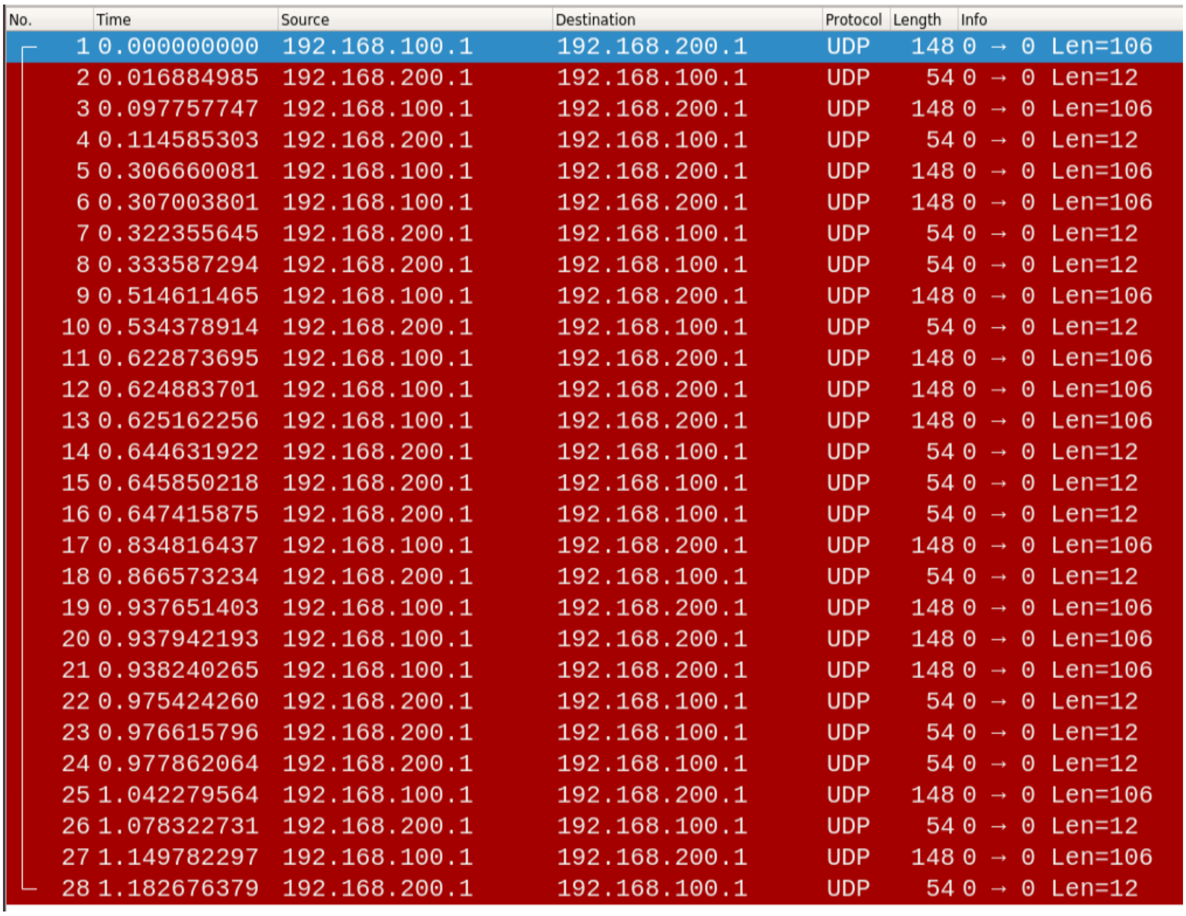
`middlebox`：

- etho:



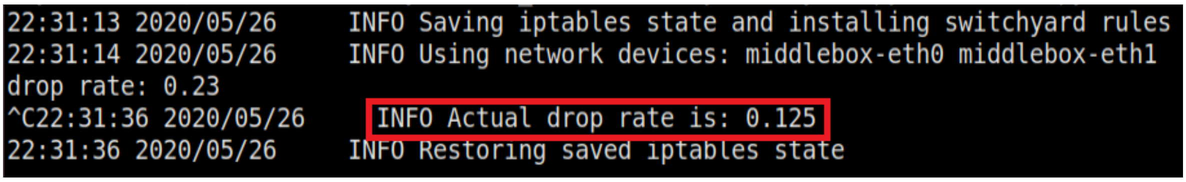
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
2	0.102723922	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
3	0.140114514	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
4	0.248360695	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
5	0.393930111	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
6	0.395340601	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
7	0.459795082	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
8	0.460056989	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
9	0.498321908	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
10	0.604162858	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
11	0.669278267	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
12	0.708628528	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
13	0.710144227	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
14	0.711474111	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
15	0.712750773	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
16	0.773125825	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
17	0.773463301	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
18	0.773732996	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
19	0.918603415	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
20	0.991282125	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
21	1.020406386	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
22	1.021364575	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
23	1.022682544	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
24	1.096326850	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
25	1.096608510	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
26	1.096859859	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
27	1.127460968	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
28	1.199480577	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
29	1.231031745	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
30	1.303505936	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12

- etho1:



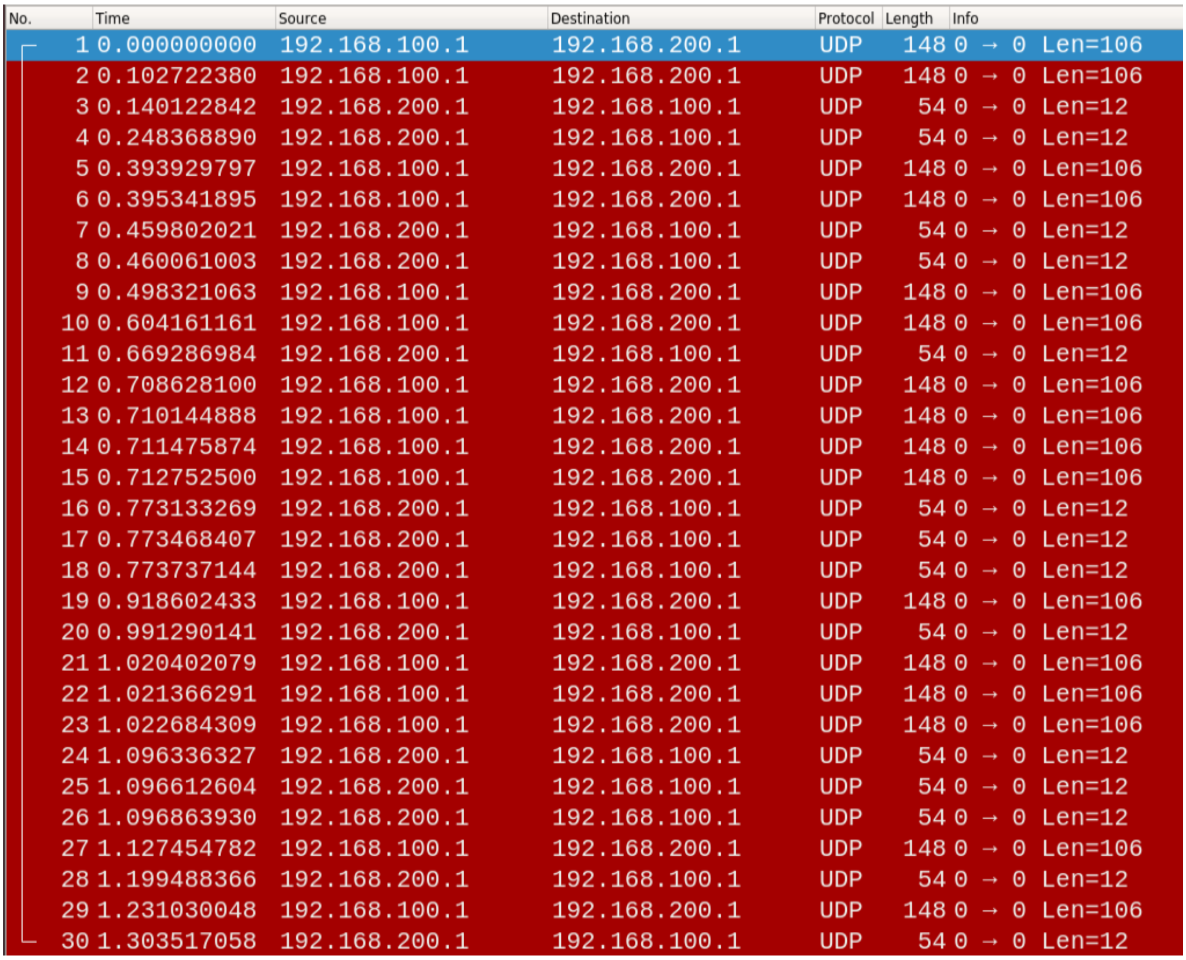
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
2	0.016884985	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
3	0.097757747	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
4	0.114585303	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
5	0.306660081	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
6	0.307003801	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
7	0.322355645	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
8	0.333587294	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
9	0.514611465	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
10	0.534378914	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
11	0.622873695	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
12	0.624883701	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
13	0.625162256	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
14	0.644631922	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
15	0.645850218	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
16	0.647415875	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
17	0.834816437	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
18	0.866573234	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
19	0.937651403	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
20	0.937942193	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
21	0.938240265	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
22	0.975424260	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
23	0.976615796	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
24	0.977862064	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
25	1.042279564	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
26	1.078322731	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
27	1.149782297	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
28	1.182676379	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12

此次实际丢包率见红框：



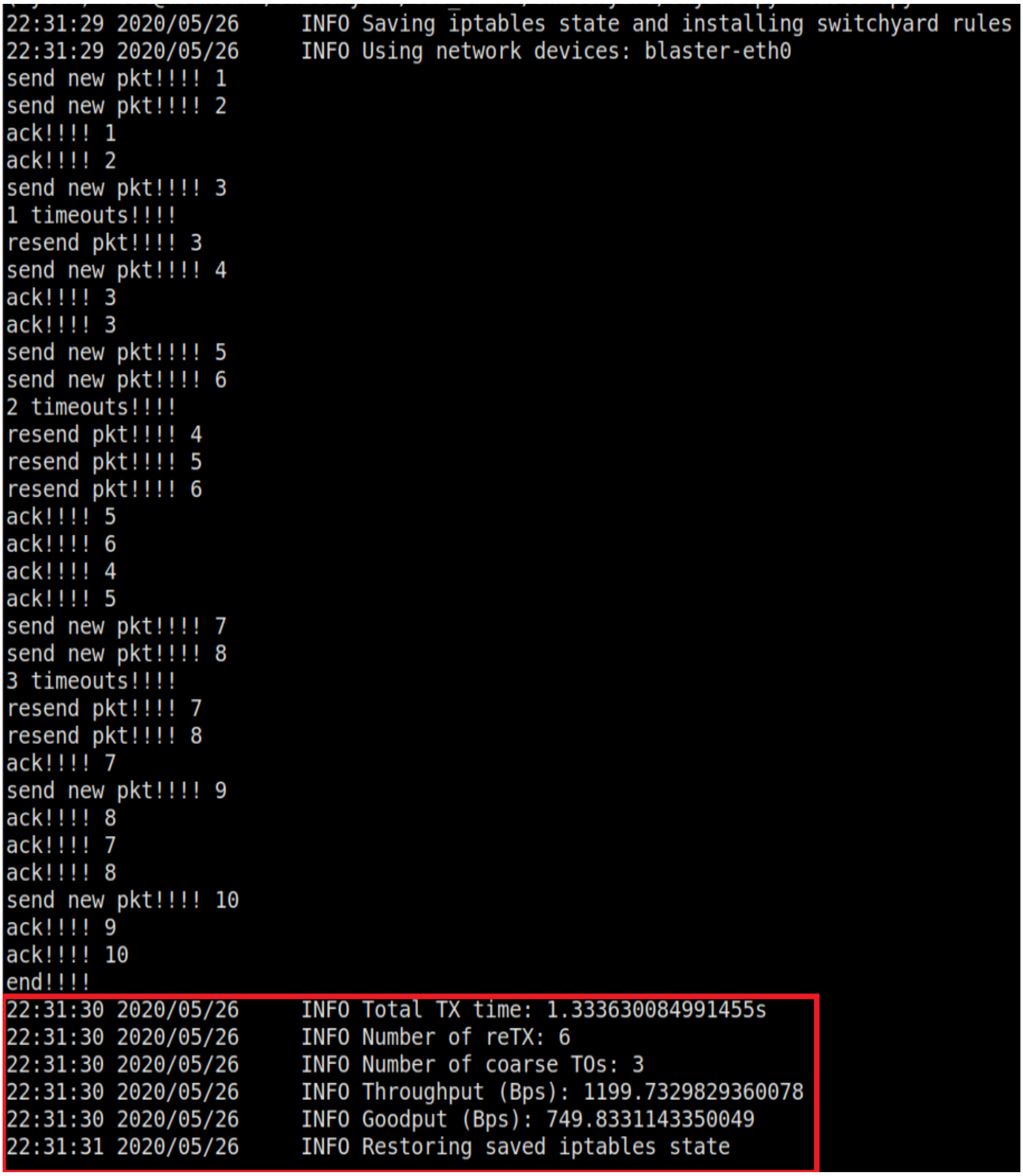
```
22:31:13 2020/05/26     INFO Saving iptables state and installing switchyard rules
22:31:14 2020/05/26     INFO Using network devices: middlebox-eth0 middlebox-eth1
drop rate: 0.23
^C22:31:36 2020/05/26     INFO Actual drop rate is: 0.125
22:31:36 2020/05/26     INFO Restoring saved iptables state
```

`blaster`：



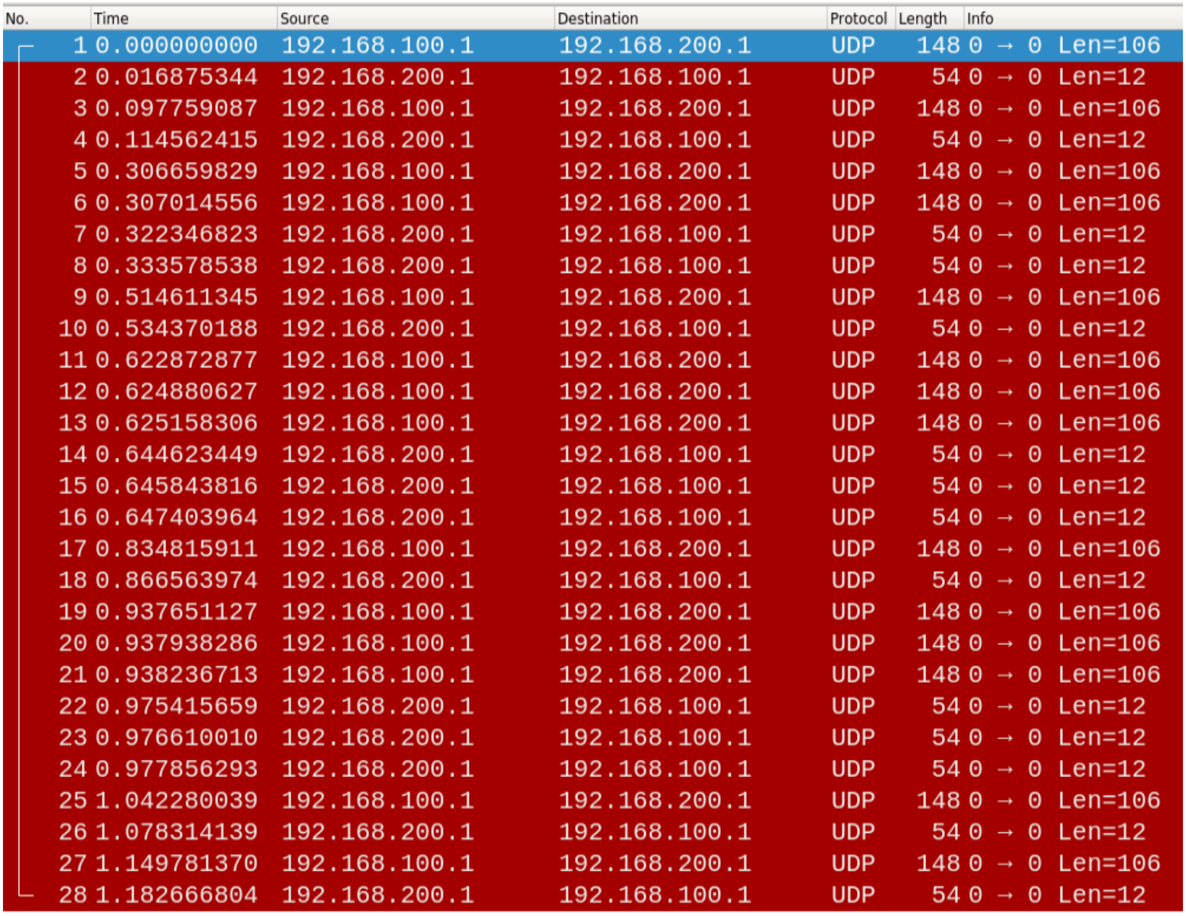
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
2	0.102722380	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
3	0.140122842	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
4	0.248368890	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
5	0.393929797	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
6	0.395341895	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
7	0.459802021	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
8	0.460061003	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
9	0.498321063	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
10	0.604161161	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
11	0.669286984	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
12	0.708628100	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
13	0.710144888	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
14	0.711475874	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
15	0.712752500	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
16	0.773133269	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
17	0.773468407	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
18	0.773737144	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
19	0.918602433	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
20	0.991290141	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
21	1.020402079	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
22	1.021366291	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
23	1.022684309	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
24	1.096336327	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
25	1.096612604	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
26	1.096863930	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
27	1.127454782	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
28	1.199488366	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
29	1.231030048	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
30	1.303517058	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12

需要输出的重要信息见红框：

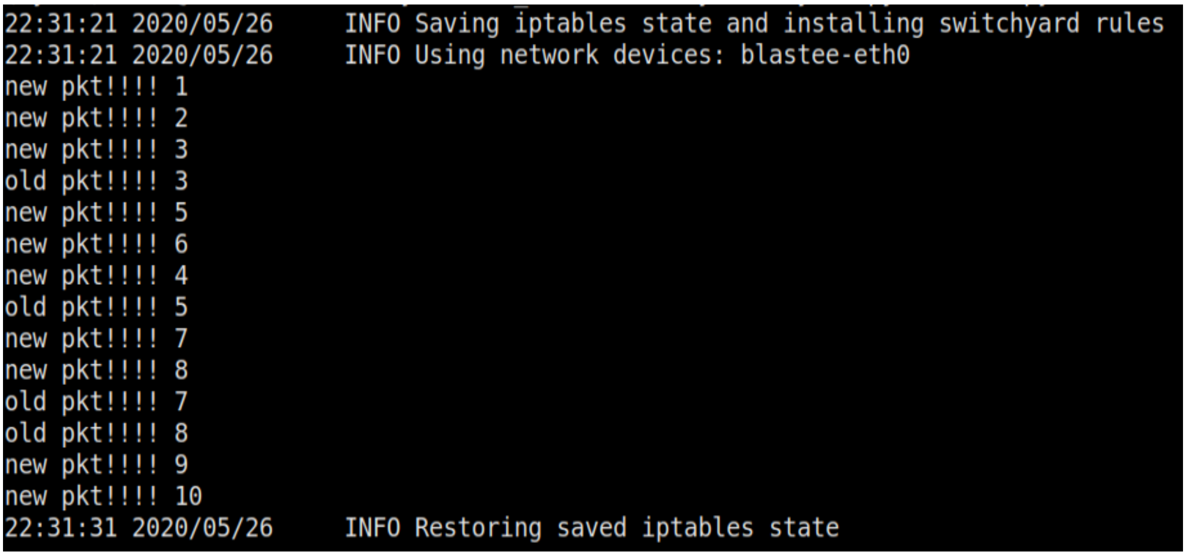


```
22:31:29 2020/05/26     INFO Saving iptables state and installing switchyard rules
22:31:29 2020/05/26     INFO Using network devices: blaster-eth0
send new pkt!!!! 1
send new pkt!!!! 2
ack!!!! 1
ack!!!! 2
send new pkt!!!! 3
1 timeouts!!!!
resend pkt!!!! 3
send new pkt!!!! 4
ack!!!! 3
ack!!!! 3
send new pkt!!!! 5
send new pkt!!!! 6
2 timeouts!!!!
resend pkt!!!! 4
resend pkt!!!! 5
resend pkt!!!! 6
ack!!!! 5
ack!!!! 6
ack!!!! 4
ack!!!! 5
send new pkt!!!! 7
send new pkt!!!! 8
3 timeouts!!!!
resend pkt!!!! 7
resend pkt!!!! 8
ack!!!! 7
send new pkt!!!! 9
ack!!!! 8
ack!!!! 7
ack!!!! 8
send new pkt!!!! 10
ack!!!! 9
ack!!!! 10
end!!!!
22:31:30 2020/05/26     INFO Total TX time: 1.333630084991455s
22:31:30 2020/05/26     INFO Number of reTX: 6
22:31:30 2020/05/26     INFO Number of coarse TOs: 3
22:31:30 2020/05/26     INFO Throughput (Bps): 1199.7329829360078
22:31:30 2020/05/26     INFO Goodput (Bps): 749.8331143350049
22:31:31 2020/05/26     INFO Restoring saved iptables state
```

`blastee`：



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
2	0.016875344	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
3	0.097759087	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
4	0.114562415	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
5	0.306659829	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
6	0.307014556	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
7	0.322346823	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
8	0.333578538	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
9	0.514611345	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
10	0.534370188	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
11	0.622872877	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
12	0.624880627	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
13	0.625158306	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
14	0.644623449	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
15	0.645843816	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
16	0.647403964	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
17	0.834815911	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
18	0.866563974	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
19	0.937651127	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
20	0.937938286	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
21	0.938236713	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
22	0.975415659	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
23	0.976610010	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
24	0.977856293	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
25	1.042280039	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
26	1.078314139	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
27	1.149781370	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
28	1.182666804	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12



```
22:31:21 2020/05/26     INFO Saving iptables state and installing switchyard rules
22:31:21 2020/05/26     INFO Using network devices: blastee-eth0
new pkt!!!! 1
new pkt!!!! 2
new pkt!!!! 3
old pkt!!!! 3
new pkt!!!! 5
new pkt!!!! 6
new pkt!!!! 4
old pkt!!!! 5
new pkt!!!! 7
new pkt!!!! 8
old pkt!!!! 7
old pkt!!!! 8
new pkt!!!! 9
new pkt!!!! 10
22:31:31 2020/05/26     INFO Restoring saved iptables state
```

过程分析：

1. blaster发出编号为1、2的包；
2. blaster收到1、2的ack；
3. blaster发出编号为3的包；
4. 第1次超时，被重传的包只有3；
5. blaster发出编号为4的包；
6. blaster收到两个3的ack，这是由于第一个**3**并未被丢包，仅仅超时；
7. blaster发出编号为5、6的包；
8. 第2次超时，被重传的包有4、5、6；
9. blaster收到5、6、4、5的ack，这是由于被重传时**5**不一定超时且未被丢包，又结合后续情况可知有一个**6**被丢包；
10. blaster发出编号为7、8的包；
11. 第3次超时，被重传的包有7、8；
12. blaster收到7的ack；

13. blaster发出编号为9的包；
14. blaster收到8、7、8的ack，理由同编号为3的包；
15. blaster发出编号为10的包；
16. blaster收到9、10的ack。
17. 传输终止，结合blastee的信息可知**编号为3、5、7、8均被收到了两次**，存在无效重传。

总结与感想

此次实验的逻辑难度大约与前两次持平，额外的难度在于掌握好3个终端的协调关系，运用全局思维进行编程。同时，此次实验除switchyard库函数外，还需要自行查阅一些python库函数用于rawbyte处理，在此过程中，也加强了实践动手能力。

在具体实现时，我也遇到了一些小问题。对于blaster，要合理设置lhs、rhs的初始值，并在之后的传输过程中善加利用（更新seq_num、判断终止条件）；对于blastee，我认为有必要做记录证明某个包是否已被收到过，不然就无法统计收到的有效包的数量，从而判断终止条件，这也算我遇到的又一个坑。

最后想说，计网的学习已经接近尾声了，但之前的章节中有不少细节都还印象模糊，赶紧利用这段时间加强吧！