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

学院	计算机科学与技术系	学号	221220085
姓名	时昌军	Email	221220085@smail.nju.edu.cn

一、实验名称

Reliable Communication

二、实验目的

在 Switchyard 中构建一个可靠的通信库，该库将由 3 个代理组成。在高层次上，**blaster**将通过**middlebox**向**blastee**发送数据包。由于IP只提供在主机之间传递数据包的尽力服务，这意味着一旦数据包进入网络，就会发生各种不好的事情：它们可能会丢失、任意延迟或重复。您的通信库将通过在 blaster 和 blastee 上实现一些基本机制来提供额外的交付保证。

您的可靠通信库将实现以下功能以提供额外的保证：

1. blastee 上每个成功接收的数据包的 ACK 机制
2. blaster上的固定尺寸滑动窗口。
3. blaster上的粗略超时以重新发送非 ACK 数据包

三、实验内容&代码

Task1: 准备

Task2: Middlebox

我们可以从 `start_mininet.py` 中读到关于端口的信息，以及 `blastee` 、 `blaster` 的 `max`、 `ip` 地址。 `middlebox` 和 `blaster` 相连的端口mac地址是 `40:00:00:00:00:01` ， `middlebox` 和 `blastee` 相连的端口mac地址是 `40:00:00:00:00:02` ， `blaster` 的mac地址是 `10:00:00:00:00:01` ， `blastee` 的mac地址是 `20:00:00:00:00:01` 。

需要处理一个特殊情况就是当 `blaster` 发给 `blastee` 时，可能会丢包，概率是 `dropRate="0.19"`

```
if fromIface == "middlebox-eth0":
    randnum = randint(1,100)
    if randnum >= self.dropRate * 100:
        packet[packet.get_header_index(Ethernet)].src = '40:00:00:00:00:02'
        packet[packet.get_header_index(Ethernet)].dst = '20:00:00:00:00:01'
        self.net.send_packet("middlebox-eth1", packet)
    else:
        log_info(f"packet is dropped.")
elif fromIface == "middlebox-eth1":
    packet[packet.get_header_index(Ethernet)].src = '40:00:00:00:00:01'
    packet[packet.get_header_index(Ethernet)].dst = '10:00:00:00:00:01'
    self.net.send_packet("middlebox-eth0", packet)
else:
    log_debug("Oops :))")
```

Task3: blastee

Blastee 的 ACK 回复的包格式应为如下结构:

<------- Switchyard headers --------> <--- Your packet header(raw bytes) --> <-- Payload in raw bytes -->

| ETH Hdr | IP Hdr | UDP Hdr | Sequence number(32 bits) | Payload (8 bytes) |

以要构造一个 `Blastee` 发送给 `Blaster` 的包，首先设置好 ETH，IP 和 UDP 包头。其中 ETH 和 IP 包头的源地址都为 `Blastee` 的 mac 地址和 ip 地址，目的地址为 `Blaster` 的 mac 地址和 ip 地址。 由 `Blaster` 发来数据包的结构可知 `packet[3]` 中的第 0 到 4 字节存放着 `Sequence number` ；第 4 到 6 字节存放着 `Length` ；第 6 字节开始存放着 `payload` 。所以在构造 `Blastee` 包的时候就要将 `Sequence number` 设置为 `packet[3]` 中的第 0 到 4 字节； `Payload` 设置为 `packet[3]` 从第 6 字节开始的 8 个字节。

```
ack_pkt=Ethernet()+IPv4(protocol=IPProtocol.UDP)+UDP()
ack_pkt[0].ethertype=EtherType.IPv4
ack_pkt[0].src=EthAddr("20:00:00:00:00:01")
ack_pkt[0].dst=EthAddr("40:00:00:00:00:02")

ack_pkt[1].ttl=64
ack_pkt[1].src=IPv4Address("192.168.200.1")
ack_pkt[1].dst=self.blasterIp
ack_pkt+=(packet[3].to_bytes()[0:4])#set sequence number

payload=packet[3].to_bytes()[6:]#set payload
length=int.from_bytes((packet[3].to_bytes()[4:6]),"big")
if length<8:
    payload+=(0).to_bytes(8-length,"big")
ack_pkt+=payload[0:8]

self.net.send_packet("blastee-eth0", ack_pkt)

seq=int.from_bytes((packet[3].to_bytes()[0:4]),"big")
if self.pkt_received[seq]==0:
    self.pkt_received[seq]=1
    self.num-=1
```

Task4: blaster

本节逻辑主要体现在是两个函数模块中，分别是 `handle_packet` 和 `handle_no_packet` 。

`handle_packet` 主要处理从 `Blastee` 发往 `Blaster` 的 ACK 包，读出收到包的 `Sequence number` 并将该序号对应的数据做好标记，表示该包已经被收到不需要被重传。并且还要及时的更新此时的 LHS 值。

```
seq=int.from_bytes(packet[3].to_bytes()[0:4],"big")
self.isAcked[seq]=1
```

```python
#check if task is finished
self.if_finished()
while self.isAcked[self.lhs]==1:
    #make sure LHS is no larger than RHS
    if self.lhs+1>self.rhs or self.lhs+1>self.num+1:
        break
    self.lhs+=1
    self.time_cnt=time.time()
    if self.lhs==self.num+1:#once the task is finished
        break
self.if_finished()
self.handle_no_packet()
```

`handle_no_packet` 主要处理内容是 `Blaster` 向 `Blastee` 发送新数据包， 并且重新发送没有收到 ACK 的数据包 首先判断 LHS 序号对应的包是否超时，如果超时需要进行一次重传；否则需要判断目前 RHS 和 LHS 的位置判断是否超过发送窗口的大小，如果没有超过则可以发送新的包，并且更新 RHS 的值

```python
if (time.time()-self.time_cnt)>self.timeout and self.isRetransmitting==False:
    #start of retransmitting
    log_info (f"coarse time out")
    self.coarseTimeout+=1
    self.isRetransmitting=True
    self.retransmit_idx=self.lhs-1

    if self.retransmit_idx<self.rhs-1:
        self.retransmit()
    if self.retransmit_idx>=self.rhs-1 or self.retransmit_idx>=self.num:#retransmission finished
        self.isRetransmitting=False
elif self.isRetransmitting==True:#still retransmit
    if self.retransmit_idx<self.rhs:
        self.retransmit()
    if self.retransmit_idx>=self.rhs-1 or self.retransmit_idx>=self.num:#retransmission finished
        self.isRetransmitting=False

#can send new packet
if self.pkt_has_sent==False:
    self.send_new_packet()
```

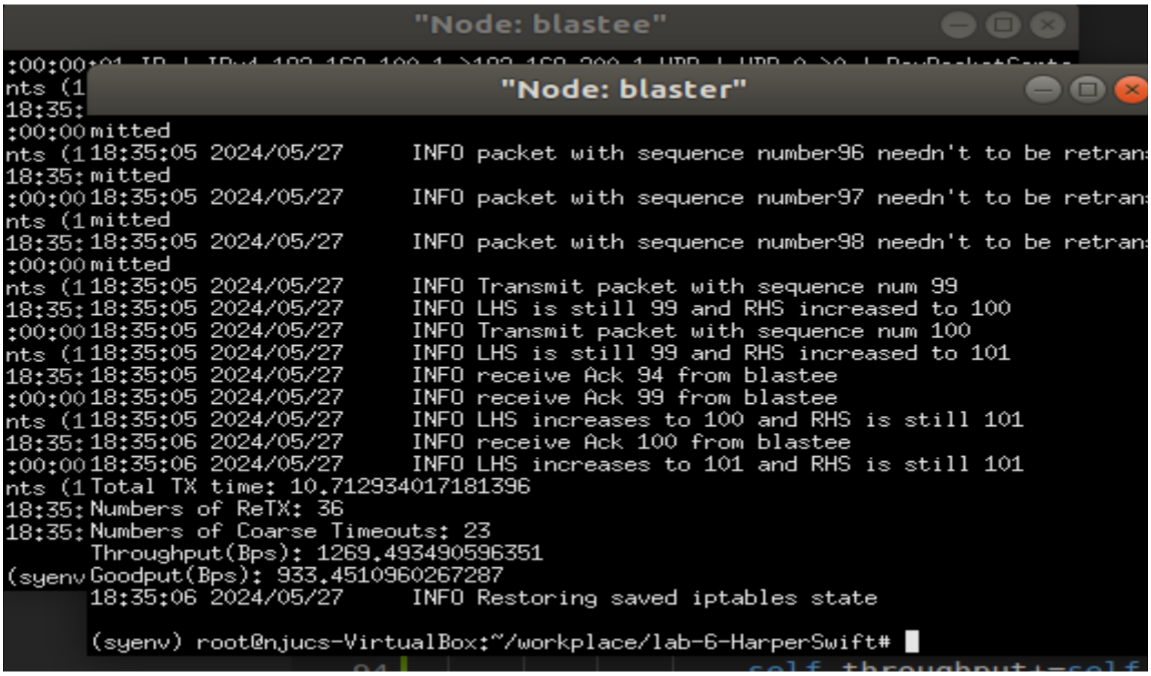
其中 `retransmit()` 负责重传以及参数的更新

```python
def retransmit(self):
    for i in range(self.retransmit_idx+1, self.rhs):#find the foremost yet to ack packet&retransmit
        self.retransmit_idx=i#the last retransmitted packet's index
        if i==self.num+1:#packet is retransmitted last
            break
        if self.isAcked[i]==0:
            self.reTX+=1
            self.isSent[i]=1
            self.throughput+=self.length
            self.net.send_packet("blaster-eth0",self.process(i))#send
            self.pkt_has_sent=True
            break
```

Task5：Running code

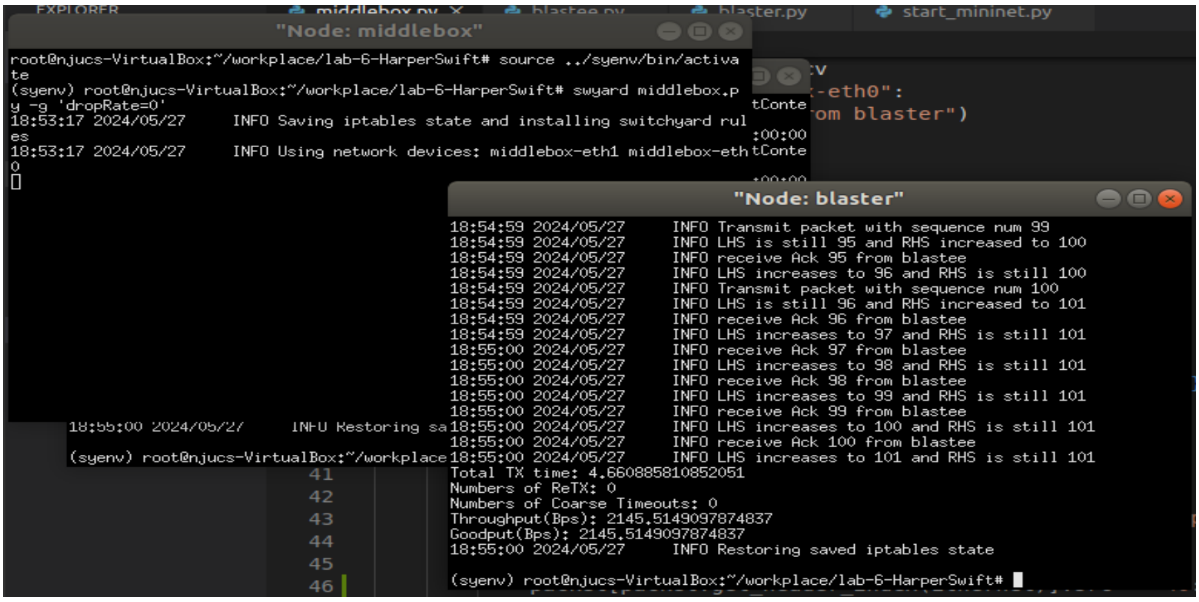
在 Mininet 中按照以下代码运行：

```
mininet> xterm middlebox
mininet> xterm blastee
mininet> xterm blaster

middlebox# swyard middlebox.py -g 'dropRate=0.19'
blastee# swyard blastee.py -g 'blasterIp=192.168.100.1 num=100'
blaster# swyard blaster.py -g 'blasteeIp=192.168.200.1 num=100 length=100 senderWindow=5 timeout=300 recvTimeout=100'
```



```
:00:00
nts (1
18:35:
:00:00 mitted
nts (1 18:35:05 2024/05/27     INFO packet with sequence number96 needn't to be retran
18:35: mitted
:00:00 18:35:05 2024/05/27     INFO packet with sequence number97 needn't to be retran
nts (1 mitted
18:35: 18:35:05 2024/05/27     INFO packet with sequence number98 needn't to be retran
:00:00 mitted
nts (1 18:35:05 2024/05/27     INFO Transmit packet with sequence num 99
18:35: 18:35:05 2024/05/27     INFO LHS is still 99 and RHS increased to 100
:00:00 18:35:05 2024/05/27     INFO Transmit packet with sequence num 100
nts (1 18:35:05 2024/05/27     INFO LHS is still 99 and RHS increased to 101
18:35: 18:35:05 2024/05/27     INFO receive Ack 94 from blastee
:00:00 18:35:05 2024/05/27     INFO receive Ack 99 from blastee
nts (1 18:35:05 2024/05/27     INFO LHS increases to 100 and RHS is still 101
18:35: 18:35:06 2024/05/27     INFO receive Ack 100 from blastee
:00:00 18:35:06 2024/05/27     INFO LHS increases to 101 and RHS is still 101
nts (1 Total TX time: 10.712934017181396
18:35: Numbers of ReTX: 36
18:35: Numbers of Coarse Timeouts: 23
       Throughput(Bps): 1269.493490596351
(syenv Goodput(Bps): 933.4510960267287
       18:35:06 2024/05/27     INFO Restoring saved iptables state

       (syenv) root@njucs-VirtualBox:~/workplace/lab-6-HarperSwift# 
```

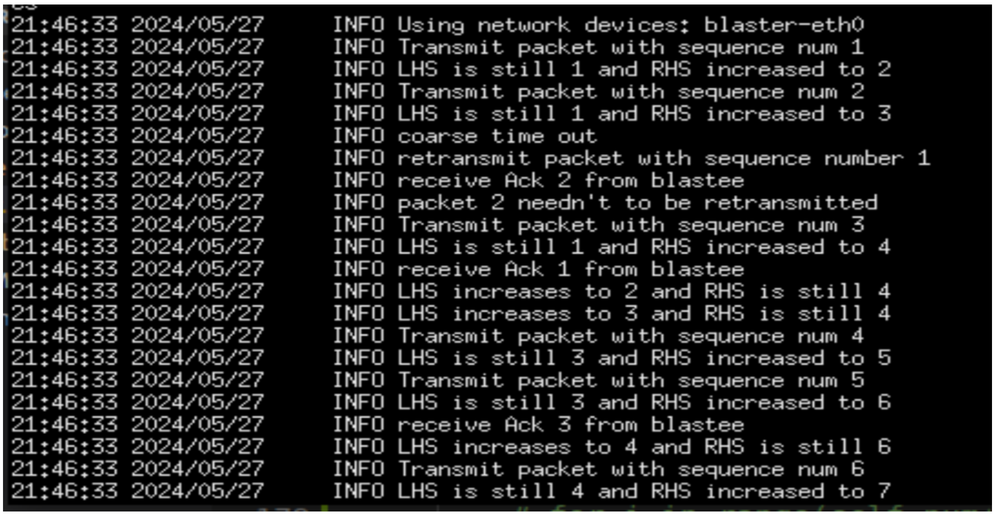
droprate改成0的情况



```
root@njucs-VirtualBox:~/workplace/lab-6-HarperSwift# source ../syenv/bin/activa
te
(syenv) root@njucs-VirtualBox:~/workplace/lab-6-HarperSwift# swyard middlebox.p
y -g 'dropRate=0'
18:53:17 2024/05/27     INFO Saving iptables state and installing switchyard rul
es
18:53:17 2024/05/27     INFO Using network devices: middlebox-eth1 middlebox-eth
0
```

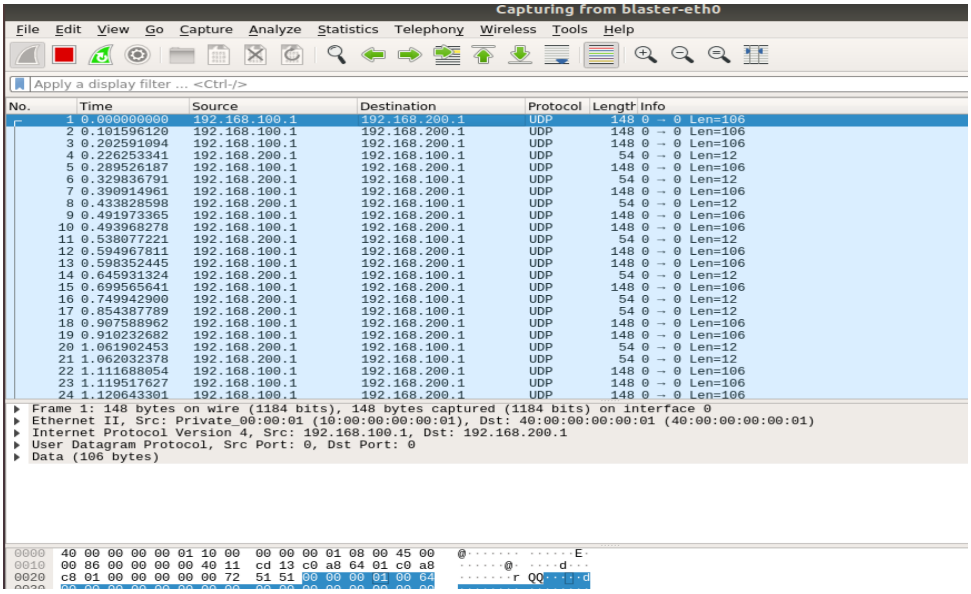
```
18:54:59 2024/05/27     INFO Transmit packet with sequence num 99
18:54:59 2024/05/27     INFO LHS is still 95 and RHS increased to 100
18:54:59 2024/05/27     INFO receive Ack 95 from blastee
18:54:59 2024/05/27     INFO LHS increases to 96 and RHS is still 100
18:54:59 2024/05/27     INFO Transmit packet with sequence num 100
18:54:59 2024/05/27     INFO LHS is still 96 and RHS increased to 101
18:54:59 2024/05/27     INFO receive Ack 96 from blastee
18:54:59 2024/05/27     INFO LHS increases to 97 and RHS is still 101
18:55:00 2024/05/27     INFO receive Ack 97 from blastee
18:55:00 2024/05/27     INFO LHS increases to 98 and RHS is still 101
18:55:00 2024/05/27     INFO receive Ack 98 from blastee
18:55:00 2024/05/27     INFO LHS increases to 99 and RHS is still 101
18:55:00 2024/05/27     INFO receive Ack 99 from blastee
18:55:00 2024/05/27     INFO LHS increases to 100 and RHS is still 101
18:55:00 2024/05/27     INFO receive Ack 100 from blastee
18:55:00 2024/05/27     INFO LHS increases to 101 and RHS is still 101
Total TX time: 4.660885810852051
Numbers of ReTX: 0
Numbers of Coarse Timeouts: 0
Throughput(Bps): 2145.5149097874837
Goodput(Bps): 2145.5149097874837
18:55:00 2024/05/27     INFO Restoring saved iptables state

(syenv) root@njucs-VirtualBox:~/workplace/lab-6-HarperSwift#
```

找一个样例进行分析：

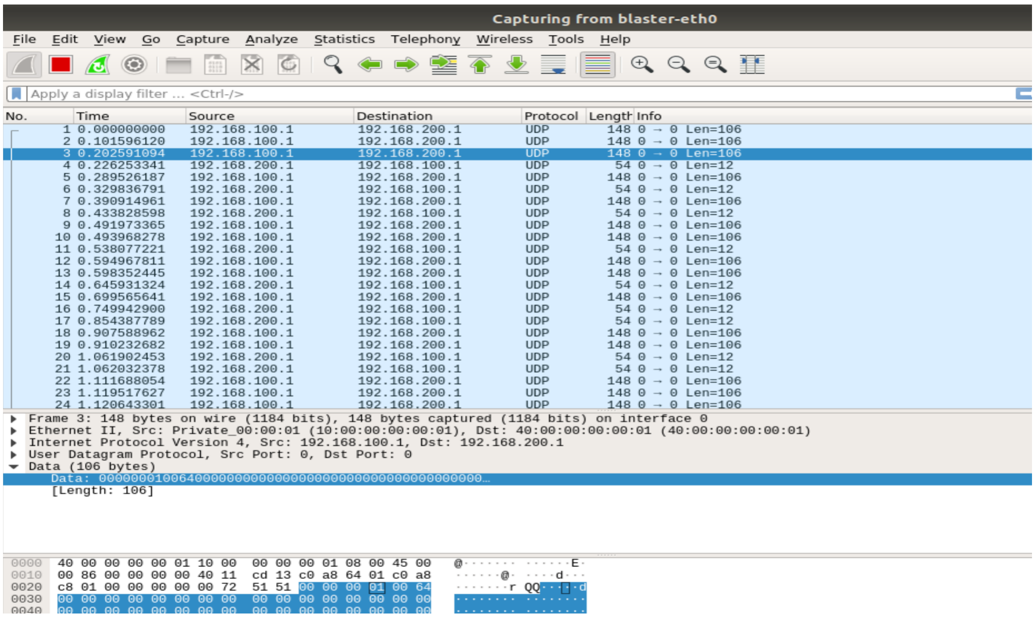


```
21:46:33 2024/05/27     INFO Using network devices: blaster-eth0
21:46:33 2024/05/27     INFO Transmit packet with sequence num 1
21:46:33 2024/05/27     INFO LHS is still 1 and RHS increased to 2
21:46:33 2024/05/27     INFO Transmit packet with sequence num 2
21:46:33 2024/05/27     INFO LHS is still 1 and RHS increased to 3
21:46:33 2024/05/27     INFO coarse time out
21:46:33 2024/05/27     INFO retransmit packet with sequence number 1
21:46:33 2024/05/27     INFO receive Ack 2 from blastee
21:46:33 2024/05/27     INFO packet 2 needn't to be retransmitted
21:46:33 2024/05/27     INFO Transmit packet with sequence num 3
21:46:33 2024/05/27     INFO LHS is still 1 and RHS increased to 4
21:46:33 2024/05/27     INFO receive Ack 1 from blastee
21:46:33 2024/05/27     INFO LHS increases to 2 and RHS is still 4
21:46:33 2024/05/27     INFO LHS increases to 3 and RHS is still 4
21:46:33 2024/05/27     INFO Transmit packet with sequence num 4
21:46:33 2024/05/27     INFO LHS is still 3 and RHS increased to 5
21:46:33 2024/05/27     INFO Transmit packet with sequence num 5
21:46:33 2024/05/27     INFO LHS is still 3 and RHS increased to 6
21:46:33 2024/05/27     INFO receive Ack 3 from blastee
21:46:33 2024/05/27     INFO LHS increases to 4 and RHS is still 6
21:46:33 2024/05/27     INFO Transmit packet with sequence num 6
21:46:33 2024/05/27     INFO LHS is still 4 and RHS increased to 7
```

首先传输数据包1，发现1没有被ack,所以重传数据包1；数据包2被接受到，但是由于1还在重传中，lhs不能更新；等到1被ack之后，lhs增加到3。后面以此类推，下面是wireshark的抓包，可以发现数据包1发送了两次，第二次受到了回复。而数据包2没有丢失，只发了一次。



```
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 0.101596120   192.168.100.1   192.168.200.1   UDP      148 0 → 0 Len=106
    3 0.202591094   192.168.100.1   192.168.200.1   UDP      148 0 → 0 Len=106
    4 0.226253341   192.168.200.1   192.168.100.1   UDP       54 0 → 0 Len=12
    5 0.289526187   192.168.100.1   192.168.200.1   UDP      148 0 → 0 Len=106
    6 0.329836791   192.168.200.1   192.168.100.1   UDP       54 0 → 0 Len=12
    7 0.390914961   192.168.100.1   192.168.200.1   UDP      148 0 → 0 Len=106
    8 0.433828598   192.168.200.1   192.168.100.1   UDP       54 0 → 0 Len=12
    9 0.491973365   192.168.100.1   192.168.200.1   UDP      148 0 → 0 Len=106
   10 0.493968278   192.168.100.1   192.168.200.1   UDP      148 0 → 0 Len=106
   11 0.538077221   192.168.200.1   192.168.100.1   UDP       54 0 → 0 Len=12
   12 0.594967811   192.168.100.1   192.168.200.1   UDP      148 0 → 0 Len=106
   13 0.598352445   192.168.100.1   192.168.200.1   UDP      148 0 → 0 Len=106
   14 0.645931324   192.168.200.1   192.168.100.1   UDP       54 0 → 0 Len=12
   15 0.699565641   192.168.100.1   192.168.200.1   UDP      148 0 → 0 Len=106
   16 0.749942900   192.168.200.1   192.168.100.1   UDP       54 0 → 0 Len=12
   17 0.854387789   192.168.200.1   192.168.100.1   UDP       54 0 → 0 Len=12
   18 0.907588962   192.168.100.1   192.168.200.1   UDP      148 0 → 0 Len=106
   19 0.910232682   192.168.100.1   192.168.200.1   UDP      148 0 → 0 Len=106
   20 1.061902453   192.168.200.1   192.168.100.1   UDP       54 0 → 0 Len=12
   21 1.062032378   192.168.200.1   192.168.100.1   UDP       54 0 → 0 Len=12
   22 1.111688054   192.168.100.1   192.168.200.1   UDP      148 0 → 0 Len=106
   23 1.119517627   192.168.100.1   192.168.200.1   UDP      148 0 → 0 Len=106
   24 1.120643301   192.168.100.1   192.168.200.1   UDP      148 0 → 0 Len=106

▸ 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: 0, Dst Port: 0
▸ Data (106 bytes)

0000  40 00 00 00 00 01 10 00  00 00 00 01 08 00 45 00   @······· ······E·
0010  00 86 00 00 00 00 40 11  cd 13 c0 a8 64 01 c0 a8   ······@· ····d···
0020  c8 01 00 00 00 00 00 72  51 51 00 00 00 01 00 64   ·······r QQ·····d
```



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	0.101596120	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
3	0.202591094	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
4	0.226253341	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
5	0.289526187	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
6	0.329836791	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
7	0.390914961	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
8	0.433828598	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
9	0.491973365	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
10	0.493968278	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
11	0.538077221	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
12	0.594967811	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
13	0.598352445	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
14	0.645931324	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
15	0.699565641	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
16	0.749942900	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
17	0.854387789	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
18	0.907588962	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
19	0.910232682	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
20	1.061902453	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
21	1.062032378	192.168.200.1	192.168.100.1	UDP	54	0 → 0 Len=12
22	1.111688054	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
23	1.119517627	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106
24	1.120643301	192.168.100.1	192.168.200.1	UDP	148	0 → 0 Len=106

```
▸ Frame 3: 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: 0, Dst Port: 0
▾ Data (106 bytes)
    Data: 0000000100640000000000000000000000000000000…
    [Length: 106]
```

```
0000  40 00 00 00 00 01 10 00  00 00 00 01 08 00 45 00   @·······  ······E·
0010  00 86 00 00 00 00 40 11  cd 13 c0 a8 64 01 c0 a8   ······@·  ····d···
0020  c8 01 00 00 00 00 00 72  51 51 00 00 00 01 00 64   ·······r  QQ·····d
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   ········  ········
```

四、实验心得

本次 LAB 相较前几次实验难度也不低，需要对课内的相关知识有掌握才能理解手册中布置的任务，比如滑动窗口相关公式等等。继续加油！