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 助教：方毓楚/于沛文/郑浩/陈伟/李国浩/李睿宸/杨溢...**
学院：计算机科学与技术系	专业(方向)：计算机科学与技术
学号：201220096	姓名：钟亚晨
Email：2991908515@qq.com	开始/完成日期：2022.05.15-2022.05.15

1.实验名称

Reliable Communication

完成清单：

> 📌完成了`middlebox`、`blaster`、`blastee`的功能逻辑；
> 📌利用Mininet和Wireshark对可靠通信库的实现正确性进行了检验（完成了所有Tasks）；
> 📌核对检验了所有FAQ，并完成了本篇实验报告；

2.实验目的

> 📌实现并通过Mininet、Wireshark仿真检验一个可靠通信库；

3.实验内容

> 📌在`blastee`上实现对于每个成功接收到的数据包的ACK机制；
> 📌在`blaster`上实现滑动窗口、用以重新发送非ACK数据包的超时机制；
> 📌在`middlebox`上实现依靠硬编码的转发功能和单向概率性丢弃数据包；

4.实验结果

简单说明，**以下对实验说明中的四个问题进行回答。**

0x01✅In the report, show how you implement the features of middlebox.

`middlebox`的逻辑概括而言只有两个部分：
①单向的，从`blaster`->`blastee`的指定概率丢包；
②硬编码的转发
实现如下：

```python
    def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
        _, fromIface, packet = recv
        if fromIface == 'middlebox-eth0':
            # log_info("Received from blaster")
            '''
            Received data packet
            Should I drop it?
            If not, modify headers & send to blastee
            '''
            # drop packet, only blaster->blastee
            # FAQ02: How to drop packet
            factor = random.random() # [0,1)
            if factor < self.dropRate:
                return None
            # forward packet blaster->blastee
            packet[Ethernet].src = '40:00:00:00:00:00'
            packet[Ethernet].dst = '20:00:00:00:00:00'
            self.net.send_packet("middlebox-eth1", packet)
        elif fromIface == "middlebox-eth1":
            # log_info("Received from blastee")
            '''
            Received ACK
            Modify headers & send to blaster. Not dropping ACK packets!
            net.send_packet("middlebox-eth0", pkt)
            '''
            # forward packet blastee->blaster
            packet[Ethernet].src = '40:00:00:00:00:00'
            packet[Ethernet].dst = '10:00:00:00:00:00'
            self.net.send_packet("middlebox-eth0", packet)
        else:
            log_info("Oops :))")
```

①26行-41行，从端口`middlebox-eth0`接收到包，即接收到来自`blaster`的包，需要转发给`blastee`。在这个传输方向上需要实现概率丢失，即35-37行，通过随机生成一个数字，与指定概率比较，如果小于则丢包，否则正常转发。这样就实现了指定概率的丢包机制；
②42行-52行，从端口`middlebox-eth1`接收到包，即接收到来自`blastee`的包，需要转发给`blaster`。这个方向上必然是接收、转发ACK，不需要概率丢包。
而两个分支的转发逻辑是一致的，即通过硬编码，修改数据包的源MAC地址的目标MAC地址，再通过相应的端口转发出去即可。
至此，实现了一个硬编码的、带有指定概率丢包机制的、用于简单转发的`middlebox`功能逻辑。

0x02✅In the report, show how you implement the features of blastee.

`blastee`的功能逻辑在于：
接收到数据包，而后根据该数据包生成ACK发出。
数据包格式为：

```
<------- Switchyard headers -----> <----- Your packet header(raw bytes) ------> <-- Payload in raw bytes --->
-------------------------------------------------------------------------------------------------------------
|  ETH Hdr |  IP Hdr  |  UDP Hdr  | Sequence number(32 bits) | Length(16 bits) |   Variable length payload  |
-------------------------------------------------------------------------------------------------------------
```

ACK格式为：

```
<------- Switchyard headers -----> <----- Your packet header(raw bytes) ------> <-- Payload in raw bytes --->
-------------------------------------------------------------------------------------------------------------
|  ETH Hdr |  IP Hdr  |  UDP Hdr  |         Sequence number(32 bits)         |      Payload  (8 bytes)    |
-------------------------------------------------------------------------------------------------------------
```

而实现思路就相对而言较为容易：
①正常生成并填写`Ethernet Header`、`IPv4 Header`等，各种源地址、目的地址都通过硬编码

方式提供；
②从收到的包中取出`Sequence number`作为ACK包中的`Sequence number`进行追加；
③根据收到包中的`Length`，来取出收到包中有效载荷的前8个字节（不足8个则利用全0进行填充），作为ACK的定长`Payload`；
④从唯一一个端口将该ACK发送出去
实现代码为：

```python
    def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
        _, fromIface, packet = recv
        # log_info(f"Pkt: {packet}")
        # log_info(f"{type(packet[3])}")

        # Packet: | Ethernet | IPv4 | UDP | sequence_num(4bytes) | length(2bytes) | vpayload |
        # -> ACK: | Ethernet | IPv4 | UDP | sequence_num(4bytes) | payload(8bytes) |

        # build headers and set address
        ack = Ethernet() + IPv4() + UDP()
        # finish Etheret header
        ack[Ethernet].src = '20:00:00:00:00:00'
        ack[Ethernet].dst = '40:00:00:00:00:00'
        # finish IPv4 header
        ack[IPv4].src = self.ip
        ack[IPv4].dst = self.targetip
        ack[IPv4].protocol = IPProtocol.UDP
        ack[IPv4].ttl = 64

        seq_num = int.from_bytes(packet[3].to_bytes()[0:4], 'big')
        log_info(f"I got a packet from {fromIface} which sequence_num is {seq_num}")

        # build sequence number
        ack += packet[3].to_bytes()[0:4]
        # build 8bytes payload
        freelen = int.from_bytes(packet[3].to_bytes()[4:6], 'big')
        if freelen >= 8:
            ack += packet[3].to_bytes()[6:14]
        else:
            ack += packet[3].to_bytes()[6:] + bytes([0 for _ in range(8 - freelen)])

        # send packet
        self.net.send_packet('blastee-eth0', ack)
```

各种逻辑已通过图中注释进行了详细说明，这里不再进行赘述

0x03✅In the report, show how you implement the features of blaster.

`blaster`的逻辑作为整个实验中最为繁多、复杂的部分，可以概括为以下四部分：
①指定数量的发包以及自定义的有效载荷内容；
②滑动窗口，最为核心的部分；
③伪超时机制，超时重传；
④对于发包情况的统计；
实现如下：
①指定数量的发包以及自定义的有效载荷内容：

```
19              num,
20              length="100",
21              senderWindow="5",
22              timeout="300",
23              recvTimeout="100"
24      ):
25          self.net = net
26          # TODO: store the parameters
27          self.ip = '192.168.100.1'   # FAQ01: hardcode
28          self.targetip = blasteeIp
29          self.send_num = int(num)
30          self.try_num = int(num)
31          self.pkt_payload_maxlen = int(length)
32          # SW
33          self.swlen = int(senderWindow)
34          self.lhs = 1
35          self.rhs = 1
36          # Simulation Window
37          self.acked = []     # 0 : not acked, 1 : acked
38          # Timeout
39          self.timeout = float(timeout) / 1000.0    # 300ms
40          self.recv_timeout = float(recvTimeout) / 1000.0    # 100ms
41          self.timer = 0
42
43          # statistic info
44          self.begin_time = 0
45          self.total_tx_time = 0
46          self.number_retx = 0
47          self.number_timeout = 0
48          self.goodtx_bytes = 0       # bytes number of successfully trans
49          self.retx_bytes = 0         # bytes number of retrans
```

```
153     def start(self):
154         '''A running daemon of the blaster.
155         Receive packets until the end of time.
156         '''
157         while True:
158             # Task Done
159             log_info(f"Now send_num is {self.send_num}")
160             if self.send_num <= 0:
161                 break
162
163             try:
164                 recv = self.net.recv_packet(timeout=self.recv_timeout)     # 1
165             except NoPackets:
166                 self.handle_no_packet()
167                 continue
168             except Shutdown:
169                 break
170
171             self.handle_packet(recv)
172         # Show Statistic Info
173         self.show_info()
174
175         self.shutdown()
176
177     def shutdown(self):
178         self.net.shutdown()
179
```

不难看到，在第29行，初始化了一个`send_num`属性，用于指示剩余需要发送的数量，并且在每有一个包发出且被ACKed后，该属性将会减1，而在第159-161行，如果`send_num`减至0时，则会退出消息循环，从而结束当前的逻辑，完成发包任务；

而自定义的有效载荷这里则通过直接创建指定`variable payload`长度的全0bytes来填充：

```
109             data = bytes(self.pkt_payload_maxlen)
110             pkt += len(data).to_bytes(2, byteorder='big')
```

若要获得更加仿真的发送，则可以将一篇小作文或者是一份代码存放到一个数组当中，而后基于该数组进行滑动窗口，而这样的实现过于繁琐，且考虑到关键在于各个逻辑的实现，该部分属于

可有可无的用于适应实际问题的多变部分，因此这里进行简化处理。

②滑动窗口的实现

同样在第32-37行，此处定义了有关滑动窗口的属性，包括了LHS、RHS以及指定的滑动窗口长度，和一个记录列表，列表中每个元素的下标+LHS值表示对应报文的序列号，而每个元素的值表示该包是否被ACKed，如果为1则被ACKed，否则未被ACKed，该列表可以作为滑动窗口的表征。

当满足条件RHS - LHS + 1 < SWLEN时，则可以发送新的报文，否则等待ACK将滑动窗口腾出位置给新报文

```
93            if self.rhs - self.lhs + 1 < self.swlen and self.try_num > 0:
94                self.send_new_pkt(pkt)
95                self.try_num -= 1
96            elif(time.time() - self.timer >= self.timeout):
97                self.repeat_pkts(pkt)
98            else:
99                pass
```

而当发送新报文时，则将RHS向右移动

```
101       def send_new_pkt(self, pkt): # + FAQ06: one packet each time
102           '''SW is not full, send new packet'''
103           log_info(f"I'm send new packet {self.rhs}")
104           # Add SequenceNum
105           pkt += self.rhs.to_bytes(4, byteorder='big')
106           # Add Payload Length
107           data = f"now I send you {self.rhs}th packet".encode(encoding='UTF-8')
108           pkt += len(data).to_bytes(2, byteorder='big')
109           # Add variable payload
110           pkt += data
111           # Update SW
112           self.rhs += 1
113           self.acked.append(0)
```

当接收到ACK时，从头遍历滑动窗口的每一项，来更新LHS，遇到未被ACKed（即对应元素为0）的则停止，最后则检查LHS是否被更新，如果更新了则对应更新用于超时机制的计时器，另外更新`acked`列表，去除左边连续的已被ACKed的部分（非常模拟滑动窗口）

```
52        def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
53            '''Receive ACK from blastee'''
54            _, fromIface, packet = recv
55            # Update statistic info
56            self.total_tx_time = time.time() - self.begin_time
57            # Get seqnum and update SW
58            seqnum = int.from_bytes(packet[3].to_bytes()[:4], byteorder='big')
59            if seqnum < self.lhs:
60                return None
61            log_info(f"I got a packet seqnum:{seqnum} LHS:{self.lhs} RHS:{self.rhs}")
62            # Some packet Acked
63            self.acked[seqnum-self.lhs] = 1
64            if self.acked[0] == 1:
65                self.timer = time.time()
66            # Try to move LHS
67            ori_lhs = self.lhs
68            while self.lhs - ori_lhs + 1 <= len(self.acked) and self.acked[self.lhs - ori_lhs] == 1:
69                self.lhs += 1
70                # Decrease send_num
71                self.send_num -= 1
72            # Update timer only when LHS moved + FAQ03
73            if(self.lhs != ori_lhs):
74                self.timer = time.time()
75            # Update SW
76            self.acked = self.acked[self.lhs - ori_lhs:]
```

③伪超时机制

当触发超时机制时，则遍历一遍`acked`，将所有状态为0的表示未ACKed的报文进行重传。

```python
def repeat_pkts(self, pkt):
    '''Timeout, Repeat Packets'''
    log_info(f"I'm repeating original packet {self.lhs}")
    # FAQ03: How timeout work?
    for index, val in enumerate(self.acked):
        # If not ACKed, repeat it!
        if val == 0:
            # Add SequenceNum
            pkt += (self.lhs + index).to_bytes(4, byteorder='big')
            # Add Payload Length
            data = f"now I resend you {self.lhs + index}th packet".encode(encoding='UTF-8')
            pkt += len(data).to_bytes(2, byteorder='big')
            # Add variable payload
            pkt += data
            # Send Packet and reset timer
            self.net.send_packet('blaster-eth0', pkt)
            # Update statistic info
            self.number_retx += 1
            self.retx_bytes += len(data)
    # Update statistic info
    self.number_timeout += 1
```

④对于发包情况的统计

```python
            # Initialize statistic info
            if self.rhs == 2:
                self.timer = time.time()
                self.begin_time = time.time()
            self.goodtx_bytes += len(data)
```

这着重需要指出的是，当RHS更新为2时，即刚刚发出第一个新包时，这时初始化超时机制计时器以及用于记录“从发送第一个包到接收最后一个ACK的时间”的变量。

而后其它部分则穿插在源代码中，对应的都有名为`Update Statistic info`的注释，可自行查看。

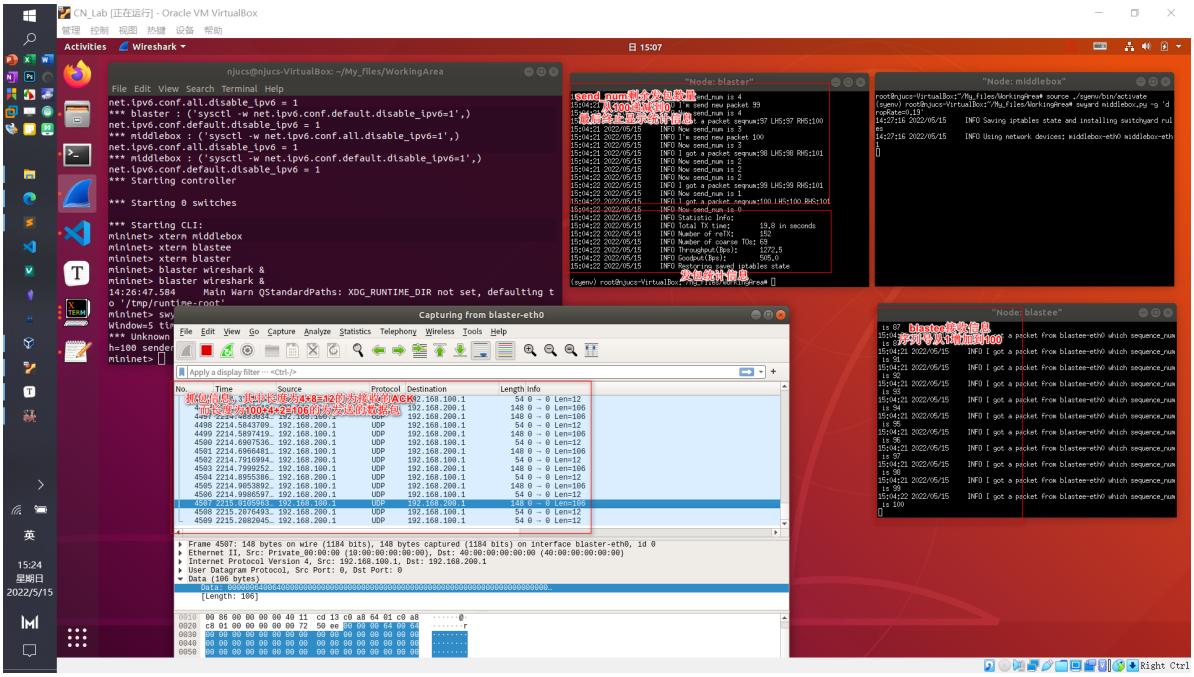
最后通过一个封装函数来显示所有统计信息：

```python
    def show_info(self):
        '''show statistic information'''
        log_info("Statistic Info:")
        log_info(f'Total TX time:        {self.total_tx_time:.1f} in seconds')
        log_info(f'Number of reTX:       {self.number_retx}')
        log_info(f'Number of coarse TOs: {self.number_timeout}')
        log_info(f'Throughput(Bps):      {(self.goodtx_bytes + self.retx_bytes) / self.total_tx_time:.1f}')
        log_info(f'Goodput(Bps):         {self.goodtx_bytes / self.total_tx_time:.1f}')

```

oxo4 ✅ Write the procedure and analysis in your report with screenshots.

首先利用实验说明所给的参数来从`blaster`向`blastee`发送100个数据包，获取到如下统计信息和输出，可以看到符合预期，`blaster`方的剩余发包数量从100递减到0，然后停止发包，打印出统计信息，而`blastee`接收方，接受到的数据包的序列号则从1到100，交错形式递增（有可能序列号大的包先被接收到，因为`middlebox`的指定概率丢包存在）



这里使用了`Wireshark`在`blaster`的端口进行抓包，可以查看获取到的数据包信息：

Capturing from

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ··· <Ctrl-/>

No.	Time	Source	Protocol	Destination	Length	Info
4483	2213.7542492…	192.168.100.1	UDP	192.168.200.1	254	0 → 0 Len=212
4484	2213.8476423…	192.168.200.1	UDP	192.168.100.1	54	0 → 0 Len=12
4485	2213.8479695…	192.168.200.1	UDP	192.168.100.1	54	0 → 0 Len=12
4486	2213.8567131…	192.168.100.1	UDP	192.168.200.1	148	0 → 0 Len=106
4487	2213.9514011…	192.168.200.1	UDP	192.168.100.1	54	0 → 0 Len=12
4488	2213.9665859…	192.168.100.1	UDP	192.168.200.1	148	0 → 0 Len=106
4489	2214.0556265…	192.168.200.1	UDP	192.168.100.1	54	0 → 0 Len=12
4490	2214.0560193…	192.168.200.1	UDP	192.168.100.1	54	0 → 0 Len=12
4491	2214.0736375…	192.168.100.1	UDP	192.168.200.1	148	0 → 0 Len=106
4492	2214.1668536…	192.168.200.1	UDP	192.168.100.1	54	0 → 0 Len=12
4493	2214.1794602…	192.168.100.1	UDP	192.168.200.1	148	0 → 0 Len=106
4494	2214.2630020…	192.168.200.1	UDP	192.168.100.1	54	0 → 0 Len=12
4495	2214.3711144…	192.168.200.1	UDP	192.168.100.1	54	0 → 0 Len=12
4496	2214.3848348…	192.168.100.1	UDP	192.168.200.1	148	0 → 0 Len=106
4497	2214.4883034…	192.168.100.1	UDP	192.168.200.1	148	0 → 0 Len=106
4498	2214.5843709…	192.168.200.1	UDP	192.168.100.1	54	0 → 0 Len=12
4499	2214.5897419…	192.168.100.1	UDP	192.168.200.1	148	0 → 0 Len=106
4500	2214.6907536…	192.168.200.1	UDP	192.168.100.1	54	0 → 0 Len=12
4501	2214.6966481…	192.168.100.1	UDP	192.168.200.1	148	0 → 0 Len=106
4502	2214.7916994…	192.168.200.1	UDP	192.168.100.1	54	0 → 0 Len=12
4503	2214.7999252…	192.168.100.1	UDP	192.168.200.1	148	0 → 0 Len=106
4504	2214.8955386…	192.168.200.1	UDP	192.168.100.1	54	0 → 0 Len=12
4505	2214.9053892…	192.168.100.1	UDP	192.168.200.1	148	0 → 0 Len=106
4506	2214.9986597…	192.168.200.1	UDP	192.168.100.1	54	0 → 0 Len=12
4507	2215.0105963…	192.168.100.1	UDP	192.168.200.1	148	0 → 0 Len=106
4508	2215.2076493…	192.168.200.1	UDP	192.168.100.1	54	0 → 0 Len=12
4509	2215.2082045…	192.168.200.1	UDP	192.168.100.1	54	0 → 0 Len=12

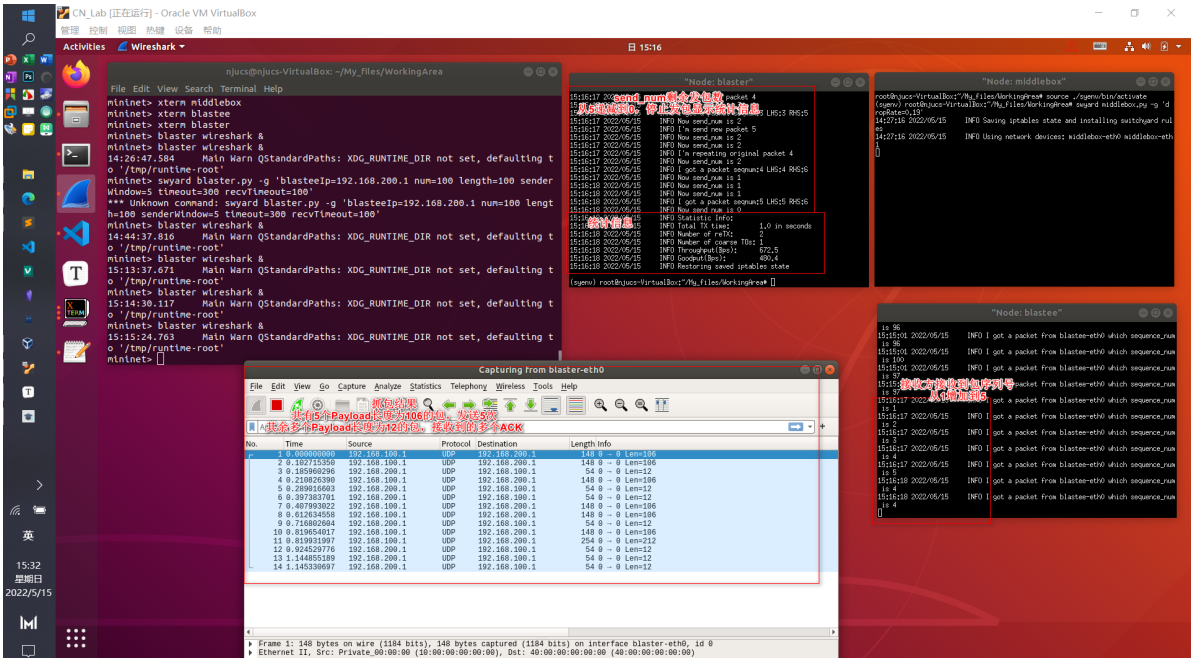
```
▸ Frame 4507: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface blaster-eth0, id 0
▸ Ethernet II, Src: Private_00:00:00 (10:00:00:00:00:00), Dst: 40:00:00:00:00:00 (40:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 192.168.100.1, Dst: 192.168.200.1
▸ User Datagram Protocol, Src Port: 0, Dst Port: 0
▾ Data (106 bytes)
    Data: 0000006400640000000000000000000000000000000000000000000000000000000000000000…
    [Length: 106]
```

```
0000  40 00 00 00 00 00 10 00  00 00 00 00 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  50 ee 00 00 00 64 00 64   ·····r  P····d·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   ····    ····
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ····    ····
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ····    ····
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ····    ····
0080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ····    ····
0090  00 00 00 00                                       ····
```

Data (data.data), 106 bytes

可以看到发送的数据包的发送方、接收方地址均正确，且有效载荷部分均为106（4bytes sequence_num + 2bytes Length + 100bytes Data）
而接收到的ACK的发送方、接收方地址同样正确，且有效载荷均为12（4bytes sequence_num + 8bytes payload）

另外，我们还可以使用同样的参数，而尝试仅限制发送5个数据包来进行更进一步的分析：



通过图示分析可以知道，行为仍然符合预期。
至此，我们通过利用`Mininet`和`Wireshark`进行方正模拟测试验证了所实现的可靠传输库功能的正确性。

5.核心代码

blaster.py，更多代码详见源代码文件。

```
class Blaster:
    def __init__(
            self,
            net: switchyard.llnetbase.LLNetBase,
            blasteeIp,
            num,
            length="100",
            senderWindow="5",
            timeout="300",
            recvTimeout="100"
    ):
        self.net = net
        # TODO: store the parameters
        self.ip = '192.168.100.1'    # FAQ01: hardcode
        self.targetip = blasteeIp
        self.send_num = int(num)
        self.try_num = int(num)
        self.pkt_payload_maxlen = int(length)
```

```
        # SW
        self.swlen = int(senderWindow)
        self.lhs = 1
        self.rhs = 1
        # Simulation Window
        self.acked = []     # 0 : not acked, 1 : acked
        # Timeout
        self.timeout = float(timeout) / 1000.0    # 300ms
        self.recv_timeout = float(recvTimeout) / 1000.0    # 100ms
        self.timer = 0

        # statistic info
        self.begin_time = 0
        self.total_tx_time = 0
        self.number_retx = 0
        self.number_timeout = 0
        self.goodtx_bytes = 0       # bytes number of successfully trans
        self.retx_bytes = 0         # bytes number of retrans


    def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
        '''Receive ACK from blastee'''
        _, fromIface, packet = recv
        # Update statistic info
        self.total_tx_time = time.time() - self.begin_time
        # Get seqnum and update SW
        seqnum = int.from_bytes(packet[3].to_bytes()[:4], byteorder='big')
        if seqnum < self.lhs:
            return None
        log_info(f"I got a packet seqnum:{seqnum} LHS:{self.lhs} RHS:{self.rhs}")
        # Some packet Acked
        self.acked[seqnum-self.lhs] = 1
        if self.acked[0] == 1:
            self.timer = time.time()
        # Try to move LHS
        ori_lhs = self.lhs
        while self.lhs - ori_lhs + 1 <= len(self.acked) and self.acked[self.lhs - ori_lhs] == 1:
            self.lhs += 1
            # Decrease send_num
            self.send_num -= 1
        # Update timer only when LHS moved + FAQ03
        if(self.lhs != ori_lhs):
            self.timer = time.time()
        # Update SW
        self.acked = self.acked[self.lhs - ori_lhs:]


```

```
66      def handle_no_packet(self):
67          '''Not received ACK from blastee, so send packet'''
68          # log_info("Didn't receive anything")
69          # Creating the headers for the packet
70          pkt = Ethernet() + IPv4() + UDP()
71          # Finish Ethernet header
72          pkt[Ethernet].src = '10:00:00:00:00:00'
73          pkt[Ethernet].dst = '40:00:00:00:00:00'
74          # Finish IPv4 header
75          pkt[IPv4].src = self.ip
76          pkt[IPv4].dst = self.targetip
77          pkt[IPv4].ttl = 64
78          pkt[IPv4].protocol = IPProtocol.UDP
79
80          if self.rhs - self.lhs + 1 < self.swlen and self.try_num > 0:
81              self.send_new_pkt(pkt)
82              self.try_num -= 1
83          elif(time.time() - self.timer >= self.timeout):
84              self.repeat_pkts(pkt)
85          else:
86              pass
87
88      def send_new_pkt(self, pkt): # + FAQ06: one packet each time
89          '''SW is not full, send new packet'''
90          log_info(f"I'm send new packet {self.rhs}")
91          # Add SequenceNum
92          pkt += self.rhs.to_bytes(4, byteorder='big')
93          # Add Payload Length
94          data = f"now I send you {self.rhs}th
    packet".encode(encoding='UTF-8')
95          pkt += len(data).to_bytes(2, byteorder='big')
96          # Add variable payload
97          pkt += data
98          # Update SW
99          self.rhs += 1
100         self.acked.append(0)
101         # Send Packet
102         self.net.send_packet('blaster-eth0', pkt)
103         # Initialize statistic info
104         if self.rhs == 2:
105             self.timer = time.time()
106             self.begin_time = time.time()
107         self.goodtx_bytes += len(data)
108
109     def repeat_pkts(self, pkt):
110         '''Timeout, Repeat Packets'''
111         log_info(f"I'm repeating original packet {self.lhs}")
112         # FAQ03: How timeout work?
113         for index, val in enumerate(self.acked):
114             # If not ACKed, repeat it!
115             if val == 0:
116                 # Add SequenceNum
```

```
                pkt += (self.lhs + index).to_bytes(4, byteorder='big')
                # Add Payload Length
                data = f"now I resend you {self.lhs + index}th
packet".encode(encoding='UTF-8')
                pkt += len(data).to_bytes(2, byteorder='big')
                # Add variable payload
                pkt += data
                # Send Packet and reset timer
                self.net.send_packet('blaster-eth0', pkt)
                # Update statistic info
                self.number_retx += 1
                self.retx_bytes += len(data)
        # Update statistic info
        self.number_timeout += 1
```

6.总结与感想

总结:

> 通过实践了解了滑动窗口的工作机制以及伪超时机制;
> 可靠传输的实现是通过双方面实现的，而非单方面实现，重传和ACK都是必要部分，协同工作以实现可靠传输;
> 对于Python语法进行了一些拓展，学习到额外的有关Python语言的知识;

感想:

> 对于难以使用Switchyard本身提供的testscenario的场景下，利用Mininet和Wireshark模拟仿真测试和分析显得尤为重要;
> 可靠传输的实现过程中会出现诸多错误，而在`pdb`当中使用`where`指令能够快速定位错误;
> 阅读英文文档可以方便找到所需要的接口以供调用，技术文档的阅读查找能力在多次实验中都显得尤为重要;