Lab Report

Lab Name Reliable Communication

Course	Computer Network
Major	Computer Science and Technology
Id	191220129
Name	Shangyu.Xing
Email	191220129@smail.nju.edu.cn
Date	2021.05

Contents

1 Objective

- Learn reliable communication and how to implement it;
- learn to implement hardware logic using the Switchyard framework;
- learn to capture network package using wireshark.

2 Requirements

This lab requires to implement a simplified network which achieves reliable communication. It has 3 main components:

- MiddleBox:

  forward packets directly from one interface to another;

  drop packets according to dropRate.

- blastee: send a corresponding ack upon receiving a packet from blaster;
- blaster:

  send packets;

  receive acks and maintain a sender window;

  retransmit if timeout;

  do statistics.

3 Procedure

I completed all the tasks as required. In this section, I will explain how I did my work in detail.

3.1 Packet Structure

Before coming to the detailed implementation, I'd like to introduce my packet structure. A packet blaster send out or an ack blastee send out consists of 4 parts (or has 4 headers) – Ethernet, IPv4, UDP and RawPacketContent. The differences are in RawPacketContent, as illustrated below:

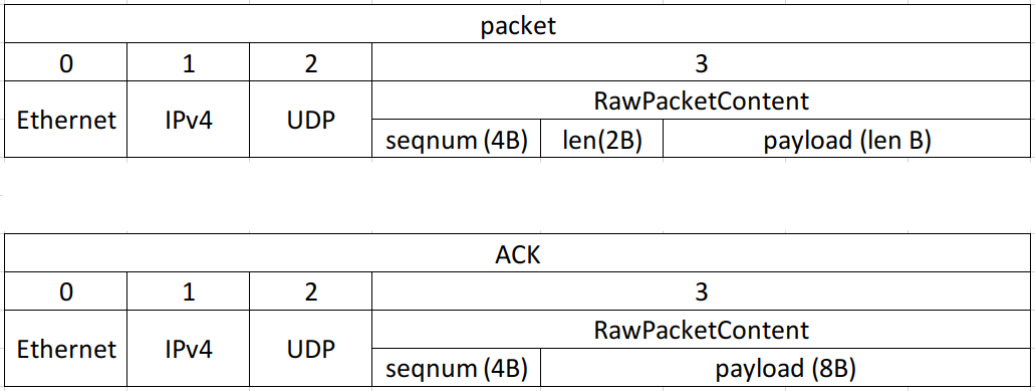

Figure 1: packet structure

3.2 MiddleBox

MiddleBox should do the following when receive a packet:

1. Check where the packet comes from;
2. if it is from blastee, forward it to blaster (only srcmac and dstmac should be modified);
3. if it is from blaster, get a random value in [0,1] and drop the packet if it is below dropRate, otherwise forward it.

Uncertain behavior introduced by random value means a painful debugging process. To make the result reproduceable, I manually set the random seed to 0.

```
def handle_packet(self, recv: switchyard.llnetbase.
    ReceivedPacket):
        _, fromIface, packet = recv
        if packet.get_header_index(Arp) != -1:
            log_info('\033[1;33mreceive an arp?!\033[0m') # ]]
            return
        if fromIface == "middlebox-eth0":
            log_info(f"Received from blaster: {packet}")
            if random() > self.dropRate:  # not drop
                heth_idx = packet.get_header_index(Ethernet)
                packet[heth_idx].src = blastee_intf_mac
                packet[heth_idx].dst = blastee_mac
                log_info(f'send to blastee: {packet}')
                self.net.send_packet("middlebox-eth1", packet)
            else:
                log_info('\033[1;31mdrop!\033[0m') # ]]
        elif fromIface == "middlebox-eth1":
```

```
            log_info(f"Received from blastee: {packet}")
            heth_idx = packet.get_header_index(Ethernet)
            packet[heth_idx].src = blaster_intf_mac
            packet[heth_idx].dst = blaster_mac
            log_info(f'send to blaster: {packet}')
            self.net.send_packet("middlebox-eth0", packet)
```

3.3 Blastee

Blastee sends an ack back when receiving a packet from blaster. According to packet structure, It should do the following:

1. Create Ethernet, IPv4, UDP headers and fill in src and dst field;
2. copy [0:4] byte from RawPacketContent of the received packet into ack's RawPacketContent (since it is sequence number);
3. extract [4:6] byte as payload length;
4. if it is larger than 8, copy the first 8 bytes of payload into ack's payload;
5. if not, copy payload and pad to 8 bytes.

```
def handle_packet(self, recv: switchyard.llnetbase.
    ReceivedPacket):
        _, fromIface, packet = recv
        log_info(f"I got a packet: {packet}")
        reply = (
            Ethernet(src=blastee_mac, dst=blastee_intf_mac,
                ethertype=EtherType.IP)
            + IPv4(
                protocol=IPProtocol.UDP,
                src=self.net.interfaces()[0].ipaddr,
                dst=self.blasterIp,
            )
            + UDP()
        )
        payload_len = int.from_bytes(packet[3].to_bytes()[4:6],
            "big")
        if payload_len >= 8:
            reply += packet[3].to_bytes()[:4] + packet[3].
                to_bytes()[6:6+8]
        else:
            reply += packet[3].to_bytes()[:4] + packet[3].
                to_bytes()[6:] + (0).to_bytes(8 - payload_len, "
                big")
        assert(len(reply[3]) == 12)
```

```
19          log_info(f"send: {reply}")
20          self.net.send_packet(fromIface, reply)
21          seqnum = int.from_bytes(packet[3].to_bytes()[:4], "big")
22          if seqnum == self.num - 1:
23              self.net.shutdown()
```

3.4 Blaster

The core part of blaster is how to implement the sender window. The most natural implementation is using an array storing all packets to be sent and two pointers indicating left and right edge of the window. However, that requires a huge amount of storage which is unacceptable on actual host. So I used a deque to represent the sender window – move left edge equivalent to popleft and move right edge equivalent to appendright. I created a class called Buffer to capsulize data and operation related to deque. Here is the detailed implementation.

3.4.1 create new packet

New packets should be created before inserting into deque. We must maintain sequence number in the process.

```
1  def new_packet(self):
2          if self.seqnum >= self.num:
3                  return None
4          pkt = Ethernet() + IPv4() + UDP()
5          pkt[1].protocol = IPProtocol.UDP
6          pkt[0].src = blaster_mac
7          pkt[0].dst = blaster_intf_mac
8          pkt[1].src = self.net.interfaces()[0].ipaddr
9          pkt[1].dst = self.blasteeIp
10         pkt += RawPacketContents(self.seqnum.to_bytes(4, 'big')
               + self.pkt_len.to_bytes(2, 'big') + (0).to_bytes(self
               .pkt_len, 'big'))
11         self.seqnum += 1
12         return pkt
```

3.4.2 handle ack packet

When received an ack, extract sequence number and match that in the sender window. Then move left edge and right edge accordingly.

```
1  def ack(self, seqnum):
2      for entry in self.packets:
3          if entry[0][3].to_bytes()[:4] == seqnum:
4              entry[1] = True
```

```
            break
    while len(self.packets) and self.packets[0][1]:
        self.packets.popleft()
        packet = self.new_packet()
        if packet is not None:
            self.packets.append([packet, False])
            log_info(f'ack send {self.seqnum - 1}')
            self.net.send_packet(self.net.interfaces()[0],
                packet)
        self.timestamp = time()
    if len(self.packets) == 0:
        self.print_info()
        self.net.shutdown()
```

3.4.3 check timeout

If blaster doesn't receive an ack in recvTimeout, it should check timeout of the sender window. If timeout it will retrainsmit.

```
def check_timeout(self):
        if time() - self.timestamp >= self.to:
                self.timeouts += 1
                for entry in self.packets:
                        if not entry[1]:
                                seqnum = int.from_bytes(entry
                                    [0][3].to_bytes()[:4], 'big')
                                log_info(f'timeout resend: seq =
                                     {seqnum}')
                                self.net.send_packet(self.net.
                                    interfaces()[0], entry[0])
                                self.retransmit += 1
                self.timestamp = time()
```

4 Test & Result

I tested my code in mininet with dropRate = 0.4, num = 10 and the result is below (the capture file is saved in report/).

```
root@ASUS-VivoBook:/home/xsy/Workspace/assignments/network/lab-6-xingshangyu# swyard blaste
r.py -g 'blasteeIp=192.168.200.1 num=10 length=100 senderWindow=3 timeout=1000 recvTimeout=
200'
17:17:13 2021/05/20     INFO Saving iptables state and installing switchyard rules
17:17:13 2021/05/20     INFO Using network devices: blaster-eth0
17:17:13 2021/05/20     INFO init send: 0
17:17:13 2021/05/20     INFO init send: 1
17:17:13 2021/05/20     INFO init send: 2
17:17:13 2021/05/20     INFO Didn't receive anything
17:17:13 2021/05/20     INFO I got a packet: seq = 0
17:17:13 2021/05/20     INFO ack send 3
17:17:13 2021/05/20     INFO I got a packet: seq = 1
17:17:13 2021/05/20     INFO ack send 4
17:17:13 2021/05/20     INFO I got a packet: seq = 2
17:17:13 2021/05/20     INFO ack send 5
17:17:13 2021/05/20     INFO Didn't receive anything
17:17:14 2021/05/20     INFO Didn't receive anything
17:17:14 2021/05/20     INFO I got a packet: seq = 4
17:17:14 2021/05/20     INFO I got a packet: seq = 5
17:17:14 2021/05/20     INFO Didn't receive anything
17:17:14 2021/05/20     INFO Didn't receive anything
17:17:14 2021/05/20     INFO Didn't receive anything
17:17:14 2021/05/20     INFO timeout resend: seq = 3
17:17:14 2021/05/20     INFO Didn't receive anything
17:17:15 2021/05/20     INFO Didn't receive anything
17:17:15 2021/05/20     INFO I got a packet: seq = 3
17:17:15 2021/05/20     INFO ack send 6
17:17:15 2021/05/20     INFO ack send 7
17:17:15 2021/05/20     INFO ack send 8
17:17:15 2021/05/20     INFO Didn't receive anything
17:17:15 2021/05/20     INFO Didn't receive anything
17:17:15 2021/05/20     INFO I got a packet: seq = 7
17:17:15 2021/05/20     INFO I got a packet: seq = 8
17:17:15 2021/05/20     INFO Didn't receive anything
17:17:15 2021/05/20     INFO Didn't receive anything
17:17:15 2021/05/20     INFO I got a packet: seq = 7
17:17:15 2021/05/20     INFO I got a packet: seq = 8
17:17:15 2021/05/20     INFO Didn't receive anything
17:17:16 2021/05/20     INFO Didn't receive anything
17:17:16 2021/05/20     INFO Didn't receive anything
17:17:16 2021/05/20     INFO timeout resend: seq = 6
17:17:16 2021/05/20     INFO Didn't receive anything
17:17:16 2021/05/20     INFO Didn't receive anything
17:17:16 2021/05/20     INFO I got a packet: seq = 6
17:17:16 2021/05/20     INFO ack send 9
17:17:16 2021/05/20     INFO Didn't receive anything
17:17:17 2021/05/20     INFO Didn't receive anything
17:17:17 2021/05/20     INFO I got a packet: seq = 9
17:17:17 2021/05/20     INFO Total TX time: 3.728550672531128
17:17:17 2021/05/20     INFO Number of reTX: 2
17:17:17 2021/05/20     INFO Number of coarse TOs: 2
17:17:17 2021/05/20     INFO Throughput (Bps): 321.8408720687654
17:17:17 2021/05/20     INFO Goodput (Bps): 268.2007267239712
```

Figure 2: blaster's log

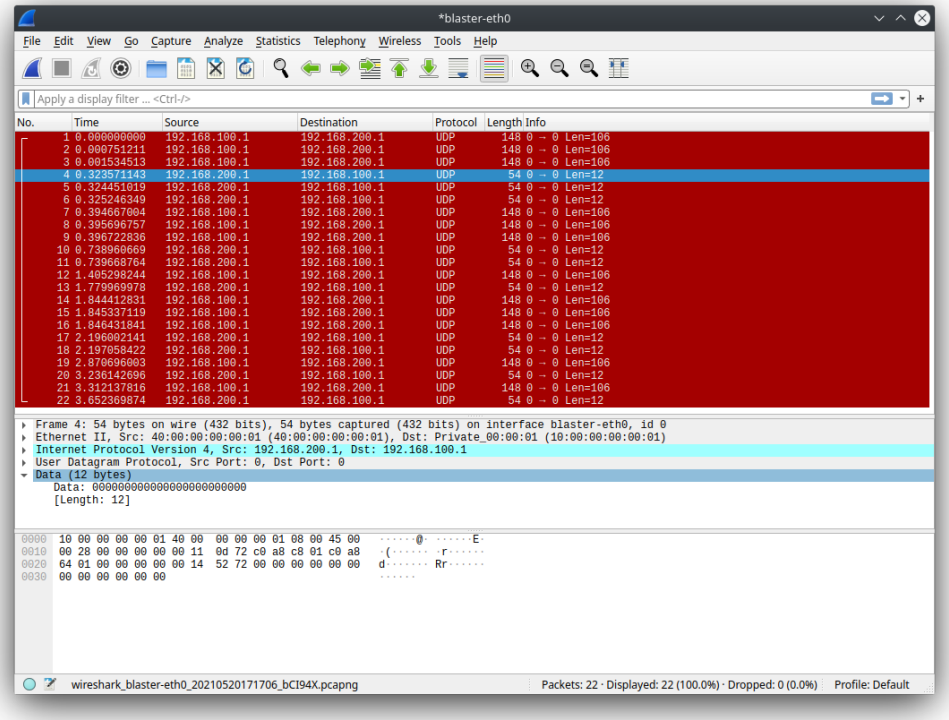

Figure 3: wireshark capture result

We can learn from the log and capture result that:

- in the 13s, blaster received 3 acks, so it moved the sender window and sent 3 new packets;
- in the 14s, blaster received acks for packet 4 and 5, but packet 3 timeout (dropped in middlebox), so it resent packet 3;
- the program ended when the last packet (packet 9) was acked.

The behavior of the network was the same as expected, so I have confidence that my program is correct.
Lastly, I present here the test result of the default parameters:

```
18:00:44 2021/05/20     INFO I got a packet: seq = 91
18:00:44 2021/05/20     INFO ack send 96
18:00:44 2021/05/20     INFO I got a packet: seq = 93
18:00:44 2021/05/20     INFO I got a packet: seq = 94
18:00:44 2021/05/20     INFO Didn't receive anything
18:00:44 2021/05/20     INFO Didn't receive anything
18:00:44 2021/05/20     INFO Didn't receive anything
18:00:44 2021/05/20     INFO timeout resend: seq = 92
18:00:44 2021/05/20     INFO timeout resend: seq = 95
18:00:44 2021/05/20     INFO timeout resend: seq = 96
18:00:44 2021/05/20     INFO I got a packet: seq = 96
18:00:44 2021/05/20     INFO Didn't receive anything
18:00:44 2021/05/20     INFO Didn't receive anything
18:00:45 2021/05/20     INFO Didn't receive anything
18:00:45 2021/05/20     INFO timeout resend: seq = 92
18:00:45 2021/05/20     INFO timeout resend: seq = 95
18:00:45 2021/05/20     INFO I got a packet: seq = 92
18:00:45 2021/05/20     INFO ack send 97
18:00:45 2021/05/20     INFO ack send 98
18:00:45 2021/05/20     INFO ack send 99
18:00:45 2021/05/20     INFO I got a packet: seq = 95
18:00:45 2021/05/20     INFO I got a packet: seq = 96
18:00:45 2021/05/20     INFO Didn't receive anything
18:00:45 2021/05/20     INFO Didn't receive anything
18:00:45 2021/05/20     INFO Didn't receive anything
18:00:45 2021/05/20     INFO timeout resend: seq = 97
18:00:45 2021/05/20     INFO timeout resend: seq = 98
18:00:45 2021/05/20     INFO timeout resend: seq = 99
18:00:45 2021/05/20     INFO I got a packet: seq = 92
18:00:45 2021/05/20     INFO I got a packet: seq = 95
18:00:45 2021/05/20     INFO I got a packet: seq = 97
18:00:45 2021/05/20     INFO I got a packet: seq = 98
18:00:45 2021/05/20     INFO I got a packet: seq = 99
18:00:45 2021/05/20     INFO Total TX time: 9.475518941879272
18:00:45 2021/05/20     INFO Number of reTX: 85
18:00:45 2021/05/20     INFO Number of coarse TOs: 24
18:00:45 2021/05/20     INFO Throughput (Bps): 1952.3996641740562
18:00:45 2021/05/20     INFO Goodput (Bps): 1055.3511698238142
```

Figure 4: test result with default parameters

5 Summary

- Knowing how to effectively use debugging tools such as pdb will greatly enhance working efficiency;
- English reading and writing skills are important.