南開大學
Nankai University

南　　开　　大　　学

计　算　机　学　院

计算机网络实验报告

实验 3-1

学号：1911437

姓名：刘浩通

年级：2019 级

专业：计算机科学与技术

2021 年 12 月 3 日

目录

一、 实验目的 **1**

二、 协议设计 **1**

（一） UDP 报文设计 1
（二） 发送端设计 2
（三） 接收端设计 3
（四） 两次握手设计 4
（五） 两次挥手设计 5
（六） 差错检验 5
（七） 确认重传 5
（八） 超时重传 6

三、 具体代码实现 **7**

（一） 计算校验和 7
（二） 两次握手建立连接 7
（三） 两次挥手 10
（四） 文件的传输 12
（五） 重传超时时间计算 15

四、 实验结果 **16**

一、 实验目的

利用数据报套接字在用户空间实现面向连接的可靠数据传输，功能包括：

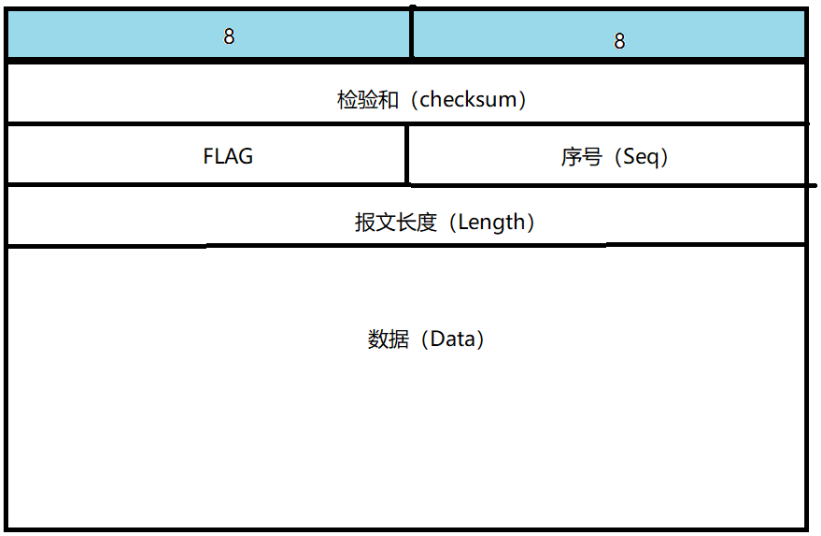
1. 建立连接
2. 差错检验
3. 确认重传
4. 流量控制采用停等机制

完成给定测试文件的传输。

二、 协议设计

(一) UDP 报文设计

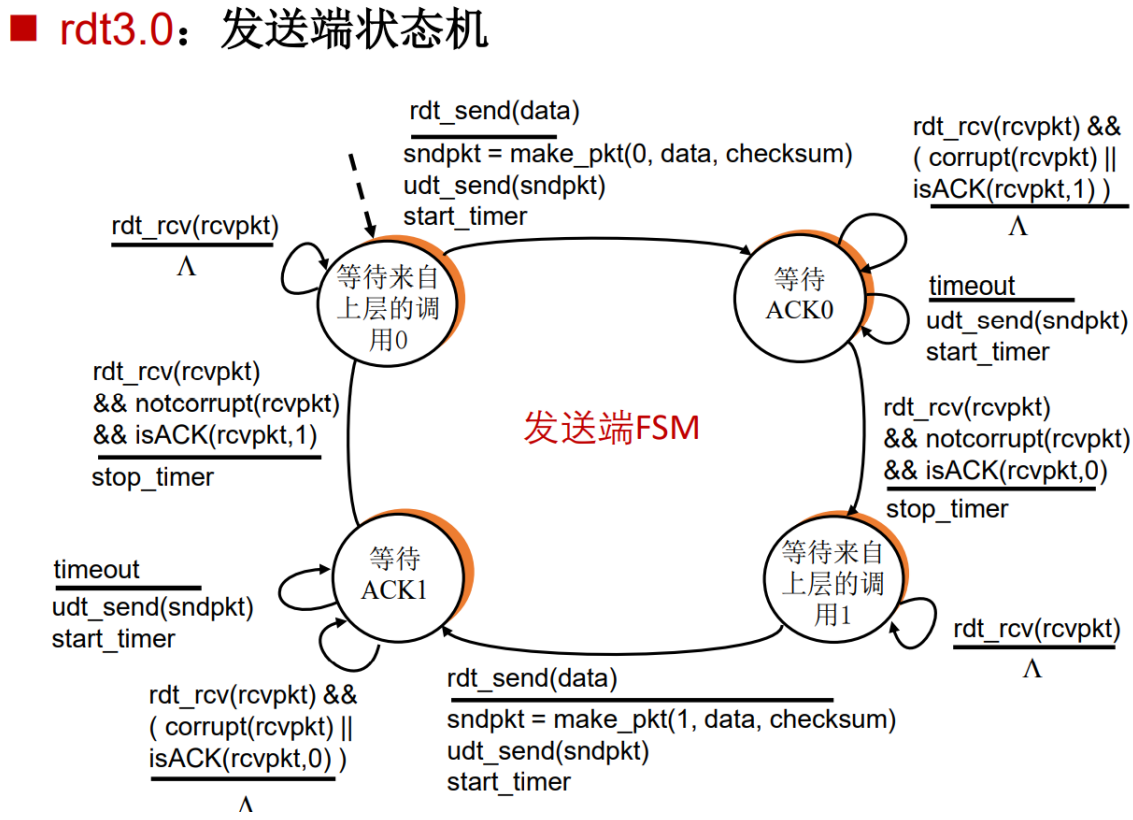
我设计的报文格式如下：



1. 校验和 (checksum)：16 位，用于判断报文是否出错
2. 标志位 (Flag):8 位，ACK 标志字 (接收消息用于回复确认)、SYN 标志字段 (握手阶段使用)、SF 标志字段 (信息，标明未结束)、EF 标志字段 (传输的最后一条信息)、FIN 标志字段 (挥手阶段使用)。

3. 序号 (seq)：8 位，因为是使用的停等协议，所以在此次实验只用了 0 和 1.
4. 报文长度 (Length):16 位，整个报文的长度，数据字段长度就是减去前面定长的头部信息。
5. data: 应用层的数据

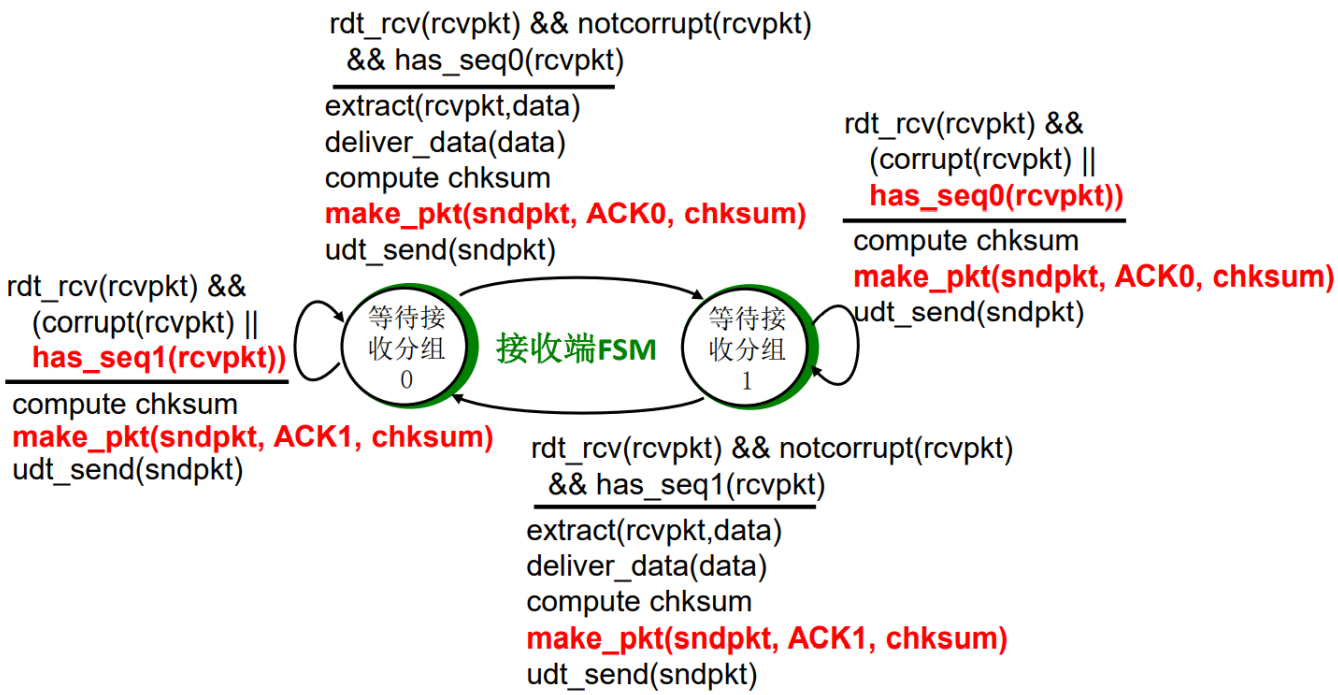
(二) 发送端设计



发送端的状态机上图所示。发送端先发送序号为 0 的报文段，然后开启定时器，等待接收端发送 ack+seq=0：

- 接收的报文段损坏或是 ack+(seq=1)，则进行等待
- 接收的报文段未损坏并且是 ack+(seq=0)，停止定时器，进入下一发送周期，等待发送序号为 1 的报文段
- 定时器超时，重传序号为 0 的报文段，重启定时器

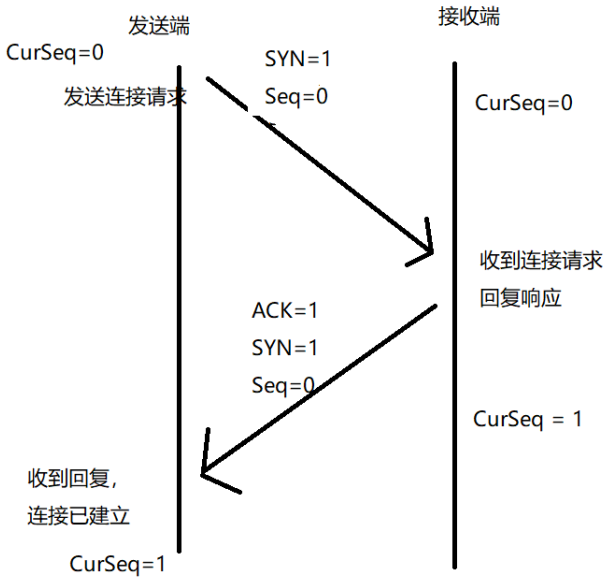
(三) 接收端设计



接收端的状态机和发送端类似。接收端先等待序号为 0 的报文段, 收到报文段后:

- 报文段损坏，则接收端不反应，继续等待正确报文
- 若发来的报文序号是 1，则发送 ack1，继续等待正确报文
- 报文段未损坏并且是序号 0，发送 ack0，进入下一接收周期

（四） 两次握手设计

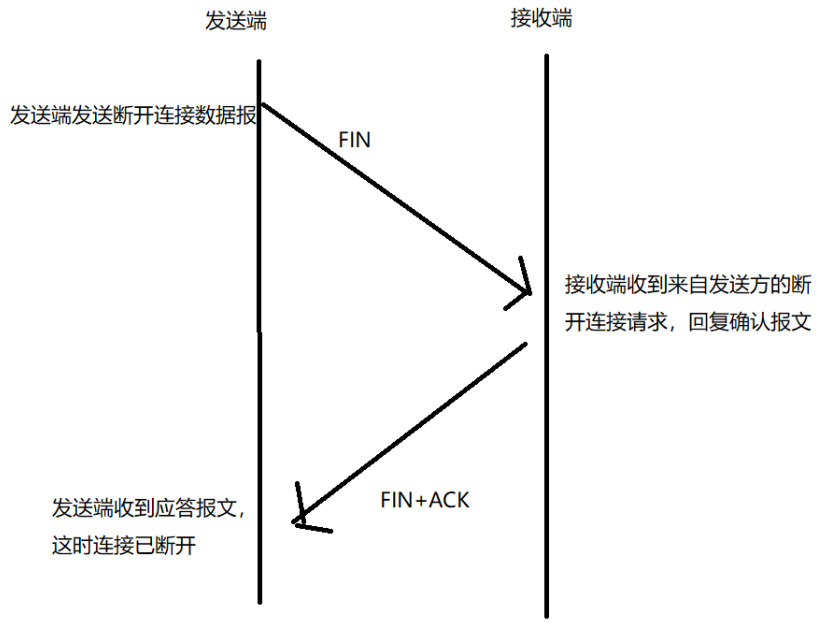


发送端会向接收端发送一个带有 SYN 并且序号为 0 的 UDP 数据报。

接收方收到此数据报后，会回复给发送端 SYN+ACK 并且序号为 0 的数据报。并且发送方进入连接建立状态，curseq 置反，进入等待序号为 1 的数据报状态。

发送方收到接收方发来的回复数据报，这时两次握手连接已经建立，发送方进入序号为 1 的状态。

（五） 两次挥手设计



准备断开连接，发送端向接收端发送带有 FIN 标志的报文。

接收方收到来自发送方的报文后，检查了序号、检验码无误后，回复 FIN+ACK 的回复报文。

发送端收到来自接收端的报文后，连接成功断开。

（六） 差错检验

1. 发送端：
   对要发送的 UDP 数据报，校验和域段清 0，将数据报用 0 补齐为 16 位整数倍，将数据报看成 16 位整数序列，进行 16 位二进制反码求和运算计算校验和，并将校验和结果取反写入校验和域段。

2. 接收端：
   对接收到的 UDP 数据报，将数据报用 0 补齐为 16 位整数倍，将数据报看成 16 位整数序列，进行 16 位二进制反码求和运算计算校验和，并将校验和结果取反，如果取反结果为 0，则没有检测到错误，否则，数据报存在差错。

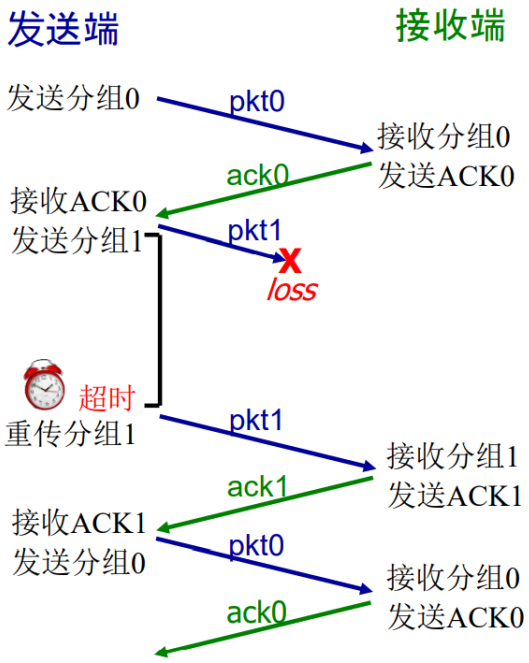
（七） 确认重传

若当前都在序号为 0 的状态：

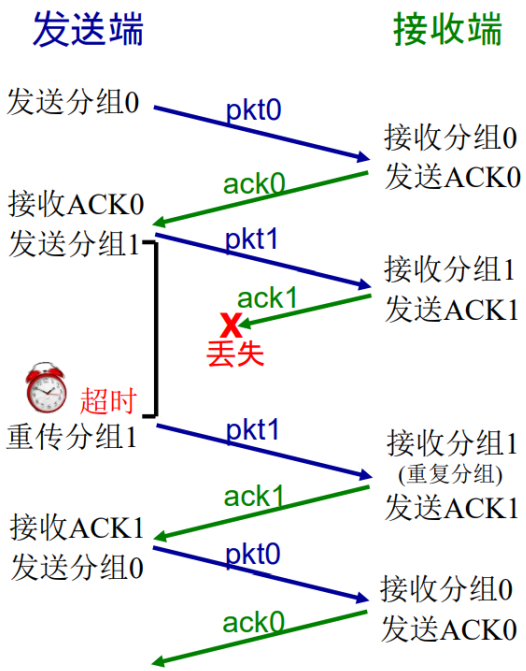
接收端收到包之后，计算校验和，如果数据包检测无差错，并且发生来的序号为 0，将序号置为 0，发送给发送端，确认收到；如果发送来的序号为 1 错，将序号置 1，发送给发送；

发送端收到接收端发来的 ACK 之后，如果序号为 0, 表示数据包发送成功，发送端继续发送下一个数据包；如果序号为 1，表示数据包有差错，等待超时重传。

（八） 超时重传



在分组丢失的情况下，发送端时间超时还没有收到来自发送端的回复则会进行重传。



在 ACK 丢失的情况下，由于发送端没有在超时时间内收到接收端确认报文，但是接收端状态这时已经转移，等待的是相反序号的报文，所以发送端超时重传后，接收端会回复这个校验正确的报文，并且回复序号是相反的，接收端并不会转移状态。

然后为了解决超时时间设置不合理的问题，这里超时时间采用了 TCP 超时重传时间的 Jacobson/Karels 算法来估算超时时间，使超时时间考虑网络时延的变化情况，也避免了超时重传过程中超时太久的问题。

三、 具体代码实现

(一) 计算校验和

```
USHORT checksum(char* buf, int len) {
        unsigned char* ubuf = (unsigned char*)buf;
        register ULONG sum = 0;
        while (len) {
                USHORT temp = USHORT(*ubuf << 8) + USHORT(*(ubuf + 1));
                sum += temp;
                if (sum & 0xFFFF0000)
                {
                        sum &= 0xFFFF;
                        sum++;
                }
                ubuf += 2;
                len -= 2;
        }
        return ~(sum & 0xFFFF);
}
```

会传入一个 char 类型数组和数组长度 len，len 是偶数。因为是 16 进制进行校验计算，而 char 是 8 位，所以是将两个 char 组成 16 位进行校验和计算。在调用此函数时，会把数组补全成偶数长度，补全用 0。

(二) 两次握手建立连接

发送端:

```
void shake_hand() {
        while (1) {
                char tmp[6];
                tmp[2] = SYN;
                tmp[3] = curSeq;
                tmp[4] = 6 >> 8;
                tmp[5] = 6 & 0xff;
                USHORT ck = checksum(tmp + 2, 4);
                tmp[0] = ck >> 8;
                tmp[1] = ck & 0xff;
                //发送请求连接
```

```
                sendto(client, tmp, 6, 0, (sockaddr*)&serverAddr, sizeof(
                    serverAddr));

                int begintime = clock();//开始计时
                char recv[6];
                int lentmp = sizeof(clientAddr);
                int fail_send = 0;
                while (recvfrom(client, recv, 6, 0, (sockaddr*)&serverAddr, &
                    lentmp) == SOCKET_ERROR)
                    if (clock() - begintime > RTO) {//超时了，即将重传
                        fail_send = 1;
                        break;
                    }
                double sampleRTT = clock() - begintime;
                if (fail_send == 0 && checksum(recv, 6) == 0 && recv[2] == (
                    SYN + ACK) && recv[3] == curSeq) {
                //没超时，状态转移，连接建立
                    addSampleRTT(sampleRTT);//计算新的超时时间
                    curSeq = curSeq == 0 ? 1 : 0;
                    return;
                }
        }
}
```

接收端:

```
void wait_shake_hand() {
        while (1) {
                char recv[6], send[6];
                int connect = 0;
                int lentmp = sizeof(clientAddr);
                while (recvfrom(server, recv, 6, 0, (sockaddr*)&clientAddr, &
                    lentmp) == SOCKET_ERROR);

                if (checksum(recv, 6) == 0 && recv[2] == SYN && recv[3] !=
                    curSeq) {
        //这个是ACK丢失，导致发送端超时重传的的报文，就是回复一个ACK，但状态
            不转移
                    cout << "ACK丢失" << endl;
                    send[2] = (SYN + ACK);
                    send[3] = recv[3];
                    send[4] = 6 >> 8;
                    send[5] = 6 & 0xff;

                    USHORT ck = checksum(send + 2, 4);
                    send[0] = ck >> 8;
                    send[1] = ck & 0xff;
                    sendto(server, send, 6, 0, (sockaddr*)&clientAddr,
                        sizeof(clientAddr));
```

```
20
21                  cout << "收到:";
22                  output(recv[2]);
23                  cout << " Length:" << (USHORT)((unsigned char)recv[4]
                        << 8) + (USHORT)(unsigned char)recv[5];
24                  cout << " Checksum:" << (USHORT)((unsigned char)recv
                        [0] << 8) + (USHORT)(unsigned char)recv[1];
25                  cout << " Seq" << (int)recv[3];
26                  cout << endl;
27                  cout << "发送:";
28                  output(send[2]);
29                  cout << " Length:" << (USHORT)((unsigned char)send[4]
                        << 8) + (USHORT)(unsigned char)send[5];
30                  cout << " Checksum:" << ck;
31                  cout << " Seq:" << (int)send[3];
32                  cout << endl;
33                  continue;
34              }
35
36
37              if (checksum(recv, 6) != 0 || recv[2] != SYN || recv[3] !=
                    curSeq)
38              continue;
39
40              //收到了SYN,若校验成功, 回复
41              send[2] = (SYN + ACK);
42              send[3] = curSeq;
43              send[4] = 6 >> 8;
44              send[5] = 6 & 0xff;
45
46              USHORT ck = checksum(send + 2, 4);
47              send[0] = ck >> 8;
48              send[1] = ck & 0xff;
49              sendto(server, send, 6, 0, (sockaddr*)&clientAddr, sizeof(
                    clientAddr));
50
51              cout << "收到:";
52              output(recv[2]);
53              cout << " Length:" << (USHORT)((unsigned char)recv[4] << 8) +
                     (USHORT)(unsigned char)recv[5];
54              cout << " Checksum:" << (USHORT)((unsigned char)recv[0] << 8)
                     + (USHORT)(unsigned char)recv[1];
55              cout << " Seq" << (int)recv[3];
56              cout << endl;
57              cout << "发送:";
58              output(send[2]);
59              cout << " Length:" << (USHORT)((unsigned char)send[4] << 8) +
                     (USHORT)(unsigned char)send[5];
```

```
                cout << " Checksum:" << ck;
                cout << " Seq:" << (int)send[3];
                cout << endl;

                curSeq = curSeq == 0 ? 1 : 0;
                break;
        }
}
```

(三) 两次挥手

发送端:

```
void wave_hand() {
        int tot_fail = 0;
        while (1) {
//发送端，会发送带有FIN的数据报给接收端，代表要断开连接
                char tmp[6];
                tmp[2] = FIN;
                tmp[3] = curSeq;
                tmp[4] = 6 >> 8;
                tmp[5] = 6 & 0xff;
                USHORT ck = checksum(tmp + 2, 4);
                tmp[0] = ck >> 8;
                tmp[1] = ck & 0xff;
                sendto(client, tmp, 6, 0, (sockaddr*)&serverAddr, sizeof(
                    serverAddr));
//开始计时，用于超时判断
                double begintime = clock();
                char recv[6];
                int lentmp = sizeof(serverAddr);
                int fail_send = 0;
//这里会记超时次数，因为接收端发送响应报文时，
//就已经发送端就已经断开连接了，所以如果发生ACK丢失
//发送端超时重传，不会收到响应，所以超时次数过多也关闭了
                while (recvfrom(client, recv, 6, 0, (sockaddr*)&serverAddr, &
                    lentmp) == SOCKET_ERROR) {
                        if (clock() - begintime > RTO) {
                                fail_send = 1;
                                tot_fail++;
                                break;
                        }
                }
                double sampleRTT = clock() - begintime;
//接收到响应报文，连接断开，退出
                if (fail_send == 0 && checksum(recv, 6) == 0 && recv[2] == (
                    FIN + ACK) && recv[3] == curSeq) {
                        addSampleRTT(sampleRTT);
```

```
                        curSeq = curSeq == 0 ? 1 : 0;
                        break;
                }
                else {
                        if (tot_fail == 10) {
                                printf("断开失败, 释放资源");
                                break;
                        }
                        continue;
                }
        }
}
```

接收端:

```
void wait_wave_hand() {
        while (1) {
                char recv[6], send[6];
                int lentmp = sizeof(clientAddr);
                while (recvfrom(server, recv, 6, 0, (sockaddr*)&clientAddr, &
                    lentmp) == SOCKET_ERROR);
                if (checksum(recv, 6) != 0 || recv[2] != (char)FIN || recv[3]
                     != curSeq)
                continue;
//接收端收到带有FIN的发送端报文
//会回复带有FIN+ACK的响应报文, 之后接收端断开连接退出了
                send[2] = (ACK + FIN);
                send[3] = curSeq;
                send[4] = 6 >> 8;
                send[5] = 6 & 0xff;
                USHORT ck = checksum(send + 2, 4);
                send[0] = ck >> 8;
                send[1] = ck & 0xff;
                sendto(server, send, 6, 0, (sockaddr*)&clientAddr, sizeof(
                    clientAddr));
                cout << "收到:";
                output(recv[2]);
                cout << " Length:" << (USHORT)((unsigned char)recv[4] << 8) +
                     (USHORT)(unsigned char)recv[5];
                cout << " Checksum:" << (USHORT)((unsigned char)recv[0] << 8)
                     + (USHORT)(unsigned char)recv[1];
                cout << " Seq" << (int)recv[3];
                cout << endl;
                cout << "发送:";
                output(send[2]);
                cout << " Length:" << (USHORT)((unsigned char)send[4] << 8) +
                     (USHORT)(unsigned char)send[5];
                cout << " Checksum:" << ck;
                cout << " Seq:" << curSeq;
```

```
                cout << endl;
                curSeq = curSeq == 0 ? 1 : 0;
                break;
        }
}
```

（四） 文件的传输

发送端:

发送端读取文件后，首先会将文件拆成一个个的小报文进行发送，因为报文有长度限制。

```
void send_message(char* message, int lent) {
        int package_num = lent / Mlenx + (lent % Mlenx != 0);//看要发送几个包
        for (int i = 0; i < package_num; i++) {
                bool last = (i == (package_num - 1));//判断是否是最后一个包
                int length = last ? lent - (package_num - 1) * Mlenx : Mlenx;
                //因为前面发送的包都是最大长度，最后一个则不是所以要另外计算
                send_package(message + i * Mlenx, length, last);

                if (i % 10 == 0)
                printf("此文件已经发送%.2f%%\n", (float)i / package_num *
                    100);
        }
}
```

然后就是发送端的投递过程:

```
bool send_package(char* message, int lent, bool last) {

        char* tmp;
        int tmp_len;

        int length = lent % 2 == 0 ? lent : lent + 1;
        //因为使用char数组来存储的数据，而在计算校验和的时候使用16位计算，所
            以要进行长度补全
        tmp = new char[length + 6];
        tmp[length + 5] = 0;
//若是最后一个传输数据的包，标志则是EF，不是则是SF
        if (last) {
                tmp[2] = EF;
        }
        else {
                tmp[2] = SF;
        }
        tmp[3] = curSeq;

        tmp[4] = (lent + 6) >> 8;
        tmp[5] = (lent + 6) & 0xff;
        //cout << tmp[3] << endl;
```

```
        //将文件数据导入到包中
        for (int i = 6; i < lent + 6; i++)
        tmp[i] = message[i - 6];
        USHORT ck = checksum(tmp + 2, length + 4);//计算校验和
        tmp[0] = ck >> 8;
        tmp[1] = ck & 0xff;

        tmp_len = lent + 6;
        int send_to_cnt = 0;
        while (1) {
                send_to_cnt++;//有重传限制
                //发送数据报
                sendto(client, tmp, tmp_len, 0, (sockaddr*)&serverAddr,
                    sizeof(serverAddr));
                int begintime = clock();
                char recv[6];
                int lentmp = sizeof(serverAddr);
                int fail_send = 0;
                while (recvfrom(client, recv, 6, 0, (sockaddr*)&serverAddr, &
                    lentmp) == SOCKET_ERROR)
                if (clock() - begintime > RTO) {
        //超时重传，如果超过了最大的重传次数，则进行挥手断开连接
                        fail_send = 1;
                        if (send_to_cnt == 10){
                                wave_hand();
                                return false;
                        }
                        break;
                }
                double sampleRTT = clock() - begintime;
                if (fail_send == 0 && checksum(recv, 6) == 0 && recv[2] ==
                    ACK && recv[3] == curSeq) {
                        addSampleRTT(sampleRTT);
                        curSeq = curSeq == 0 ? 1 : 0;
                        return true;
                }
        }
}
```

接收端:

```
void recv_message(char* message, int& len_recv) {
        char recv[Mlenx + 6];
        int lentmp = sizeof(clientAddr);
        len_recv = 0;
        while (1) {
                while (1) {
        //这一层循环是用来接收报文，并且判断正确的
        //如果收到接收方想要的报文，并且发生转移
```

```
9           //则会跳出这一层循环，并且将接收到的数据保存在一个临时数值中
10          //直至收到EF标志的报文，则退出最外层循环
11                      memset(recv, 0, sizeof(recv));
12                      while (recvfrom(server, recv, Mlenx + 6, 0, (sockaddr
                            *)&clientAddr, &lentmp) == SOCKET_ERROR);
13                      char send[6];
14                      if (checksum(recv, Mlenx + 6) == 0 && recv[3] ==
                            curSeq) {
15                      //收到正确的报文
16                              send[2] = ACK;
17                              send[3] = curSeq;
18                              send[4] = 6 >> 8;
19                              send[5] = 6 & 0xff;
20                              USHORT ck = checksum(send + 2, 4);
21
22                              send[0] = ck >> 8;
23                              send[1] = ck & 0xff;
24                              sendto(server, send, 6, 0, (sockaddr*)&
                                    clientAddr, sizeof(clientAddr));
25
26                              cout << "收到:";
27                              output(recv[2]);
28                              cout << " Length:" << (USHORT)((unsigned char
                                    )recv[4] << 8) + (USHORT)(unsigned char)
                                    recv[5];
29                              cout << " Checksum:" << (USHORT)((unsigned
                                    char)recv[0] << 8) + (USHORT)(unsigned
                                    char)recv[1];
30                              cout << " Seq:" << (int)recv[3];
31                              cout << endl;
32                              cout << "发送:";
33                              output(send[2]);
34                              cout << " Length:" << 6;
35                              cout << " Checksum:" << ck;
36                              cout << " Seq:" << curSeq;
37                              cout << endl;
38                              curSeq = curSeq == 0 ? 1 : 0;
39                              break;
40                      }
41                      else if (checksum(recv, Mlenx + 6) == 0 && recv[3] !=
                            curSeq) {
42                              //这个是接收方ACK丢失，发送方重传的情况
43                              send[2] = ACK;
44                              send[3] = recv[3];//返回的序号是与当前curSeq
                                    相反的序号
45                              send[4] = 6 >> 8;
46                              send[5] = 6 & 0xff;
47                              USHORT ck = checksum(send + 2, 4);
```

```
48
49                                  send[0] = ck >> 8;
50                                  send[1] = ck & 0xff;
51                                  sendto(server, send, 6, 0, (sockaddr*)&
                                        clientAddr, sizeof(clientAddr));
52
53                                  cout << "收到:";
54                                  output(recv[2]);
55                                  cout << " Length:" << (USHORT)((unsigned char
                                        )recv[4] << 8) + (USHORT)(unsigned char)
                                        recv[5];
56                                  cout << " Checksum:" << (USHORT)((unsigned
                                        char)recv[0] << 8) + (USHORT)(unsigned
                                        char)recv[1];
57                                  cout << " Seq:" << (int)recv[3];
58                                  cout << endl;
59                                  cout << "发送:";
60                                  output(send[2]);
61                                  cout << " Length:" << 6;
62                                  cout << " Checksum:" << ck;
63                                  cout << " Seq:" << curSeq;
64                                  cout << endl;
65                          }
66                  }
67                  USHORT length = (USHORT)((unsigned char)recv[4] << 8) + (
                        USHORT)(unsigned char)recv[5];
68                  //将数据保存在这个数组中
69                  for (int i = 6; i < length; i++) {
70                          message[len_recv] = recv[i];
71                          len_recv++;
72                  }
73                  //若收到的数据包是EF, 也就是最后一个文件报文, 则退出
74                  if (EF == recv[2])
75                  break;
76          }
77  }
```

(五) 重传超时时间计算

根据算法写得:

EstimatedRTT=(1 -) EstimatedRTT+ SampleRTT

DevRTT = (1-) *DevRTT+ * | EstimatedRTT - SampleRTT|

(推荐值为 1/4)

RTO = μ * EstimatedRTT + * DevRTT

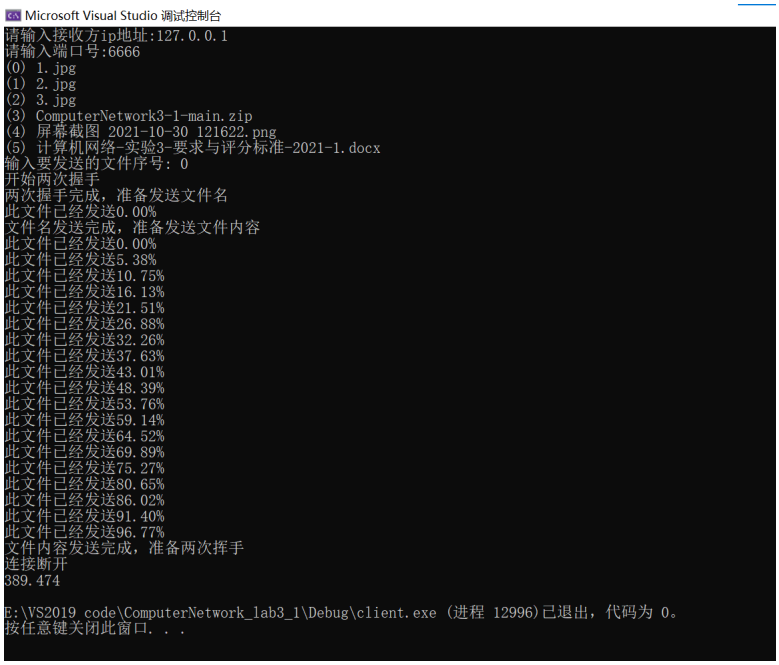
(μ 1, 4)

```
1  void addSampleRTT(double sampleRTT){
2          EstimatedRTT = 0.875 * EstimatedRTT + 0.125 * sampleRTT;
```

```
        DevRTT = 0.75 * DevRTT + 0.25 * abs(sampleRTT - EstimatedRTT);
        RTO = EstimatedRTT + 4 * DevRTT;
}
```

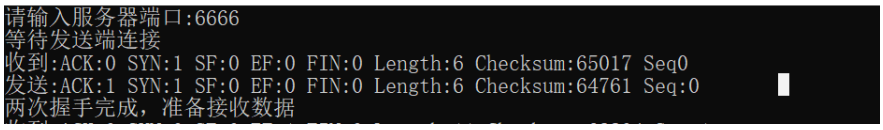
四、 实验结果

这里是用发送方发送 1.jpg 来进行实验验证

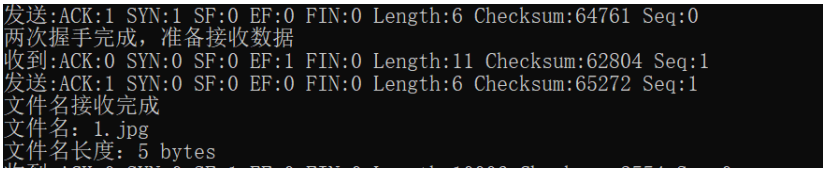


双方通信的报文信息日志由接收方打印：

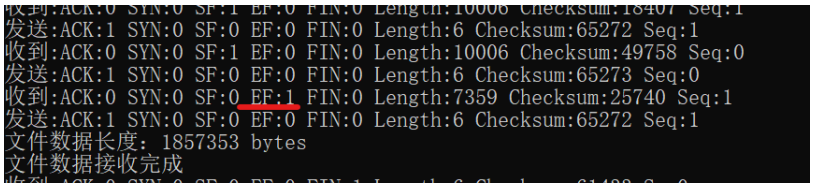
首先是两次握手阶段，发送方发送 SYN+seq=0 的请求握手 UDP 数据报，接受方回复 SYN+ACK+seq=0 的握手回复报文。由于是 rdt3.0 协议，状态转移等待 seq=1 的报文，发送方收到回复后状态也转移下次发送 seq=1 的报文。



然后是先发送文件名称给接收方：



接着发送文件的数据：(当文件数据发送完毕后，最后一个报文的标志位是 EF，代表文件数据已经传输完毕)



当文件传输完毕后，进入两次挥手状态，发送方发送含有 FIN 标志位报文，接收方回复 FIN+ACK 的报文。