



南开大学
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：

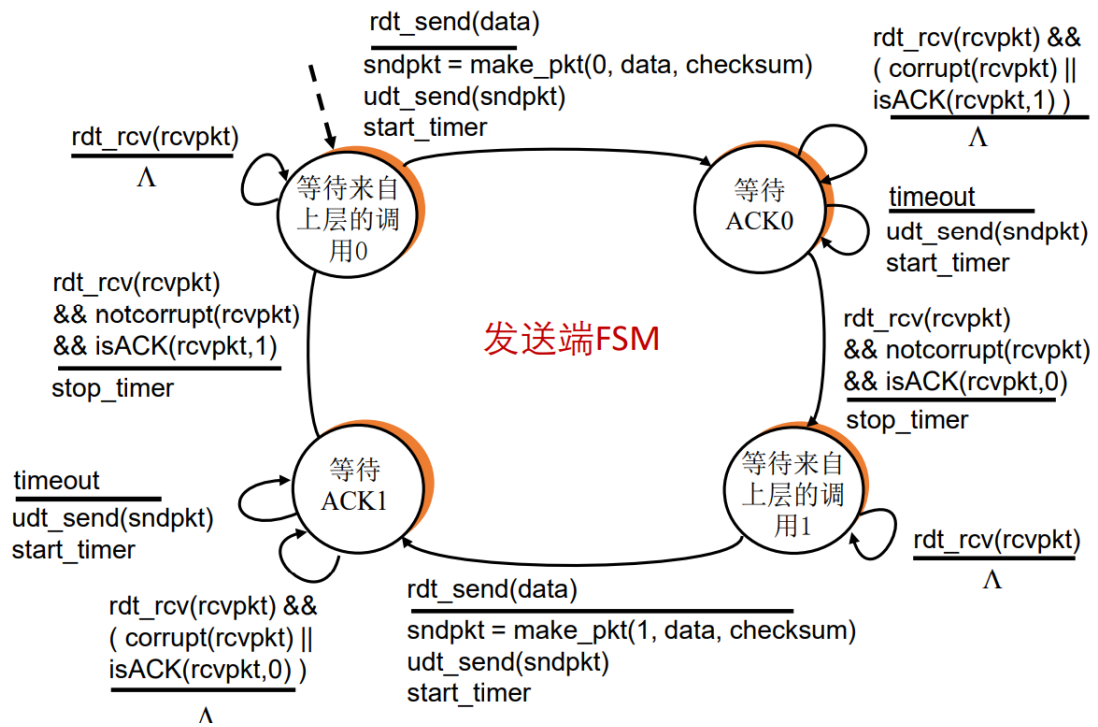
8	8
检验和 (checksum)	
FLAG	序号 (Seq)
报文长度 (Length)	
数据 (Data)	

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

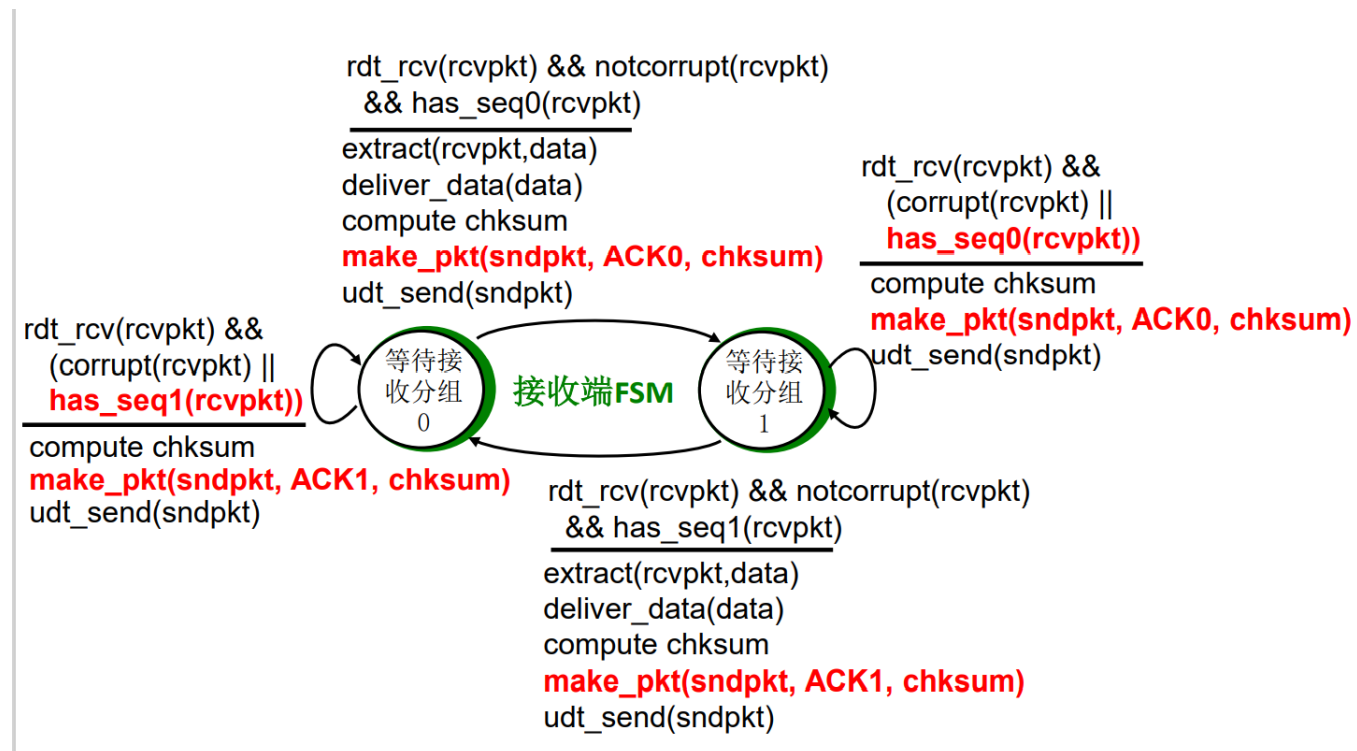
■ rdt3.0: 发送端状态机



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

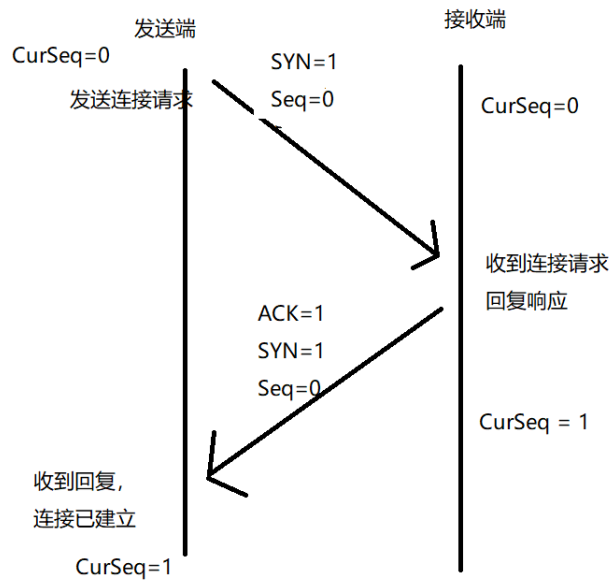
(三) 接收端设计



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

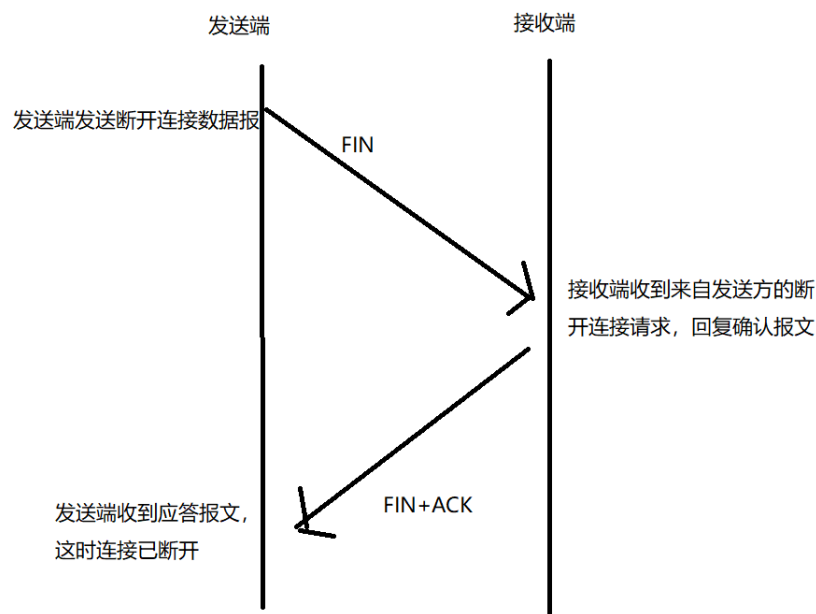


发送端会向接收端发送一个带有 SYN 并且序号为 0 的 UDP 数据报。

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。

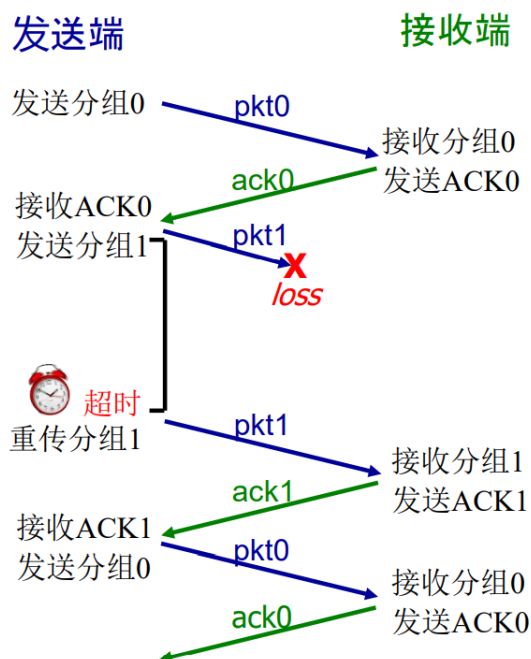
(七) 确认重传

若当前都在序号为 0 的状态：

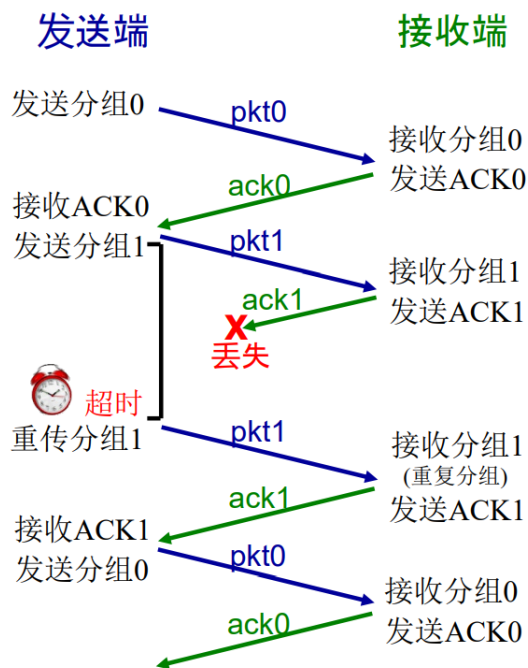
接收端收到包之后，计算校验和，如果数据包检测无差错，并且发生来的序号为 0，将序号置为 0，发送给发送端，确认收到；如果发送来的序号为 1 错，将序号置 1，发送给发送端；

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

```
1 USHORT checksum(char* buf, int len) {
2     unsigned char* ubuf = (unsigned char*)buf;
3     register ULONG sum = 0;
4     while (len) {
5         USHORT temp = USHORT(*ubuf << 8) + USHORT(*(ubuf + 1));
6         sum += temp;
7         if (sum & 0xFFFF0000)
8         {
9             sum &= 0xFFFF;
10            sum++;
11        }
12        ubuf += 2;
13        len -= 2;
14    }
15    return ~(sum & 0xFFFF);
16 }
```

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：

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

```

12         sendto(client, tmp, 6, 0, (sockaddr*)&serverAddr, sizeof(
13             serverAddr));
14
15         int begintime = clock(); // 开始计时
16         char recv[6];
17         int lentmp = sizeof(clientAddr);
18         int fail_send = 0;
19         while (recvfrom(client, recv, 6, 0, (sockaddr*)&serverAddr, &
20             lentmp) == SOCKET_ERROR)
21             if (clock() - begintime > RTO) { // 超时了, 即将重传
22                 fail_send = 1;
23                 break;
24             }
25         double sampleRTT = clock() - begintime;
26         if (fail_send == 0 && checksum(recv, 6) == 0 && recv[2] == (
27             SYN + ACK) && recv[3] == curSeq) {
28             // 没超时, 状态转移, 连接建立
29             addSampleRTT(sampleRTT); // 计算新的超时时间
30             curSeq = curSeq == 0 ? 1 : 0;
31             return;
32         }
33     }
34 }

```

接收端:

```

1 void wait_shake_hand() {
2     while (1) {
3         char recv[6], send[6];
4         int connect = 0;
5         int lentmp = sizeof(clientAddr);
6         while (recvfrom(server, recv, 6, 0, (sockaddr*)&clientAddr, &
7             lentmp) == SOCKET_ERROR);
8
9         if (checksum(recv, 6) == 0 && recv[2] == SYN && recv[3] !=
10             curSeq) {
11             // 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
12             // 不转移
13
14             cout << "ACK丢失" << endl;
15             send[2] = (SYN + ACK);
16             send[3] = recv[3];
17             send[4] = 6 >> 8;
18             send[5] = 6 & 0xff;
19
20             USHORT ck = checksum(send + 2, 4);
21             send[0] = ck >> 8;
22             send[1] = ck & 0xff;
23             sendto(server, send, 6, 0, (sockaddr*)&clientAddr,
24                 sizeof(clientAddr));
25         }
26     }
27 }

```

```

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

```

```
60         cout << " Checksum:" << ck;
61         cout << " Seq:" << (int)send[3];
62         cout << endl;
63
64         curSeq = curSeq == 0 ? 1 : 0;
65         break;
66     }
67 }
```

(三) 两次挥手

发送端:

```
1 void wave_hand() {
2     int tot_fail = 0;
3     while (1) {
4         //发送端, 会发送带有FIN的数据报给接收端, 代表要断开连接
5         char tmp[6];
6         tmp[2] = FIN;
7         tmp[3] = curSeq;
8         tmp[4] = 6 >> 8;
9         tmp[5] = 6 & 0xff;
10        USHORT ck = checksum(tmp + 2, 4);
11        tmp[0] = ck >> 8;
12        tmp[1] = ck & 0xff;
13        sendto(client, tmp, 6, 0, (sockaddr*)&serverAddr, sizeof(
            serverAddr));
14        //开始计时, 用于超时判断
15        double begintime = clock();
16        char recv[6];
17        int lentmp = sizeof(serverAddr);
18        int fail_send = 0;
19        //这里会记超时次数, 因为接收端发送响应报文时,
20        //就已经发送端就已经断开连接了, 所以如果发生ACK丢失
21        //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
22        while (recvfrom(client, recv, 6, 0, (sockaddr*)&serverAddr, &
            lentmp) == SOCKET_ERROR) {
23            if (clock() - begintime > RTO) {
24                fail_send = 1;
25                tot_fail++;
26                break;
27            }
28        }
29        double sampleRTT = clock() - begintime;
30        //接收到响应报文, 连接断开, 退出
31        if (fail_send == 0 && checksum(recv, 6) == 0 && recv[2] == (
            FIN + ACK) && recv[3] == curSeq) {
32            addSampleRTT(sampleRTT);
```

```

33         curSeq = curSeq == 0 ? 1 : 0;
34         break;
35     }
36     else {
37         if (tot_fail == 10) {
38             printf("断开失败, 释放资源");
39             break;
40         }
41         continue;
42     }
43 }
44 }

```

接收端:

```

1 void wait_wave_hand() {
2     while (1) {
3         char recv[6], send[6];
4         int lentmp = sizeof(clientAddr);
5         while (recvfrom(server, recv, 6, 0, (sockaddr*)&clientAddr, &
6             lentmp) == SOCKET_ERROR);
7         if (checksum(recv, 6) != 0 || recv[2] != (char)FIN || recv[3]
8             != curSeq)
9             continue;
10        //接收端收到带有FIN的发送端报文
11        //会回复带有FIN+ACK的响应报文, 之后接收端断开连接退出了
12        send[2] = (ACK + FIN);
13        send[3] = curSeq;
14        send[4] = 6 >> 8;
15        send[5] = 6 & 0xff;
16        USHORT ck = checksum(send + 2, 4);
17        send[0] = ck >> 8;
18        send[1] = ck & 0xff;
19        sendto(server, send, 6, 0, (sockaddr*)&clientAddr, sizeof(
20            clientAddr));
21        cout << "收到:";
22        output(recv[2]);
23        cout << " Length:" << (USHORT)((unsigned char)recv[4] << 8) +
24            (USHORT)(unsigned char)recv[5];
25        cout << " Checksum:" << (USHORT)((unsigned char)recv[0] << 8)
26            + (USHORT)(unsigned char)recv[1];
27        cout << " Seq" << (int)recv[3];
28        cout << endl;
29        cout << "发送:";
30        output(send[2]);
31        cout << " Length:" << (USHORT)((unsigned char)send[4] << 8) +
32            (USHORT)(unsigned char)send[5];
33        cout << " Checksum:" << ck;
34        cout << " Seq:" << curSeq;

```

```

29         cout << endl;
30         curSeq = curSeq == 0 ? 1 : 0;
31         break;
32     }
33 }

```

(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。

```

1 void send_message(char* message, int lent) {
2     int package_num = lent / Mlenx + (lent % Mlenx != 0); //看要发送几个包
3     for (int i = 0; i < package_num; i++) {
4         bool last = (i == (package_num - 1)); //判断是否是最后一个包
5         int length = last ? lent - (package_num - 1) * Mlenx : Mlenx;
6         //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
7         send_package(message + i * Mlenx, length, last);
8
9         if (i % 10 == 0)
10            printf("此文件已经发送%.2f%%\n", (float)i / package_num *
11                100);
12    }
13 }

```

然后就是发送端的投递过程:

```

1 bool send_package(char* message, int lent, bool last) {
2
3     char* tmp;
4     int tmp_len;
5
6     int length = lent % 2 == 0 ? lent : lent + 1;
7     //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
8     //以要进行长度补全
9     tmp = new char[length + 6];
10    tmp[length + 5] = 0;
11    //若是最后一个传输数据的包, 标志则是EF, 不是则是SF
12    if (last) {
13        tmp[2] = EF;
14    }
15    else {
16        tmp[2] = SF;
17    }
18    tmp[3] = curSeq;
19
20    tmp[4] = (lent + 6) >> 8;
21    tmp[5] = (lent + 6) & 0xff;
22    //cout << tmp[3] << endl;
23 }

```

```

22 //将文件数据导入到包中
23 for (int i = 6; i < lent + 6; i++)
24 tmp[i] = message[i - 6];
25 USHORT ck = checksum(tmp + 2, length + 4); //计算校验和
26 tmp[0] = ck >> 8;
27 tmp[1] = ck & 0xff;
28
29 tmp_len = lent + 6;
30 int send_to_cnt = 0;
31 while (1) {
32     send_to_cnt++; //有重传限制
33     //发送数据报
34     sendto(client, tmp, tmp_len, 0, (sockaddr*)&serverAddr,
35           sizeof(serverAddr));
36     int begintime = clock();
37     char recv[6];
38     int lentmp = sizeof(serverAddr);
39     int fail_send = 0;
40     while (recvfrom(client, recv, 6, 0, (sockaddr*)&serverAddr, &
41                     lentmp) == SOCKET_ERROR)
42     if (clock() - begintime > RTO) {
43         //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
44         fail_send = 1;
45         if (send_to_cnt == 10) {
46             wave_hand();
47             return false;
48         }
49         break;
50     }
51     double sampleRTT = clock() - begintime;
52     if (fail_send == 0 && checksum(recv, 6) == 0 && recv[2] ==
53         ACK && recv[3] == curSeq) {
54         addSampleRTT(sampleRTT);
55         curSeq = curSeq == 0 ? 1 : 0;
56         return true;
57     }
58 }
59 }

```

接收端:

```

1 void recv_message(char* message, int& len_recv) {
2     char recv[Mlenx + 6];
3     int lentmp = sizeof(clientAddr);
4     len_recv = 0;
5     while (1) {
6         while (1) {
7             //这一层循环是用来接收报文, 并且判断正确的
8             //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
13         *)&clientAddr, &lentmp) == SOCKET_ERROR);
14     char send[6];
15     if (checksum(recv, Mlenx + 6) == 0 && recv[3] ==
16         curSeq) {
17         //收到正确的报文
18         send[2] = ACK;
19         send[3] = curSeq;
20         send[4] = 6 >> 8;
21         send[5] = 6 & 0xff;
22         USHORT ck = checksum(send + 2, 4);
23
24         send[0] = ck >> 8;
25         send[1] = ck & 0xff;
26         sendto(server, send, 6, 0, (sockaddr*)&
27             clientAddr, sizeof(clientAddr));
28
29         cout << "收到:";
30         output(recv[2]);
31         cout << " Length:" << (USHORT)((unsigned char
32             )recv[4] << 8) + (USHORT)(unsigned char)
33             recv[5];
34         cout << " Checksum:" << (USHORT)((unsigned
35             char)recv[0] << 8) + (USHORT)(unsigned
36             char)recv[1];
37         cout << " Seq:" << (int)recv[3];
38         cout << endl;
39         cout << "发送:";
40         output(send[2]);
41         cout << " Length:" << 6;
42         cout << " Checksum:" << ck;
43         cout << " Seq:" << curSeq;
44         cout << endl;
45         curSeq = curSeq == 0 ? 1 : 0;
46         break;
47     }
48     else if (checksum(recv, Mlenx + 6) == 0 && recv[3] !=
49         curSeq) {
50         //这个是接收方ACK丢失，发送方重传的情况
51         send[2] = ACK;
52         send[3] = recv[3]; //返回的序号是与当前curSeq
53             相反的序号
54         send[4] = 6 >> 8;
55         send[5] = 6 & 0xff;
56         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：

$$\text{EstimatedRTT} = (1 - \alpha) \text{EstimatedRTT} + \alpha \text{SampleRTT}$$

$$\text{DevRTT} = (1 - \alpha) \text{DevRTT} + \alpha |\text{EstimatedRTT} - \text{SampleRTT}|$$

(推荐值为 1/4)

$$\text{RTO} = \mu \text{EstimatedRTT} + \sigma \text{DevRTT}$$

$$(\mu = 1, \sigma = 4)$$

```

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

```

```

3   DevRTT = 0.75 * DevRTT + 0.25 * abs(sampleRTT - EstimatedRTT);
4   RTO = EstimatedRTT + 4 * DevRTT;
5 }

```

四、 实验结果

这里是用发送方发送 1.jpg 来进行实验验证

```

Microsoft Visual Studio 调试控制台
请输入接收方ip地址:127.0.0.1
请输入端口号:6666
(0) 1.jpg
(1) 2.jpg
(2) 3.jpg
(3) ComputerNetwork3-1-main.zip
(4) 屏幕截图 2021-10-30 121622.png
(5) 计算机网络-实验3-要求与评分标准-2021-1.docx
输入要发送的文件序号: 0
开始两次握手
两次握手完成, 准备发送文件名
此文件已经发送0.00%
文件名发送完成, 准备发送文件内容
此文件已经发送0.00%
此文件已经发送5.38%
此文件已经发送10.75%
此文件已经发送16.13%
此文件已经发送21.51%
此文件已经发送26.88%
此文件已经发送32.26%
此文件已经发送37.63%
此文件已经发送43.01%
此文件已经发送48.39%
此文件已经发送53.76%
此文件已经发送59.14%
此文件已经发送64.52%
此文件已经发送69.89%
此文件已经发送75.27%
此文件已经发送80.65%
此文件已经发送86.02%
此文件已经发送91.40%
此文件已经发送96.77%
文件内容发送完成, 准备两次挥手
连接断开
689.474
E:\VS2019 code\ComputerNetwork_lab3_1\Debug\client.exe (进程 12996)已退出, 代码为 0。
按任意键关闭此窗口...

```

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。

```

请输入服务器端口:6666
等待发送端连接
收到:ACK:0 SYN:1 SF:0 EF:0 FIN:0 Length:6 Checksum:65017 Seq:0
发送:ACK:1 SYN:1 SF:0 EF:0 FIN:0 Length:6 Checksum:64761 Seq:0
两次握手完成, 准备接收数据

```

然后是先发送文件名称给接收方:

```

发送:ACK:1 SYN:1 SF:0 EF:0 FIN:0 Length:6 Checksum:64761 Seq:0
两次握手完成, 准备接收数据
收到:ACK:0 SYN:0 SF:0 EF:1 FIN:0 Length:11 Checksum:62804 Seq:1
发送:ACK:1 SYN:0 SF:0 EF:0 FIN:0 Length:6 Checksum:65272 Seq:1
文件名接收完成
文件名: 1.jpg
文件名长度: 5 bytes

```

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)

```
收到:ACK:0 SYN:0 SF:1 EF:0 FIN:0 Length:10006 Checksum:18407 Seq:1
发送:ACK:1 SYN:0 SF:0 EF:0 FIN:0 Length:6 Checksum:65272 Seq:1
收到:ACK:0 SYN:0 SF:1 EF:0 FIN:0 Length:10006 Checksum:49758 Seq:0
发送:ACK:1 SYN:0 SF:0 EF:0 FIN:0 Length:6 Checksum:65273 Seq:0
收到:ACK:0 SYN:0 SF:0 EF:1 FIN:0 Length:7359 Checksum:25740 Seq:1
发送:ACK:1 SYN:0 SF:0 EF:0 FIN:0 Length:6 Checksum:65272 Seq:1
文件数据长度: 1857353 bytes
文件数据接收完成
收到:ACK:0 SYN:0 SF:0 EF:0 FIN:1 Length:6 Checksum:61433 Seq:0
```

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。

```
文件数据接收完成
收到:ACK:0 SYN:0 SF:0 EF:0 FIN:1 Length:6 Checksum:61433 Seq:0
发送:ACK:1 SYN:0 SF:0 EF:0 FIN:1 Length:6 Checksum:61177 Seq:0
已断开连接
吞吐率:52133.3Kbps
请按任意键继续...
```