

实验3-1报告

2013605 张文迪

实验要求

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：建立连接、差错检测、确认重传等。流量控制采用停等机制，完成给定测试文件的传输。

协议设计

本次实验采用 RDT3.0 实现可靠数据传输

报文结构:

1	2	3	4	5	6	7	8			
Seq										
Ack										
DataSize										
Checksum								Flag		
Data(MAX_LENGTH=10240Bytes)										

定义:

```
1  #define MAX_DATA_SIZE 10240
2  struct RDTHdr
3  {
4      unsigned int seq;//序列号，发送端
5      unsigned int ack;//确认号码，发送端和接收端用来控制
6      unsigned short checksum;//校验和 16位
7      unsigned int dataSize;      //标识发送的数据的长度,边界判断与校验和
8      char flag;                  //ACK, FIN, SYN, END
9
10     RDTHdr()
11     {
12         this->seq=this->ack=0;
13         this->checksum=this->dataSize=this->flag=0;
14     }
15 };
16
17 struct RDTPacket
```

```

18 {
19     /* data */
20     RDTHed head;
21     char data[MAX_DATA_SIZE];
22 };

```

如图所示，数据报文由报文头和数据部分组成。其都为定长。

报文头

Seq：在RDT3.0只有0和1两种取值。表示发送的报文的序列号，接收端识别并确认。

Ack：与SEQ对应，只有0和1两种取值。表示接收端对收到的报文的序列号的确认。

CheckSum：校验和，可以确认报文在传输过程中是否受到损坏，用于差错检测。

Flag：用于握手和挥手过程的标识。主要用到了低四位标识不同的包。

DataSize：标识数据部分实际有效大小，用来确定传输文件的边界。

SYN 0x1：用于三次握手

ACK 0x2：用于三次握手和四次挥手

FIN 0x4：用于四次挥手

END 0x8：用于标识单个文件传输完毕。

报文数据

由于路由程序转包的最大包大小为 15000 字节，所以报文总的数据大小不能超过15000字节，所以设计报文的数据部分大小为10240字节。

差错检测

利用数据报中携带冗余位（校验和域段）来检测数据报传输过程中出现的差错。

发送端：

- 校验和域段清0，将数据报用0补齐为16位整数倍
- 将伪头部和数据报一起看成16位整数序列
- 进行 16 位二进制反码求和运算，计算结果取反写入校验和域段

接收端：

- 无需清空校验位
- 计算与发送端相同
- 如果计算结果位全0，没有检测到错误；否则，说明数据报存在差错

```

1 //计算校验和
2 unsigned short Calchecksum(unsigned short *packet, unsigned int dataSize)
3 {
4     unsigned long sum = 0;
5     int count = (dataSize + 1) / 2;
6
7     unsigned short *temp = new unsigned short [count];
8     memset(temp, 0, 2 * count);
9     memcpy(temp, packet, dataSize);

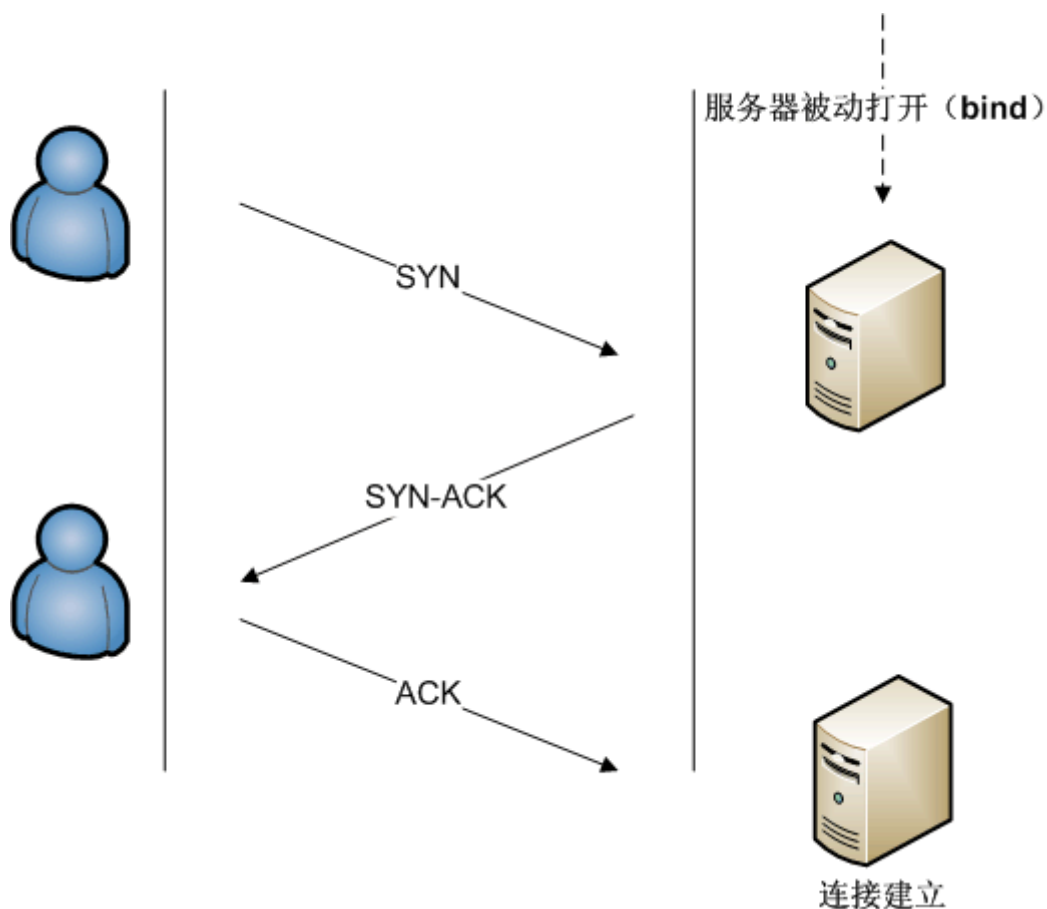
```

```

10
11     while (count--) {
12         sum += *temp++;
13         if (sum & 0xFFFF0000) {
14             sum &= 0xFFFF;
15             sum++;
16         }
17     }
18     return ~(sum & 0xFFFF);
19 }

```

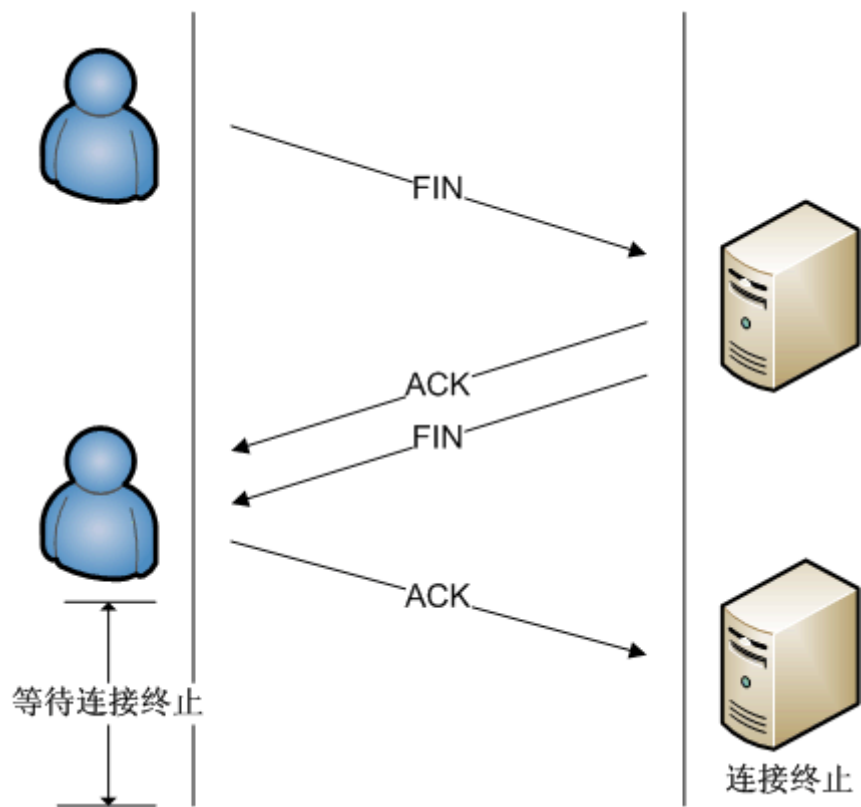
建立连接：三次握手



类似于TCP三次握手过程：

1. 客户端向服务器端发送一个SYN包，请求一个主动打开。进入SYN_SEND状态
2. 服务器端收到一个合法的SYN包后，回送一个SYN/ACK。进入SYN_RECV状态。
3. 客户端收到SYN/ACK包后，发送一个ACK包然后进入Established状态。当服务器端收到这个ACK包的时候。进入Established状态。

关闭连接：四次挥手



类似于TCP的四次挥手：

1. 第一次挥手：客户端向服务器发送FIN包，然后进入FIN-WAIT-1状态表示本方的数据发送全部结束，等待连接另一端的ACK确认包或FIN&ACK请求包。
2. 第二次挥手：服务器发送一个ACK给客户端，然后客户端进入FIN-WAIT-2状态，这时可以接收数据，但不再发送数据。服务器进入CLOSE-WAIT状态，这时可以发送数据，但不再接收数据。
3. 第三次挥手，服务器发送一个FIN给客户端，进入LAST-ACK状态，等待确认包
4. 第四次挥手，客户端收到FIN后，发送一个ACK包，同时进入TIME-WAIT状态，等待足够时间以确保被动关闭端收到了终止请求的确认包。然后server进入CLOSED 状态，完全没有连接。

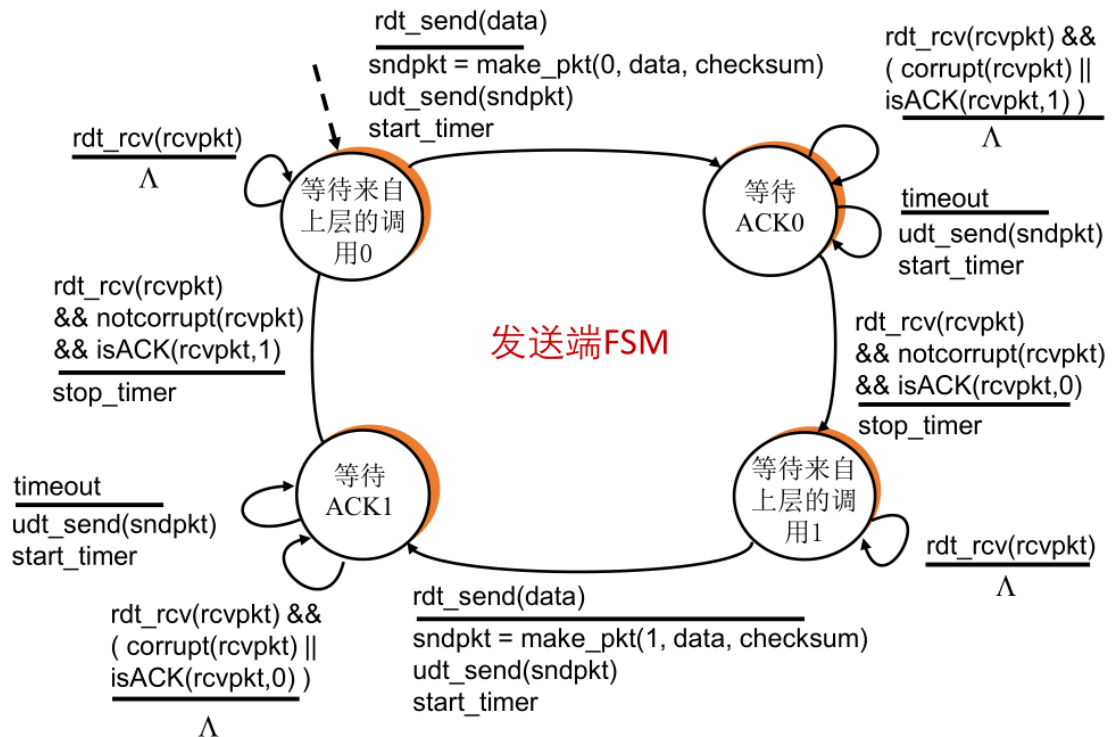
可靠数据传输

本次实验采用 RDT3.0 实现可靠数据传输

发送端

发送端状态机：

■ rdt3.0: 发送端状态机



- 发送端给定的序列号只有两个0和1，当发送出一个数据包后，便转换状态等待确认包。
- 发送端接收到重复的ACK或者校验和出现错误，便什么也不做（不切换状态也不停止计时）若超时则重传当前的分组。
- 接收到正确的确认包会停止计时并切换状态

```

1 void send(char *fileBuffer, size_t filelen, SOCKET &socket, SOCKADDR_IN
  &addr) {
2     u_long imode = 1;
3     ioctlsocket(socket, FIONBIO, &imode); //先进入非阻塞模式
4     int packetNum = int(filelen / MAX_DATA_SIZE); int remain = filelen %
MAX_DATA_SIZE ? 1 : 0;
5     packetNum += remain;
6     int num = 0; //数据包的索引
7     int stage = 0; //有限自动状态机
8     int addrLen = sizeof(addr);
9     char *dataBuffer = new char[MAX_DATA_SIZE], *pktBuffer = new
char[sizeof(RDTPacket)];
10    RDTPacket sendPkt, rcvPkt;
11    cout << "总共需要传输" << packetNum << "个数据包" << endl;
12    clock_t start;
13    while (true) {
14        int dataSize;
15        if (num == packetNum) {
16            RDTHed endHead;
17            setEND(endHead.flag);
18            endHead.checkSum = CalcheckSum((u_short *) &endHead,
sizeof(RDTHed));
19            memcpy(pktBuffer, &endHead, sizeof(RDTHed));
20            sendto(socket, pktBuffer, sizeof(RDTHed), 0, (SOCKADDR *)
&addr, addrLen);
21
22            while (recvfrom(socket, pktBuffer, sizeof(RDTHed), 0,
(SOCKADDR *) &addr, &addrLen) <= 0) {

```

```

23         if (clock() - start >= MAX_TIMEOUT) {
24             memcpy(pktBuffer, &endHead, sizeof(RDTHead));
25             sendto(socket, pktBuffer, sizeof(RDTHead), 0, (SOCKADDR
*) &addr, addrLen);
26             start = clock();
27         }
28     }
29     RDTHead serverACK;
30     memcpy(&serverACK, pktBuffer, sizeof(RDTHead));
31     if(isACK(serverACK.flag))
32         cout<<"文件传输完成"<<endl;
33     return;
34 }
35 switch (stage) {
36     case 0:
37         dataSize = MAX_DATA_SIZE;
38         if((num+1)*MAX_DATA_SIZE>filelen){
39             dataSize = filelen-num*MAX_DATA_SIZE;
40         }
41         memcpy(dataBuffer, fileBuffer + num * MAX_DATA_SIZE,
dataSize);
42         sendPkt = mkPacket(0, dataBuffer, dataSize);
43         memcpy(pktBuffer, &sendPkt, sizeof(RDTPacket));
44         sendto(socket, pktBuffer, sizeof(RDTPacket), 0, (SOCKADDR
*) &addr, addrLen);
45         start = clock();//计时
46         stage = 1;
47         break;
48     case 1:
49         //超时的情况
50         while (recvfrom(socket, pktBuffer, sizeof(RDTPacket), 0,
(SOCKADDR *) &addr, &addrLen) <= 0) {
51             if (clock() - start >= MAX_TIMEOUT) {
52                 sendto(socket, pktBuffer, sizeof(RDTPacket), 0,
(SOCKADDR *) &addr, addrLen);
53                 cout << num << "号数据包超时重传" << endl;
54                 start = clock();
55             }
56         }
57         memcpy(&rcvPkt, pktBuffer, sizeof(RDTPacket));
58
59         //收到重复的包或者校验和错误
60         if (rcvPkt.head.ack == 1 || Calchecksum((u_short *)
&rcvPkt, sizeof(RDTPacket)) != 0) {
61             stage = 1;
62             break;
63         }
64         if (rcvPkt.head.ack == 0 || Calchecksum((u_short *)
&rcvPkt, sizeof(RDTPacket)) == 0) {
65             stage = 2;
66             num++;
67             break;
68         }
69         break;
70     case 2:
71         dataSize = MAX_DATA_SIZE;
72         if((num+1)*MAX_DATA_SIZE>filelen){
73             dataSize = filelen-num*MAX_DATA_SIZE;

```

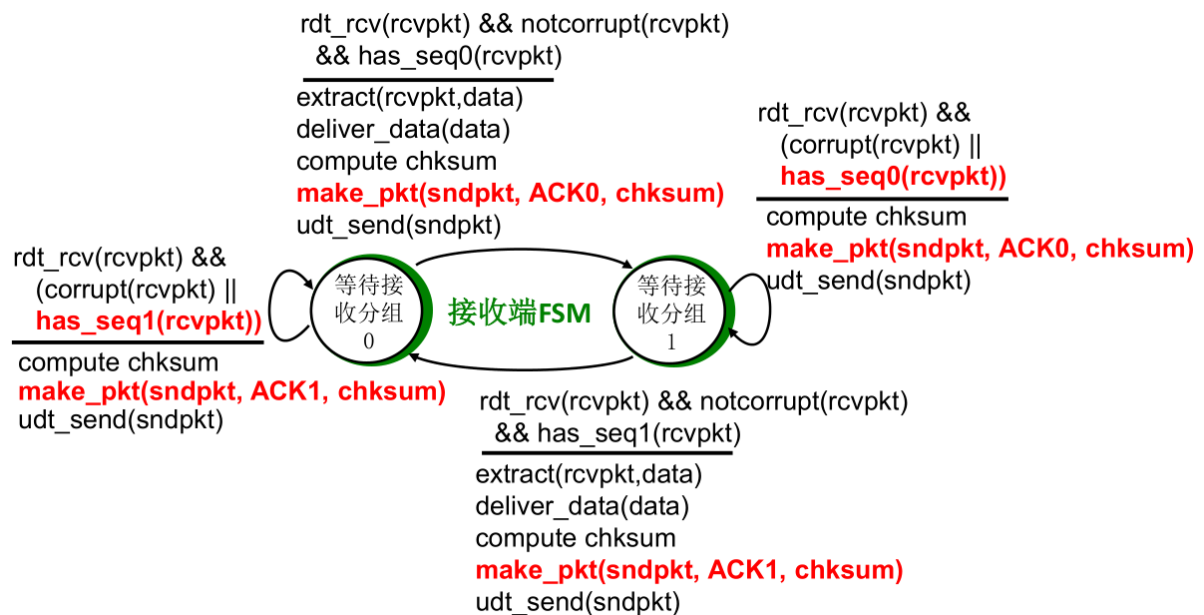
```

74         }
75         memcpy(dataBuffer, fileBuffer + num * MAX_DATA_SIZE,
dataSize);
76         sendPkt = mkPacket(1, dataBuffer, dataSize);
77         memcpy(pktBuffer, &sendPkt, sizeof(RDTPacket));
78         sendto(socket, pktBuffer, sizeof(RDTPacket), 0, (SOCKADDR
*) &addr, addrLen);
79         start = clock();
80         stage = 3;
81         break;
82     case 3:
83         //超时情况
84         while (recvfrom(socket, pktBuffer, sizeof(RDTPacket), 0,
(SOCKADDR *) &addr, &addrLen) <= 0) {
85             if (clock() - start >= MAX_TIMEOUT) {
86                 sendto(socket, pktBuffer, sizeof(RDTPacket), 0,
(SOCKADDR *) &addr, addrLen);
87                 cout << num << "号数据包超时重传" << endl;
88                 start = clock();
89             }
90         }
91         memcpy(&rcvPkt, pktBuffer, sizeof(RDTPacket));
92         //收到重复的包或者校验和错误
93         if (rcvPkt.head.ack == 0 || Calchecksum((u_short *)
&rcvPkt, sizeof(RDTPacket)) != 0) {
94             stage = 3;
95             break;
96         }
97         if (rcvPkt.head.ack == 0 || Calchecksum((u_short *)
&rcvPkt, sizeof(RDTPacket)) == 0) {
98             stage = 0;
99             num++;
100             break;
101         }
102         break;
103     }
104 }
105 }

```

接收端

接收端状态机：



- 当接收到带有正确序列号（当前状态等待的）的数据包时，会发送与接收到的数据包序列号相同的ACK给发送端，并切换状态
- 当接收到带有错误序列号（不是状态等待的）的数据包或者校验和错误时，会发送上一个数据包序列号给发送端，并维持当前状态

```

1  bool recv(char *fileBuffer, SOCKET &socket, SOCKADDR_IN &addr, unsigned long
    &filelen) {
2      int event = 0; //状态
3      int num = 0;   //数据包个数
4      int dataSize;  //数据包数据段长度
5      int addrLen = sizeof(addr);
6      char *pktBuffer = new char[sizeof(RDTPacket)];
7      RDTPacket rcvPkt, sendPkt;
8      RDTHead overHead;
9      while (true) {
10         memset(pktBuffer, 0, sizeof(RDTPacket));
11         switch (event) {
12             case 0:
13                 //先确认是不是发送的结束包
14                 recvfrom(socket, pktBuffer, sizeof(RDTPacket), 0, (SOCKADDR
15 *) &addr, &addrLen);
16                 memcpy(&overHead, pktBuffer, sizeof(RDTHead));
17                 if (isEND(overHead.flag)) {
18                     cout << "传输完毕" << endl;
19                     RDTHead endPacket;
20                     setACK(endPacket.flag);
21                     endPacket.checkSum = CalcheckSum((u_short *) &endPacket,
22 sizeof(RDTHead));
23                     memcpy(pktBuffer, &endPacket, sizeof(RDTHead));
24                     sendto(socket, pktBuffer, sizeof(RDTHead), 0, (SOCKADDR
25 *) &addr, addrLen);
26                     return true;
27                 }
28                 memcpy(&rcvPkt, pktBuffer, sizeof(RDTPacket));

```



```

29         //校验位不正确或收到重复的包
30         if (rcvPkt.head.seq == 1 || Calchecksum((u_short *) &rcvPkt,
sizeof(RDTPacket)) != 0) {
31             sendPkt = mkPacket(1);
32             memcpy(pktBuffer, &sendPkt, sizeof(RDTPacket));
33             sendto(socket, pktBuffer, sizeof(RDTPacket), 0,
(SOCKADDR *) &addr, addrLen);
34             event = 0;
35             cout << num << "号数据包重复或损坏, 抛弃" << endl;
36             break;
37         }
38
39
40         //收到正确的数据包
41         if (rcvPkt.head.seq == 0 || Calchecksum((u_short *) &rcvPkt,
sizeof(RDTPacket)) == 0){
42             dataSize = rcvPkt.head.dataSize;
43             memcpy(fileBuffer + filelen, rcvPkt.data, dataSize);
44             filelen += dataSize;
45             //发送确认包
46             sendPkt = mkPacket(0);
47             memcpy(pktBuffer, &sendPkt, sizeof(RDTPacket));
48             sendto(socket, pktBuffer, sizeof(RDTPacket), 0,
(SOCKADDR *) &addr, addrLen);
49             event = 1;
50             num++;
51             break;
52         }
53         break;
54         case 1:
55             //先确认是不是发送的结束包
56             recvfrom(socket, pktBuffer, sizeof(RDTPacket), 0, (SOCKADDR
*) &addr, &addrLen);
57             memcpy(&overHead, pktBuffer, sizeof(RDTHHead));
58             if (isEND(overHead.flag)) {
59                 cout << "传输完毕" << endl;
60                 RDTHHead endPacket;
61                 setACK(endPacket.flag);
62                 endPacket.checkSum = Calchecksum((u_short *) &endPacket,
sizeof(RDTHHead));
63                 memcpy(pktBuffer, &endPacket, sizeof(RDTHHead));
64                 sendto(socket, pktBuffer, sizeof(RDTHHead), 0, (SOCKADDR
*) &addr, addrLen);
65                 return true;
66             }
67             memcpy(&rcvPkt, pktBuffer, sizeof(RDTPacket));
68             if (rcvPkt.head.seq == 0 || Calchecksum((u_short *) &rcvPkt,
sizeof(RDTPacket)) != 0) {
69                 sendPkt = mkPacket(0);
70                 memcpy(pktBuffer, &sendPkt, sizeof(RDTPacket));
71                 sendto(socket, pktBuffer, sizeof(RDTPacket), 0,
(SOCKADDR *) &addr, addrLen);
72                 event = 1;
73                 cout << num<< "号数据包重复或损坏, 抛弃" << endl;
74                 break;
75             }
76             //正确接收的情况

```

```

77         if (rcvPkt.head.seq == 1 || Calchecksum((u_short *) &rcvPkt,
sizeof(RDTPacket)) == 0) {
78             dataSize = rcvPkt.head.dataSize;
79             memcpy(fileBuffer + filelen, rcvPkt.data, dataSize);
80             filelen += dataSize;
81             //发送确认包
82             sendPkt = mkPacket(1);
83             memcpy(pktBuffer, &sendPkt, sizeof(RDTPacket));
84             sendto(socket, pktBuffer, sizeof(RDTPacket), 0,
(SOCKADDR *) &addr, addrLen);
85             event = 0;
86             num++;
87             break;
88         }
89         break;
90     }
91 }
92 }

```

其他具体实现

文件边界

先获取要发送文件的大小

```

1 // 这是一个存储文件(夹)信息的结构体，其中有文件大小和创建时间、访问时间、修改时间等
2 struct stat statbuf;
3 // 提供文件名字符串，获得文件属性结构体
4 stat(fileName, &statbuf);
5 // 获取文件大小
6 size_t fileLen = statbuf.st_size;
7 char *fileBuffer = new char[fileLen];
8 myfile.read(fileBuffer, fileLen);
9 myfile.close();

```

确认要发送的总的数据包个数：

```

1 int packetNum = int(filelen / MAX_DATA_SIZE);
2 int remain = filelen % MAX_DATA_SIZE ? 1 : 0;
3 packetNum += remain;
4 int num = 0; //当前发送的数据包的索引

```

确认发送结束包的时机：

```

1 if (num == packetNum) {
2     RDTHed endHead;
3     setEND(endHead.flag);
4     endHead.checkSum = Calchecksum((u_short *) &endHead,
sizeof(RDTHed));
5     memcpy(pktBuffer, &endHead, sizeof(RDTHed));
6     sendto(socket, pktBuffer, sizeof(RDTHed), 0, (SOCKADDR *)
&addr, addrLen);
7     while (recvfrom(socket, pktBuffer, sizeof(RDTHed), 0, (SOCKADDR
*) &addr, &addrLen) <= 0) {
8         if (clock() - start >= MAX_TIMEOUT) {

```

```

9         memcpy(pktBuffer, &endHead, sizeof(RDTHed));
10        sendto(socket, pktBuffer, sizeof(RDTHed), 0, (SOCKADDR
*) &addr, addrLen);
11        start = clock();
12    }
13    }
14    RDTHed serverACK;
15    memcpy(&serverACK, pktBuffer, sizeof(RDTHed));
16    if(isACK(serverACK.flag))
17        cout<<"文件传输完成"<<endl;
18    return;
19    }

```

阻塞与非阻塞

当我们要计时接收下一个数据包的时间时，我们需要将recv函数调整为非阻塞函数：

```

1  u_long imode = 1;
2  ioctlsocket(sockClient, FIONBIO, &imode); //非阻塞
3
4  u_long imode = 0;
5  ioctlsocket(sockClient, FIONBIO, &imode); //阻塞

```

建立连接：三次握手

客户端

```

1  //三次握手建立连接，只需要发送协议头部分。
2  bool ConServer(SOCKET socket, SOCKADDR_IN serverAddr)
3  {
4      //第一次握手
5      int addrLen = sizeof(serverAddr);
6      RDTHed clientSYN;
7      setSYN(clientSYN.flag);
8      clientSYN.checkSum=CalcheckSum((unsigned short *)&clientSYN,
sizeof(clientSYN));
9      char buffer[sizeof(clientSYN)];
10     memset(buffer, 0, sizeof(clientSYN));
11     sendRDTHed(buffer, &clientSYN, socket, serverAddr);
12     cout << "第一次握手,进入SYN_SEND状态" << endl;
13     //第二次握手
14     RDTHed serverSYN_ACK;
15     u_long mode = 1;
16     ioctlsocket(socket, FIONBIO, &mode); //设置为非阻塞
17     clock_t start =clock();
18     while (recvfrom(socket, buffer, sizeof(serverSYN_ACK), 0, (SOCKADDR *)
&serverAddr, &addrLen) <= 0) {
19         if (clock() - start >= MAX_TIMEOUT) {
20             cout<<"第一次握手超时重传"<<endl;
21             memcpy(buffer, &clientSYN, sizeof(clientSYN));
22             sendto(socket, buffer, sizeof(clientSYN), 0, (SOCKADDR *)
&serverAddr, addrLen);
23             start = clock();
24         }
25     }

```

```

26     memcpy(&serverSYN_ACK, buffer, sizeof(serverSYN_ACK));
27     if (isSYN_ACK(serverSYN_ACK.flag)&& (Calchecksum((u_short *)
28 &serverSYN_ACK, sizeof(RDTHed)) == 0)) {
29         cout << "第二次握手成功" << endl;
30     } else {
31         cout << "第二次握手失败" << endl;
32         return false;
33     }
34     //第三次握手
35     RDTHed clientACK;
36     setACK(clientACK.flag);
37     clientACK.checkSum=Calchecksum((unsigned short *)&clientACK,
38 sizeof(clientACK));
39     memcpy(buffer, &clientACK, sizeof(clientACK));
40     if(sendto(socket, buffer, sizeof(clientACK), 0, (SOCKADDR *)
41 &serverAddr, addrLen)==-1){
42         return false;
43     }
44     cout<<"第三次握手进入TIME-WAIT状态"<<endl;
45     start = clock();
46     while (clock() - start <= 2 * MAX_TIMEOUT) {
47         if (recvfrom(socket, buffer, sizeof(RDTHed), 0, (SOCKADDR *)
48 &serverAddr, &addrLen) <= 0)
49             continue;
50         //第三次握手确认包丢失
51         memcpy(buffer, &clientACK, sizeof(RDTHed));
52         sendto(socket, buffer, sizeof(RDTHed), 0, (sockaddr *) &serverAddr,
53 addrLen);
54         cout<<"第三次握手超时重传"<<endl;
55         start = clock();
56     }
57     cout<<"第三次握手进入Established状态"<<endl;
58     u_long imode = 0;
59     ioctlsocket(socket, FIONBIO, &imode); //阻塞
60     return true;
61 }

```

关闭连接：四次挥手

客户端

```

1  bool DisConServer(SOCKET clientSocket, SOCKADDR_IN serverAddr) {
2      int addrLen = sizeof(serverAddr);
3      char buffer[sizeof(RDTHed)];
4      RDTHed clientFIN;
5      setFIN_ACK(clientFIN.flag);
6      clientFIN.checkSum = Calchecksum((u_short *) &clientFIN,
7 sizeof(RDTHed));
8      memcpy(buffer, &clientFIN, sizeof(RDTHed));
9      sendto(clientSocket, buffer, sizeof(RDTHed), 0, (SOCKADDR *)
10 &serverAddr, addrLen);
11     cout<<"客户端发起第一次挥手进入FIN-WAIT-1状态"<<endl;
12     unsigned long imode = 1;
13     ioctlsocket(clientSocket, FIONBIO, &imode); //改为非阻塞模式
14     clock_t start = clock();

```

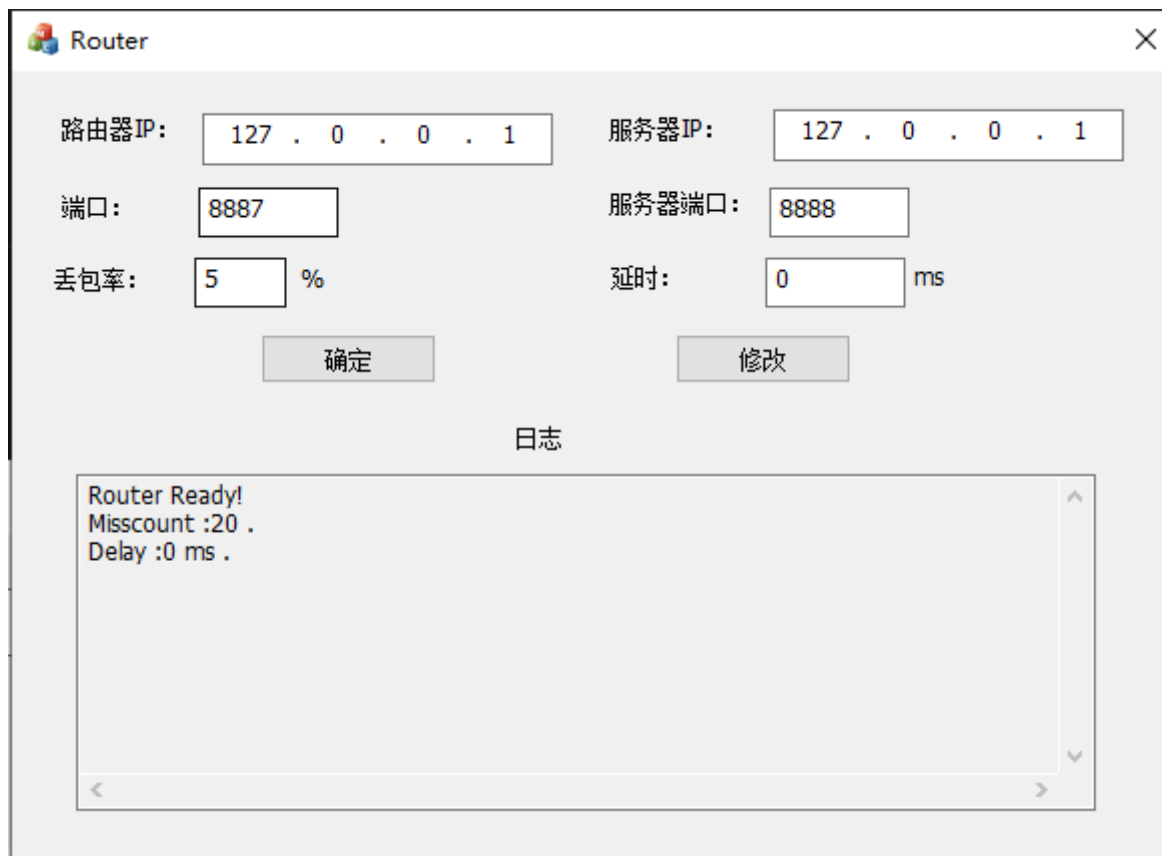
```

13     while (recvfrom(clientSocket, buffer, sizeof(RDTHed), 0, (sockaddr *)&serverAddr, &addrLen) <= 0) {
14         if (clock() - start >= MAX_TIMEOUT) {
15             cout<<"第一次挥手超时重传"<<endl;
16             memcpy(buffer, &clientFIN, sizeof(RDTHed));
17             sendto(clientSocket, buffer, sizeof(RDTHed), 0, (SOCKADDR *)&serverAddr, addrLen);
18             start = clock();
19         }
20     }
21     RDTHed serverACK;
22     memcpy(&serverACK, buffer, sizeof(RDTHed));
23     if ((isACK(serverACK.flag)) && (Calchecksum((u_short *) buffer, sizeof(RDTHed) == 0))) {
24         cout << "客户端进入FIN-WAIT-2状态" << endl;
25     } else {
26         cout << "错误" << endl;
27         return false;
28     }
29     imode = 0;
30     ioctlsocket(clientSocket, FIONBIO, &imode); //阻塞
31     recvfrom(clientSocket, buffer, sizeof(RDTHed), 0, (SOCKADDR *)&serverAddr, &addrLen);
32     RDTHed serverFIN;
33     memcpy(&serverFIN, buffer, sizeof(RDTHed));
34     if ((isFIN_ACK(serverFIN.flag)) && (Calchecksum((u_short *) buffer, sizeof(RDTHed) == 0))) {
35         cout << "第三次挥手" << endl;
36     } else {
37         cout << "错误" << endl;
38         return false;
39     }
40     imode = 1;
41     ioctlsocket(clientSocket, FIONBIO, &imode);
42     RDTHed clientACK;
43     setACK(clientACK.flag);
44     sendto(clientSocket, buffer, sizeof(RDTHed), 0, (SOCKADDR *)&serverAddr, addrLen);
45     start = clock();
46     cout<<"第四次挥手进入TIME-WAIT状态"<<endl;
47     while (clock() - start <= 2 * MAX_TIMEOUT) {
48         if (recvfrom(clientSocket, buffer, sizeof(RDTHed), 0, (SOCKADDR *)&serverAddr, &addrLen) <= 0)
49             continue;
50         //确认包丢失
51         cout<<"第四次挥手超时重传"<<endl;
52         memcpy(buffer, &clientACK, sizeof(RDTHed));
53         sendto(clientSocket, buffer, sizeof(RDTHed), 0, (sockaddr *)&serverAddr, addrLen);
54         start = clock();
55     }
56     cout << "第四次挥手进入closed状态" << endl;
57     closesocket(clientSocket);
58     return true;
59 }

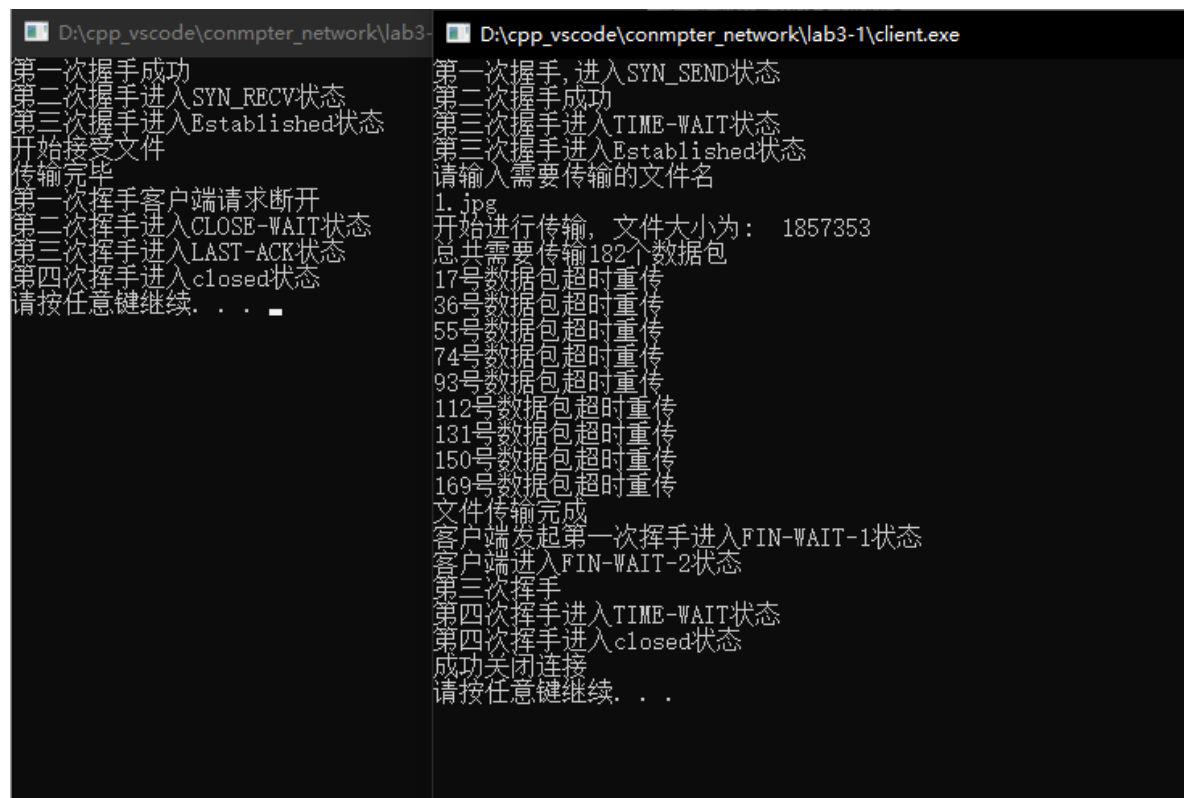
```

测试

- 将路程序的端口设置为8887，服务器端口与程序中指定的端口保持一致，客户端的目标地址设置为路由器地址。
- 设置丢包率为5%



得到的完整的传输过程：



如下图所示接收端得到的文件与测试文件中1.jpg完全相同

