



南开大学  
Nankai University

南 开 大 学

计 算 机 学 院

计算机网络实验报告

---

Lab 3.1 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

---

徐俊智

年级：2022 级

专业：计算机科学与技术

指导教师：吴英

2024 年 12 月 4 日

## 目录

一、 实验目的	1
二、 实验要求	1
三、 UDP 报文段格式	1
四、 建立连接 & 关闭连接	3
五、 差错检测	10
六、 发送文件	11
1. 停等机制 .....	14
2. 超时重传 .....	15
七、 接收文件	16
1. 接收确认 .....	18
八、 测试	19

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

## 二、 实验要求

1. 实现单向传输。
2. 对于每个任务要求给出详细的协议设计。
3. 给出实现的拥塞控制算法的原理说明。
4. 完成给定测试文件的传输，显示传输时间和平均吞吐率。
5. 性能测试指标：吞吐率、文件传输时延，给出图形结果并进行分析。
6. 完成详细的实验报告。
7. 编写的程序应该结构清晰，具有较好的可读性。
8. 现场演示。
9. 提交程序源码、可执行文件和实验报告。

## 三、 UDP 报文段格式

```
1  class Message {
2  public:
3      u_long flag;           // 伪首部
4      u_short seq;          // 序列号
5      u_short ack;          // 确认号
6      u_long len;           // 数据部分长度
7      u_long num;           // 数据包个数
8      u_short checksum;     // 校验和
9      char data[1024];      // 数据
10
11     Message() { memset(this, 0, sizeof(Message)); }
12
13     bool isSYN() { return this->flag & 1; }
14
15     bool isACK() { return this->flag & 2; }
16
17     bool isFIN() { return this->flag & 4; }
18
19     bool isSTART() { return this->flag & 8; }
20
21     bool isEND() { return this->flag & 16; }
```

Message 的成员有伪首部 flag, 发送号 seq 和确认号 ack, 数据部分长度 len, 校验和 checksum, 数据部分 data[1024], 其长度设置为 1024。

其中, flag 包含的属性有: SYN, ACK, FIN, START, END。SYN 和 FIN 分别用于建立和关闭连接, ACK 表示响应, START 和 END 表示文件传输的开始和结束。

其次, 定义了查询 flag 是否包含某个属性的函数。

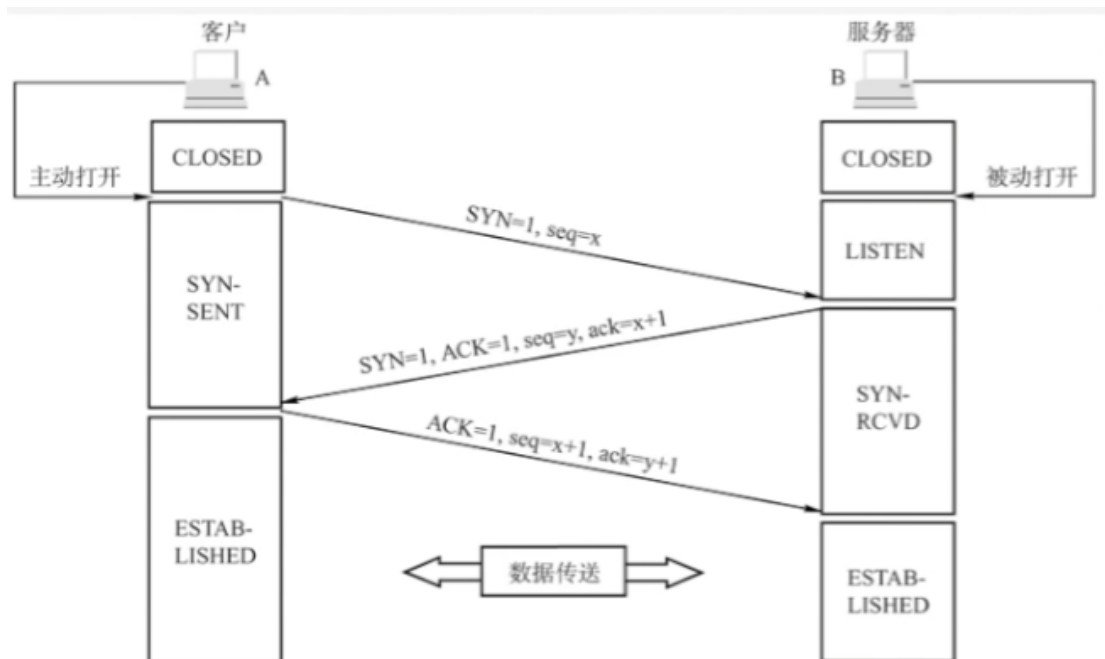
```
1 void setSYN() { this->flag |= 1; }
2 void setACK() { this->flag |= 2; }
3 void setFIN() { this->flag |= 4; }
4 void setSTART() { this->flag |= 8; }
5 void setEND() { this->flag |= 16; }
```

最后, 定义了计算校验和的函数 check\_sum 和检查数据包是否损坏的函数 packetCorruption。

```
1 void setChecksum() {
2     int sum = 0;
3     u_char* temp = (u_char*)this;
4     for (int i = 0; i < 8; i++) {
5         sum += (temp[i << 1] << 8) + temp[i << 1 | 1];
6         while (sum >= 0x10000) {
7             // 溢出
8             int t = sum >> 16; // 将最高位回滚添加至最低位
9             sum += t;
10        }
11    }
12    this->checksum = ~(u_short)sum; // 按位取反, 方便校验计算
13 }
14
15 bool packetCorruption() {
16     int sum = 0;
17     u_char* temp = (u_char*)this;
18     for (int i = 0; i < 8; i++) {
19         sum += (temp[i << 1] << 8) + temp[i << 1 | 1];
20         while (sum >= 0x10000) {
21             // 溢出
22             int t = sum >> 16; // 计算方法与设置校验和相同
23             sum += t;
24        }
25    }
26    // 把计算出来的校验和和报文中该字段的值相加, 如果等于 0xffff, 则校验成功
27    if (checksum + (u_short)sum == 65535)
28        return false;
29    return true;
30 }
```

## 四、 建立连接 & 关闭连接

通过三次握手建立连接



- 第一次握手：客户端发出连接请求报文， $SYN=1$ ， $seq=x=2000$ 。
- 第二次握手：服务器端收到来自客户端的连接请求报文后，通过标志位  $SYN=1$  知道了客户端请求建立连接。然后服务器端向客户端发出确认报文， $SYN=1$ ， $ACK=1$ ， $seq=y=1000$ ， $ack=x+1=2001$ 。
- 第三次握手：客户端收到来自服务器端的确认报文后，检查  $ACK$  是否为 1、 $ack$  是否为  $x+1$ 。如果正确，客户端向服务器端发出确认报文， $ACK=1$ ， $seq=x+1=2001$ ， $ack=y+1=1001$ 。

代码如下：

```

1 //Server.cpp
2 bool waitConnect() {
3     // 设置套接字为非阻塞模式
4     int mode = 1;
5     ioctlsocket(socketServer, FIONBIO, (u_long FAR*) & mode);
6
7     Message sendMsg, recvMsg;
8     clock_t start;
9
10    // 接收第一次握手消息
11    while (1) {
12        if (recvfrom(socketServer, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
13            clientAddr, &len) != SOCKET_ERROR) {
14            if (recvMsg.isSYN() && !recvMsg.packetCorruption()) {
15                cout << "服务器端接收到第一次握手消息！第一次握手成功！" <<
16                    endl;

```

```
15         break;
16     }
17 }
18 }
19
20 // 发送第二次握手消息
21 cout << "服务器端发送第二次握手消息! " << endl;
22 sendMsg.setSYN();
23 sendMsg.setACK();
24 sendMsg.seq = 1000;
25 sendMsg.ack = recvMsg.seq + 1;
26 sendMsg.setChecksum();
27 if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
    clientAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
28     cout << "服务器端发送第二次握手消息失败!" << endl;
29     cout << "当前网络状态不佳, 请稍后再试" << endl;
30     return 0;
31 }
32
33 // 接收第三次握手消息, 超时重传
34 start = clock();
35 while (1) {
36     if (recvfrom(socketServer, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
        clientAddr, &len) != SOCKET_ERROR) {
37         if (recvMsg.isACK() && recvMsg.ack == sendMsg.seq + 1 && !recvMsg.
            packetCorruption()) {
38             cout << "服务器端接收到第三次握手消息! 第三次握手成功! " <<
                endl;
39             break;
40         }
41     }
42     if (clock() - start > RTO) {
43         cout << "第二次握手超时, 服务器端重新发送第二次握手消息" << endl;
44         if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
            clientAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
45             cout << "服务器端发送第二次握手消息失败!" << endl;
46             cout << "当前网络状态不佳, 请稍后再试" << endl;
47             return 0;
48         }
49         start = clock();
50     }
51 }
52
53 // 设置套接字为阻塞模式
54 mode = 0;
55 ioctlsocket(socketServer, FIONBIO, (u_long FAR*) & mode);
56
57 return 1;
```

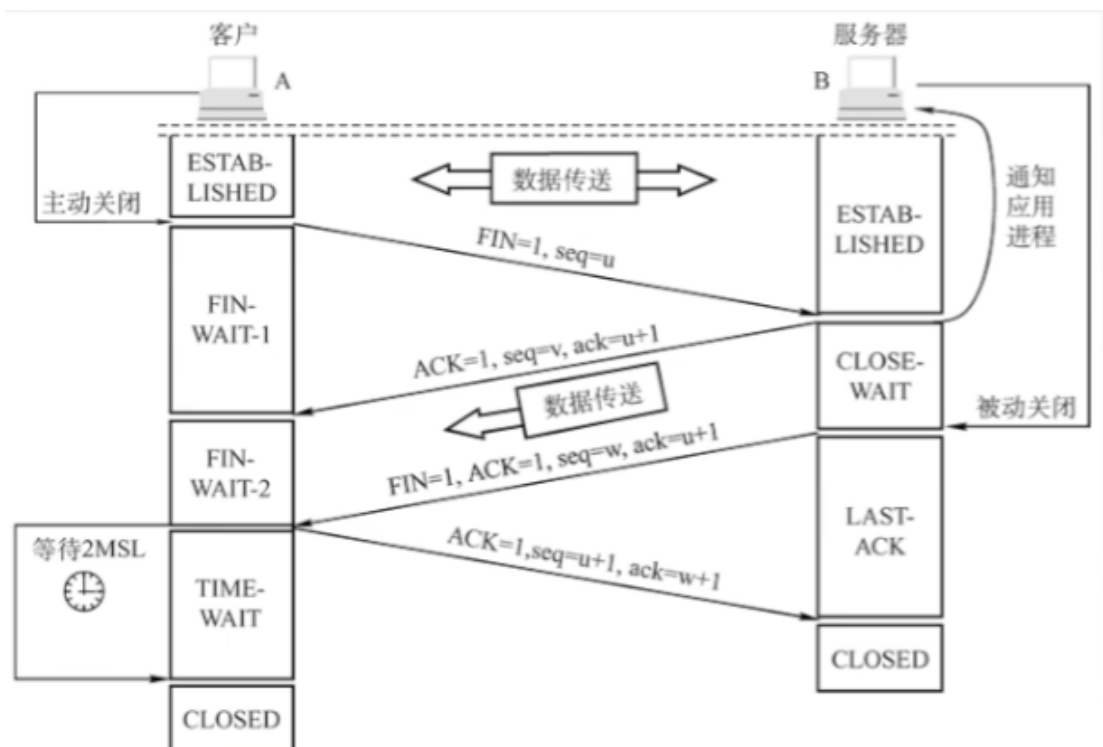
```
58 }
59
60 //Client.cpp
61 bool waitConnect() {
62     Message sendMsg, recvMsg;
63     clock_t start;
64
65     // 设置套接字为非阻塞模式
66     int mode = 1;
67     ioctlsocket(socketClient, FIONBIO, (u_long FAR*) & mode);
68
69     // 发送第一次握手消息
70     cout << "尝试建立连接! 客户端发送第一次握手消息" << endl;
71     sendMsg.setSYN();
72     sendMsg.seq = 2000;
73     sendMsg.setChecksum();
74     if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
75         serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
76         cout << "客户端发送第一次握手消息失败!" << endl;
77         cout << "当前网络状态不佳, 请稍后再试" << endl;
78         return 0;
79     }
80
81     // 接收第二次握手消息, 超时重传
82     start = clock();
83     while (1) {
84         if (recvfrom(socketClient, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
85             serverAddr, &len) != SOCKET_ERROR) {
86             if (recvMsg.isSYN() && recvMsg.isACK() && recvMsg.ack == sendMsg.
87                 seq + 1 && !recvMsg.packetCorruption()) {
88                 cout << "客户端接收到第二次握手消息! 第二次握手成功!" << endl
89                 ;
90                 break;
91             }
92         }
93         if (clock() - start > RTO) {
94             cout << "第一次握手超时, 客户端重新发送第一次握手消息" << endl;
95             if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
96                 serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
97                 cout << "客户端发送第一次握手消息失败!" << endl;
98                 cout << "当前网络状态不佳, 请稍后再试" << endl;
99                 return 0;
100             }
101             start = clock();
102         }
103     }
104
105     // 发送第三次握手消息
```

```

101     cout << "客户端发送第三次握手消息" << endl;
102     sendMsg.setACK();
103     sendMsg.seq = 2001;
104     sendMsg.ack = recvMsg.seq + 1;
105     sendMsg.setChecksum();
106     if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
107         serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
108         cout << "客户端发送第三次握手消息失败!" << endl;
109         cout << "当前网络状态不佳, 请稍后再试" << endl;
110         return 0;
111     }
112     return 1;
113 }

```

通过四次挥手关闭连接



- 第一次挥手：客户端发出连接释放报文， $FIN=1$ ， $seq=u=3000$ 。
- 第二次挥手：服务器端收到来自客户端的连接请求报文后，通过标志位  $FIN=1$  知道了客户端请求释放连接。然后服务器端向客户端发出确认报文， $ACK=1$ ， $seq=v=4000$ ， $ack=u+1=3001$ 。
- 第三次挥手：当服务器端确认数据传输完毕后，向客户端发送连接释放报文， $FIN=1$ ， $ACK=1$ ， $seq=w=5000$ ， $ack=u+1=3001$ 。
- 第四次挥手：客户端收到来自服务器端的连接释放报文后，通过标志位  $FIN=1$  知道了服务器端请求释放连接，然后客户端向服务器发出确认报文， $ACK=1$ ， $seq=u+1=3001$ ， $ack=w+1=5001$ 。

代码如下：



```
1 //Server.cpp
2 bool closeConnect(Message recvMsg) {
3     Message sendMsg;
4     clock_t start;
5
6     /*
7      第一次挥手在recv_file函数里面处理
8     */
9
10    // 设置套接字为非阻塞模式
11    int mode = 1;
12    ioctlsocket(socketServer, FIONBIO, (u_long FAR*) & mode);
13
14    // 发送第二次挥手消息
15    cout << "服务器端发送第二次挥手消息!" << endl;
16    sendMsg.setACK();
17    sendMsg.seq = 4000;
18    sendMsg.ack = recvMsg.seq + 1;
19    sendMsg.setChecksum();
20    if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
21        clientAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
22        cout << "服务器端发送第二次挥手消息失败!" << endl;
23        cout << "当前网络状态不佳, 请稍后再试" << endl;
24        return 0;
25    }
26
27    // 发送第三次挥手消息
28    cout << "服务器端发送第三次挥手消息!" << endl;
29    sendMsg.setFIN();
30    sendMsg.setACK();
31    sendMsg.seq = 5000;
32    sendMsg.ack = recvMsg.seq + 1;
33    sendMsg.setChecksum();
34    if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
35        clientAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
36        cout << "服务器端发送第三次挥手消息失败!" << endl;
37        cout << "当前网络状态不佳, 请稍后再试" << endl;
38        return 0;
39    }
40
41    // 接收第四次挥手消息, 超时重传
42    start = clock();
43    while (1) {
44        if (recvfrom(socketServer, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
45            clientAddr, &len) != SOCKET_ERROR) {
46            if (recvMsg.isACK() && recvMsg.ack == sendMsg.seq + 1 && !recvMsg
47                .packetCorruption()) {
```

```

44         cout << "服务器端接收到第四次挥手消息! 第四次挥手成功!" <<
45             endl;
46         break;
47     }
48     if (clock() - start > RTO) {
49         cout << "第三次挥手超时, 服务器端重新发送第三次挥手消息" << endl;
50         if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
51             clientAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
52             cout << "服务器端发送第三次挥手消息失败!" << endl;
53             cout << "当前网络状态不佳, 请稍后再试" << endl;
54             return 0;
55         }
56         start = clock();
57     }
58     return 0;
59 }
60
61
62 // Client.cpp
63 bool closeConnect() {
64     Message sendMsg, recvMsg;
65     clock_t start;
66
67     // 发送第一次挥手消息
68     cout << "尝试关闭连接! 客户端发送第一次挥手消息" << endl;
69     sendMsg.setFIN();
70     sendMsg.seq = 3000;
71     sendMsg.setChecksum();
72     if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
73         serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
74         cout << "客户端发送第一次挥手消息失败!" << endl;
75         cout << "当前网络状态不佳, 请稍后再试" << endl;
76         return 0;
77     }
78
79     // 接收第二次挥手消息, 超时重传
80     start = clock();
81     while (1) {
82         if (recvfrom(socketClient, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
83             serverAddr, &len) != SOCKET_ERROR) {
84             if (recvMsg.isACK() && recvMsg.ack == sendMsg.seq + 1 && !recvMsg
85                 .packetCorruption()) {
86                 cout << "客户端接收到第二次挥手消息! 第二次挥手成功!" <<
87                     endl;
88                 break;
89             }
90         }
91     }
92 }

```

```

86     }
87     if (clock() - start > RTO) {
88         cout << "第一次挥手超时, 客户端重新发送第一次挥手消息" << endl;
89         if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
90             serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
91             cout << "客户端发送第一次挥手消息失败!" << endl;
92             cout << "当前网络状态不佳, 请稍后再试" << endl;
93             return 0;
94         }
95         start = clock();
96     }
97 }
98 // 接收第三次挥手消息, 超时重传
99 start = clock();
100 while (1) {
101     if (recvfrom(socketClient, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
102         serverAddr, &len) != SOCKET_ERROR) {
103         if (recvMsg.isACK() && recvMsg.ack == sendMsg.seq + 1 && !recvMsg
104             .packetCorruption()) {
105             cout << "客户端接收到第三次挥手消息! 第三次挥手成功!" <<
106                 endl;
107             break;
108         }
109     }
110     if (clock() - start > RTO) {
111         cout << "第二次挥手超时, 客户端重新发送第二次挥手消息" << endl;
112         if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
113             serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
114             cout << "客户端发送第二次挥手消息失败!" << endl;
115             cout << "当前网络状态不佳, 请稍后再试" << endl;
116             return 0;
117         }
118         start = clock();
119     }
120 }
121 // 发送第四次挥手消息
122 cout << "客户端发送第四次挥手消息!" << endl;
123 sendMsg.setACK();
124 sendMsg.seq = 3001;
125 sendMsg.ack = recvMsg.seq + 1;
126 sendMsg.setChecksum();
127 if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
128     serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
129     cout << "客户端发送第四次挥手消息失败!" << endl;
130     cout << "当前网络状态不佳, 请稍后再试" << endl;
131     return 0;

```

```

128     }
129
130     return 1;
131 }

```

因为建立和关闭连接时都是客户先发起，服务器需要先阻塞在一个 while 循环里不断的接收消息，直到接收了客户发来的信息，才能继续往下执行。

## 五、 差错检测

计算检验和主要有三个步骤：

- 求和：把需要校验的数据看成以 16 位为单位的数字组成，依次进行二进制求和。
- 回卷：求和后超过 16 位的加到低 16 位。
- 取反：最后结果取反码就是检验和。

客户端发送数据包的时候，通过 setChecksum 函数设置校验和。服务器端接收数据包的时候，通过 packetCorruption 函数验证数据包是否正确。代码如下：

```

1 void setChecksum() {
2     this->checksum = 0; // 清0校验和字段
3     int dataLen = this->len; // 数据部分长度
4     int paddingLen = (16 - (dataLen % 16)) % 16; // 数据部分需要填0
5     char* paddedData = new char[dataLen + paddingLen]; // 填充后数据
6     memcpy(paddedData, this->data, dataLen);
7     memset(paddedData + dataLen, 0, paddingLen);
8     // 分段求和，并处理溢出
9     u_short* buffer = (u_short*)this;
10    int sum = 0;
11    for (int i = 0; i < (sizeof(Message) + paddingLen) / 2; i++) {
12        sum += buffer[i];
13        if (sum > 0xFFFF) {
14            sum = (sum & 0xFFFF) + (sum >> 16);
15        }
16    }
17    // 计算结果取反写入校验和字段
18    this->checksum = ~sum;
19    // 释放动态分配的内存
20    delete[] paddedData;
21 }
22 bool packetCorruption() {
23     // 计算数据长度并填充
24     int dataLen = this->len;
25     int paddingLen = (16 - (dataLen % 16)) % 16;
26     // 使用动态内存分配
27     char* paddedData = new char[dataLen + paddingLen];
28     memcpy(paddedData, this->data, dataLen);
29     memset(paddedData + dataLen, 0, paddingLen);

```

```

30 // 进行16 - bit段反码求和
31 u_short* buffer = (u_short*)this;
32 int sum = 0;
33 for (int i = 0; i < (sizeof(Message) + paddingLen) / 2; i++) {
34     sum += buffer[i];
35     if (sum > 0xFFFF) {
36         sum = (sum & 0xFFFF) + (sum >> 16);
37     }
38 }
39 // 如果计算结果为全为1则无差错; 否则, 有差错
40 bool result = sum != 0xFFFF;
41 // 释放动态分配的内存
42 delete[] paddedData;
43 return result;
44 }

```

## 六、 发送文件

如果客户端发送的是带有 FIN 字段的报文, 那么进入挥手模式。

如果发送的是带有 START 字段、内容是文件名的第一个数据包, 那么以二进制方式打开文件 (因为要发送的文件包含图片等非文本文件)。然后将文件拆成若干个 1024Bytes 大小的数据包, 顺序发送出去, 最后一个数据包要加上 END 字段。代码如下:

```

1 void send_file() {
2     Message sendMsg, recvMsg;
3     clock_t start, end;
4     char filePath[20];
5     ifstream in;
6     int filePtrLoc;
7     int dataAmount;
8     int packetNum;
9     int checksum;
10
11     cout << "请输入要发送的文件名: ";
12     memset(filePath, 0, 20);
13     string temp;
14     cin >> temp;
15     string inputPath = "./input/" + temp;
16
17     if (temp == "quit") {
18         closeConnect();
19         quit = true;
20         return;
21     }
22     else if (temp == "1.jpg" || temp == "2.jpg" || temp == "3.jpg" || temp ==
23         "helloworld.txt") {
24         strcpy_s(filePath, sizeof(filePath), temp.c_str());

```

```
24     in.open(inputPath, ifstream::in | ios::binary); // 以读取模式、二进制
        方式打开文件
25     in.seekg(0, ios_base::end); // 将文件流指针移动到文件的末尾
26     dataAmount = in.tellg(); // 文件大小 (以字节为单位)
27     filePtrLoc = dataAmount;
28     packetNum = filePtrLoc / 1024 + 1; // 数据包数量
29     in.seekg(0, ios_base::beg); // 将文件流指针移回文件的开头
30     cout << "文件" << temp << "有" << packetNum << "个数据包" << endl;
31 }
32 else {
33     cout << "文件不存在, 请重新输入您要传输的文件名!" << endl;
34     return;
35 }
36
37 // 发送第一个包, 内容是文件名
38 cout << "客户端发送文件名" << endl;
39 memcpy(sendMsg.data, filePath, strlen(filePath));
40 sendMsg.setSTART();
41 sendMsg.seq = 0;
42 sendMsg.len = strlen(filePath);
43 sendMsg.num = packetNum;
44 sendMsg.setChecksum();
45 cout << "checksum: " << sendMsg.checksum << endl;
46 if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
    serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
47     cout << "客户端发送文件名失败!" << endl;
48     return;
49 }
50
51 start = clock();
52 while (1) {
53     if (recvfrom(socketClient, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
        serverAddr, &len) != SOCKET_ERROR) {
54         if (recvMsg.isACK() && recvMsg.ack == sendMsg.seq + 1 && !recvMsg
            .packetCorruption()) {
55             cout << "客户端发送文件名成功!" << endl;
56             break;
57         }
58     }
59     if (clock() - start > RTO) {
60         cout << "应答超时, 客户端重新发送文件名" << endl;
61         cout << "checksum: " << sendMsg.checksum << endl << endl;
62         if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
            serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
63             cout << "客户端发送文件名失败!" << endl;
64             cout << "当前网络状态不佳, 请稍后再试" << endl;
65             return;
66         }
67     }
68 }
```

```

67         start = clock();
68     }
69 }
70
71 // 开始发送文件内容
72 cout << "客户端开始发送文件内容!" << endl;
73 int seq = 1;
74 start = clock();
75 for (int i = 0; i < packetNum; i++) {
76     if (i == packetNum - 1) {
77         in.read(sendMsg.data, filePtrLoc);
78         sendMsg.seq = seq;
79         sendMsg.len = filePtrLoc;
80         sendMsg.setEND(); // 文件结束标志
81         filePtrLoc = 0;
82     }
83     else {
84         in.read(sendMsg.data, 1024); // 读取文件数据
85         sendMsg.seq = seq;
86         sendMsg.len = 1024;
87         filePtrLoc += 1024;
88     }
89
90     // 发送数据包
91     sendMsg.seq = seq;
92     sendMsg.setChecksum();
93     cout << "seq:" << seq << endl;
94     cout << "len:" << sendMsg.len << endl;
95     cout << "checksum: " << sendMsg.checksum << endl << endl;
96     if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
97         serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
98         cout << "发送数据包失败!" << endl;
99     }
100
101     // 设置套接字为非阻塞模式
102     int mode = 1;
103     ioctlsocket(socketClient, FIONBIO, (u_long FAR*) & mode);
104     int count = 0;
105
106     clock_t c = clock();
107     while (1) {
108         // 尝试接收ack
109         if (recvfrom(socketClient, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)
110             &serverAddr, &len)) {
111             if (recvMsg.isACK() && recvMsg.ack == seq && !recvMsg.
112                 packetCorruption()) {
113                 break;
114             }
115         }
116     }
117 }

```

```

112     }
113
114     // 检查是否超时
115     if (clock() - c > RTO) {
116         cout << "应答超时, 重新发送数据包" << endl;
117         if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (
            SOCKADDR*)&serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR)
            {
118             cout << "发送数据包失败!" << endl;
119         }
120         count++;
121         cout << "尝试重新发送seq为" << seq << "的数据包第" << count
            << "次, 最多5次" << endl;
122         if (count >= 5) {
123             cout << "尝试次数超过5次, 退出发送" << endl;
124             return;
125         }
126         c = clock();
127     }
128     count = 0;
129     // 为了避免CPU占用率过高, 添加延迟
130     Sleep(2);
131 }
132 seq++;
133 }
134 end = clock();
135 cout << "成功发送文件!" << endl;
136
137 double TotalTime = (double)(end - start) / CLOCKS_PER_SEC;
138 cout << "传输总时间:␣" << TotalTime << "s" << endl;
139 cout << "吞吐率:␣" << (double)dataAmount / TotalTime << "␣bytes/s" <<
    endl << endl;
140
141 // 关闭文件并准备发送下一个文件
142 in.close();
143 in.clear();
144 }

```

## 1. 停等机制

客户端发送数据包后, 要进入一个循环等待直至收到服务器端的 ack 响应后, 才能 break 跳出循环, 继续发送下一个数据包。

```

1 // 发送数据包
2 sendMsg.seq = seq;
3 sendMsg.setChecksum();
4 cout << "seq:" << seq << endl;
5 cout << "len:" << sendMsg.len << endl;;

```



```

6  cout << "checksum: " << sendMsg.checksum << endl << endl;
7  if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*
8      cout << "发送数据包失败!" << endl;
9  }
10 // 设置套接字为非阻塞模式
11 int mode = 1;
12 ioctlsocket(socketClient, FIONBIO, (u_long FAR*) & mode);
13 int count = 0;
14 clock_t c = clock();
15 while (1) {
16     // 尝试接收ack
17     if (recvfrom(socketClient, (char*)&recvMsg, BUFFER, 0, (SOC
18         if (recvMsg.isACK() && recvMsg.ack == seq && !recvMsg.p
19         break;
20     }
21 }
22 // 检查是否超时
23 if (clock() - c > RTO) {
24     cout << "应答超时, 重新发送数据包" << endl;
25     if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (S
26         cout << "发送数据包失败!" << endl;
27     }
28     count++;
29     cout << "尝试重新发送seq为" << seq << "的数据包第" << c
30     if (count >= 5) {
31         cout << "尝试次数超过5次, 退出发送" << endl;
32         return;
33     }
34     c = clock();
35 }
36 count = 0;
37 // 为了避免CPU占用率过高, 添加延迟
38 Sleep(2);
39 }

```

## 2. 超时重传

如果客户端在  $2 * \text{CLOCKS\_PER\_SEC}$  的时间内没有收到服务器端回复的 ACK 报文, 就重新发送该数据包。

```

1  const int RTO = 2 * CLOCKS_PER_SEC; // 超时重传时间
2
3  // 检查是否超时
4  if (clock() - c > RTO) {
5      cout << "应答超时, 重新发送数据包" << endl;
6      if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOC
7          cout << "发送数据包失败!" << endl;
8      }

```

```

9     count++;
10    cout << "尝试重新发送seq为" << seq << "的数据包第" << cou
11    if (count >= 5) {
12        cout << "尝试次数超过5次，退出发送" << endl;
13        return;
14    }
15    c = clock();
16 }

```

发送消息后使用 clock() 函数计时，如果时间超过 PTO 还没有收到服务器返回的 ack 响应，就会重新发送消息。此处设置了最大的重发次数为 5 次，重发 5 次后还没有收到服务器返回的 ack 响应，就认为连接已断开，并直接 return，避免消耗网络资源。

## 七、 接收文件

如果服务器端接收到 FIN 报文，说明客户端准备断开连接，进入挥手模式（需要注意的是，接收第一次挥手的消息已经在 recv\_file 函数中处理了，closeConnect 函数只需处理剩下的三次挥手即可）。

如果服务器端接收到 START 报文，说明客户端准备开始发送文件，服务器端以二进制方式打开文件，并开始写入接收到的数据。

```

1 void recv_file() {
2     cout << "服务器正在等待接收文件中....." << endl;
3     Message recvMsg, sendMsg;
4     clock_t start, end;
5     char filePath[20];
6     string outputPath;
7     ofstream out;
8     int dataAmount = 0;
9     int packetNum;
10
11     // 接收文件名
12     while (1) {
13         if (recvfrom(socketServer, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
14             clientAddr, &len) != SOCKET_ERROR) {
15             // 接收第一次挥手信息
16             if (recvMsg.isFIN() && !recvMsg.packetCorruption()) {
17                 cout << "客户端准备断开连接！进入挥手模式！" << endl;
18                 cout << "服务器端接收到第一次挥手消息！第一次挥手成功！" <<
19                     endl;
20                 closeConnect(recvMsg);
21                 quit = true;
22                 return;
23             }
24             if (recvMsg.isSTART() && !recvMsg.packetCorruption()) {
25                 ZeroMemory(filePath, 20);
26                 memcpy(filePath, recvMsg.data, recvMsg.len);
27                 outputPath = "./output/" + string(filePath);

```

```
26         out.open(outputPath, ios::out | ios::binary); //以写入模式、二
           进制模式打开文件
27         cout << "文件名为: " << filePath << endl;
28         cout << "checksum: " << recvMsg.checksum << endl << endl;
29
30         if (!out.is_open()) {
31             cout << "文件打开失败!!! " << endl;
32             exit(1);
33         }
34
35         packetNum = recvMsg.num;
36         cout << "文件" << filePath << "有" << packetNum << "个数据包"
           << endl;
37
38         // 发送ack给客户端
39         sendMsg.setACK();
40         sendMsg.ack = recvMsg.seq + 1;
41         sendMsg.setChecksum();
42         if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (
           SOCKADDR*)&clientAddr, sizeof(SOCKADDR)) == SOCKET_ERROR)
           {
43             cout << "服务器端发送ack报文失败!" << endl;
44             cout << "当前网络状态不佳, 请稍后再试" << endl;
45             return;
46         }
47         break;
48     }
49 }
50
51
52 // 设置套接字为阻塞模式
53 int mode = 0;
54 ioctlsocket(socketServer, FIONBIO, (u_long FAR*) & mode);
55
56 // 开始接收文件内容
57 cout << "服务器端开始接收文件内容!" << endl;
58 int expectedSeq = 1;
59 start = clock();
60 for (int i = 0; i < packetNum; i++) {
61     while (1) {
62         if (recvfrom(socketServer, (char*)&recvMsg, BUFFER, 0, (SOCKADDR
           *)&clientAddr, &len) != SOCKET_ERROR) {
63             // 检查序列号是否正确
64             if (recvMsg.seq == expectedSeq && !recvMsg.packetCorruption()
           ) {
65                 // 以追加模式打开文件, 并写入文件
66                 ofstream out(outputPath, ios::app | std::ios::binary);
67                 out.write(recvMsg.data, recvMsg.len); // 写入数据到文件
```

```

68         dataAmount += recvMsg.len;
69         out.close();
70
71         // 发送ack给客户端
72         cout << "seq:" << recvMsg.seq << endl;
73         cout << "len:" << recvMsg.len << endl;;
74         cout << "checksum: " << recvMsg.checksum << endl << endl;
75         sendMsg.setACK();
76         sendMsg.ack = recvMsg.seq;
77         sendMsg.setChecksum();
78         if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (
79             SOCKADDR*)&clientAddr, sizeof(SOCKADDR)) ==
80             SOCKET_ERROR) {
81             cout << "服务器端发送ack报文失败!" << endl;
82             cout << "当前网络状态不佳, 请稍后再试" << endl;
83             return;
84         }
85         expectedSeq++;
86     }
87     // 检查文件传输是否结束
88     if (recvMsg.isEND() && !recvMsg.packetCorruption()) {
89         end = clock();
90         cout << "接收文件成功!" << endl;
91         out.close();
92         out.clear();
93
94         double TotalTime = (double)(end - start) / CLOCKS_PER_SEC
95         ;
96         cout << "传输总时间" << TotalTime << "s" << endl;
97         cout << "吞吐率" << (double)dataAmount / TotalTime << "
98             bytes/s" << endl << endl;
99
100         return;
101     }
102 }
103 }

```

## 1. 接收确认

当服务器端接受到来自客户端的数据包后, 就发送 ACK 报文给客户端。代码如下:

```

1 // 检查序列号是否正确
2 if (recvMsg.seq == expectedSeq && !recvMsg.packetCorrup
3     // 以追加模式打开文件, 并写入文件

```

```

4   ofstream out(outputPath, ios::app | std::ios::binar
5   out.write(recvMsg.data, recvMsg.len); // 写入数据到
6   dataAmount += recvMsg.len;
7   out.close();
8   // 发送ack给客户端
9   cout << "seq:" << recvMsg.seq << endl;
10  cout << "len:" << recvMsg.len << endl;
11  cout << "checksum: " << recvMsg.checksum << endl <<
12  sendMsg.setACK();
13  sendMsg.ack = recvMsg.seq;
14  sendMsg.setChecksum();
15  if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0
16      cout << "服务器端发送ack报文失败!" << endl;
17      cout << "当前网络状态不佳, 请稍后再试" << endl;
18      return;
19  }
20  expectedSeq++;
21 }

```

## 八、 测试

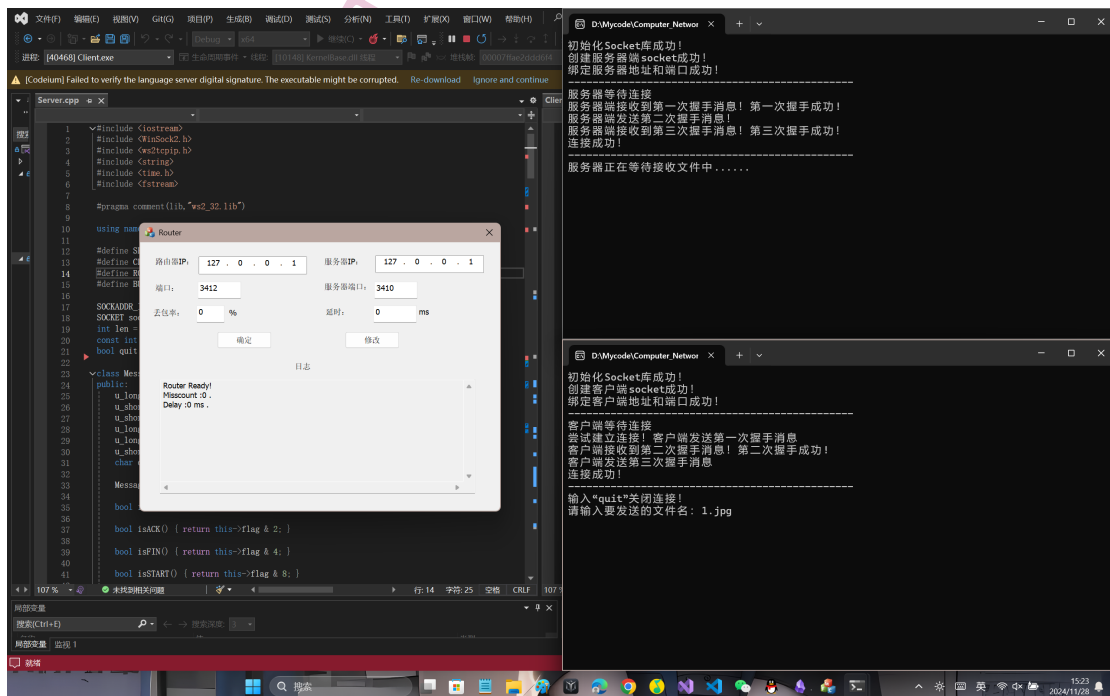
我们分别定义了服务器端、客户端、router 的端口号, IP 统一使用 127.0.0.1。

```

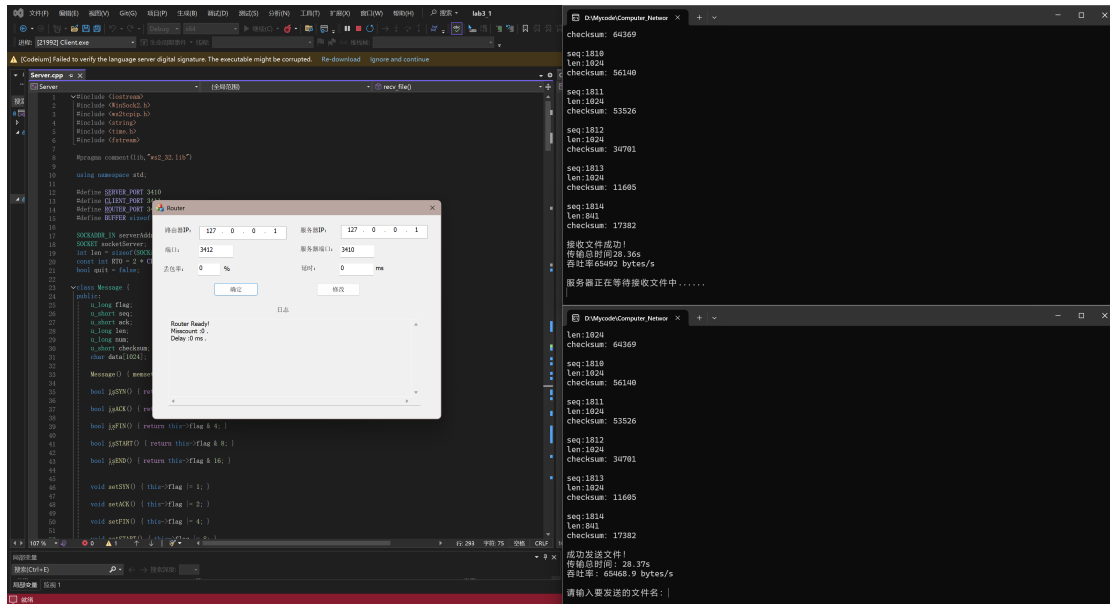
1 #define SERVER_PORT 3410
2 #define CLIENT_PORT 3411
3 #define ROUTER_PORT 3412

```

打开路由程序, 将丢包率和延迟均设为 0%, 开始运行, 首先建立连接

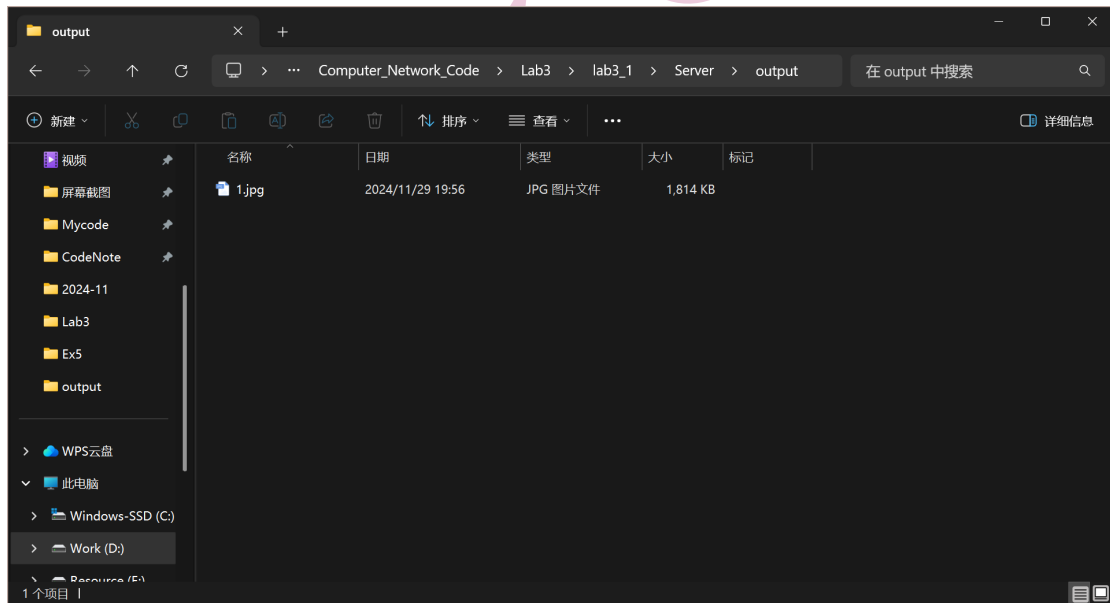


接下来发送文件“1.jpg”



检查 Server

output 目录，出现文件“1.jpg”，且与客户端发送的文件大小相等。



接下来测试超时重传，打开路由程序，将丢包率修改为 5%，开始运行



```

D:\Mycode\Computer_Networ x + v
seq:1802
len:1024
checksum: 1377

seq:1803
len:1024
checksum: 7799

seq:1804
len:1024
checksum: 14441

seq:1805
len:1024
checksum: 43476

seq:1806
len:1024
checksum: 37279

seq:1807
len:1024
checksum: 28978

seq:1808
len:1024
checksum: 24037

D:\Mycode\Computer_Networ x + v
checksum: 52502

seq:1802
len:1024
checksum: 1377

seq:1803
len:1024
checksum: 7799

seq:1804
len:1024
checksum: 14441

seq:1805
len:1024
checksum: 43476

应答超时, 重新发送数据包
尝试重新发送 seq 为 1805 的数据包第 1 次, 最多 5 次
seq:1806
len:1024
checksum: 37279

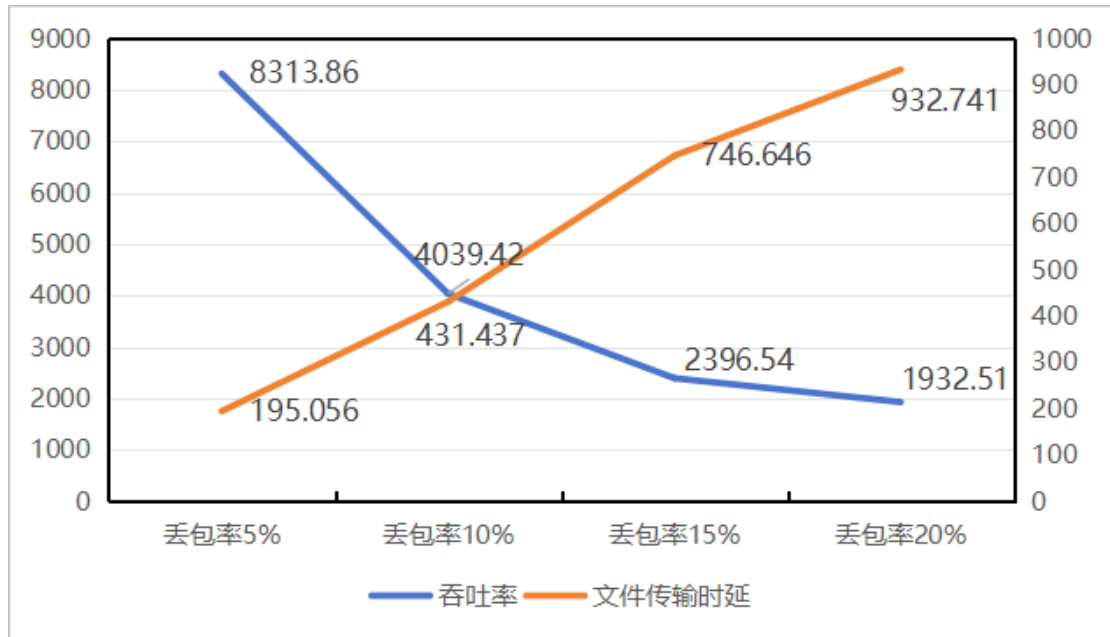
seq:1807
len:1024
checksum: 28978

```

我们以丢包率 0%、延时 0% 为基准，分别测试丢包率为 5%、10%、15%、20% 情况下，传输文件“1.jpg”的吞吐率和文件传输时延（即传输时间减去没有丢包时的传输时间），并绘制表格和图形结果。

	丢包率 5%	丢包率 10%	丢包率 15%	丢包率 20%
吞吐率 (Bytes/s)	8313.86	4039.42	2396.54	1932.51
文件传输时延 (s)	195.056	431.437	746.646	932.741





可以看到，随着丢包率的增加，文件传输时延大幅提升，基本与丢包率成正比；吞吐量大幅下降，基本与丢包率成反比。

断开连接：

