



南開大學
Nankai University

南 开 大 学

计 算 机 学 院

计算机网络实验报告

Lab 3.2 基于 UDP 服务设计可靠传输协议并编程实现

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 10 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. 流水线协议	13
2. 滑动窗口	14
3. 超时重传	17
七、 接收文件	18
1. 累积确认	21
八、 测试	24

一、 实验目的

在实验 3-1 的基础上，将停等机制改成基于滑动窗口的流量控制机制，采用固定窗口大小，支持累积确认，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。

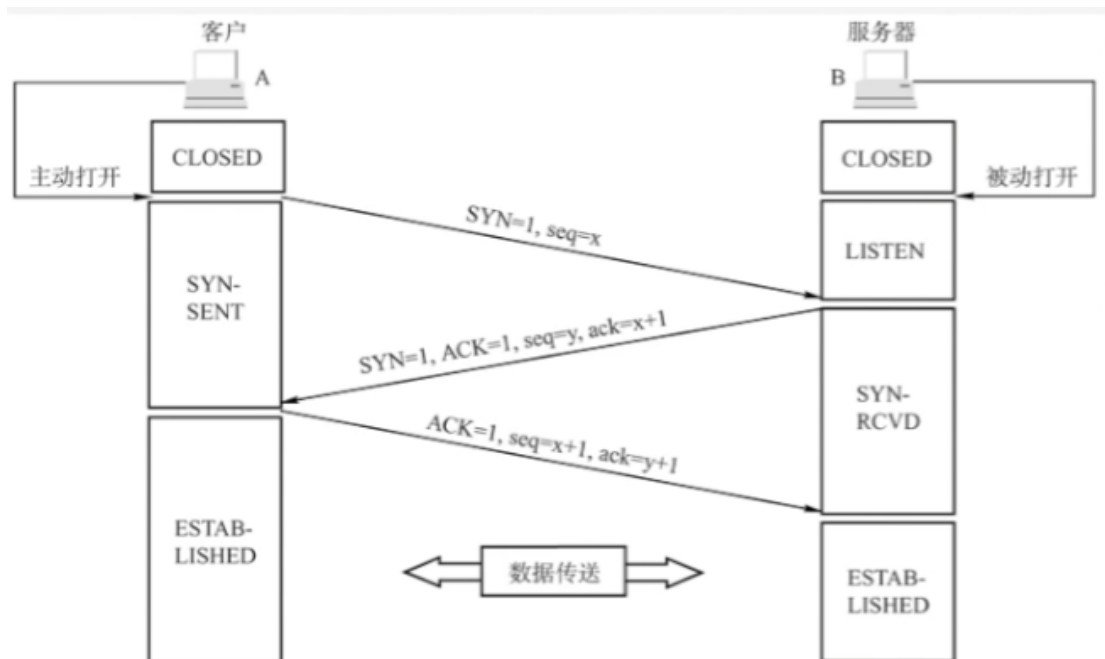
```
1 void setSYN() { this->flag |= 1; }
2 void setACK() { this->flag |= 2; }
3 void setFIN() { this->flag |= 4; }
4 void setSTART() { this->flag |= 8; }
5 void setEND() { this->flag |= 16; }
```

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 packetIncorruption。

```
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 packetIncorruption() {
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.packetIncorruption()) {
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.
            packetIncorruption()) {
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;
```

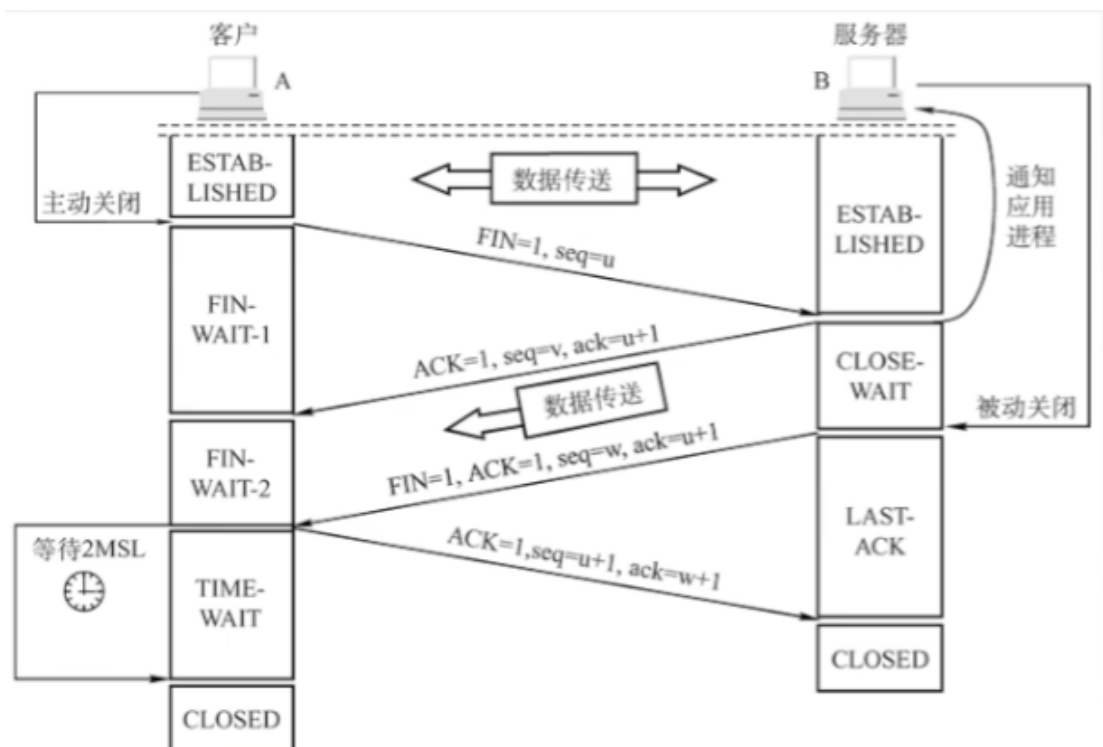
```
58 }
59
60 //Client.cpp
61 bool waitConnect() {
62     Message sendMsg, recvMsg;
63     clock_t start;
64
65     // 设置套接字为非阻塞模式
66     int mode = 1;
67     ioctlsocket(socketClient, FIONBIO, (u_long FAR*) & mode);
68
69     // 发送第一次握手消息
70     cout << "尝试建立连接! 客户端发送第一次握手消息" << endl;
71     sendMsg.setSYN();
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.packetIncorruption()) {
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                .packetIncorruption()) {
```

```

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                 .packetIncorruption()) {
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             .packetIncorruption()) {
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;
132 }
```

```

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 packetIncorruption 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 packetIncorruption() {
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

客户端首先需要检测用户输入, 如果用户输入的是“quit”, 说明用户要关闭连接, 那么进入挥手模式。

如果用户输入的是文件名, 我们就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)将文件拆成若干个 1024Bytes 大小的数据包。

我们要发送的第一个数据包内容是文件名, 发送成功后正式开始发送文件内容。

为了实现并行的发送和接收, 我们在 `send_file` 函数中创建两个线程, 分别是发送线程 `send_thread` 和接收线程 `recv_thread`。

```

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

```

```

65         cout << "checksum: " << sendMsg.checksum << endl << endl;
66         if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
            routerAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
67             cout << "客户端发送文件名失败!" << endl;
68             cout << "当前网络状态不佳, 请稍后再试" << endl;
69             return;
70         }
71         start = clock();
72     }
73 }
74
75 // 开始发送文件内容
76 cout << "客户端开始发送文件内容!" << endl << endl;
77 start = clock();
78
79 // 创建发送和接收线程
80 thread sender(send_thread, socketClient, ref(routerAddr), ref(in),
            filePtrLoc, packetNum);
81 thread receiver(recv_thread, socketClient, packetNum);
82
83 sender.join(); // 等待发送线程完成
84 receiver.join(); // 等待接收线程完成
85
86 end = clock();
87 cout << "成功发送文件!" << endl;
88
89 double TotalTime = (double)(end - start) / CLOCKS_PER_SEC;
90 cout << "传输总时间:_" << TotalTime << "s" << endl;
91 cout << "吞吐率:_" << (double)dataAmount / TotalTime << "_bytes/s" <<
    endl << endl;
92
93 // 关闭文件并准备发送下一个文件
94 in.close();
95 in.clear();
96
97 base = 1; // 窗口的起始序列号
98 next_seq = 1; // 下一个待发送的序列号
99 send_time = 0; // 报文发送起始时间
100 send_over = false; // 传输完毕
101 }

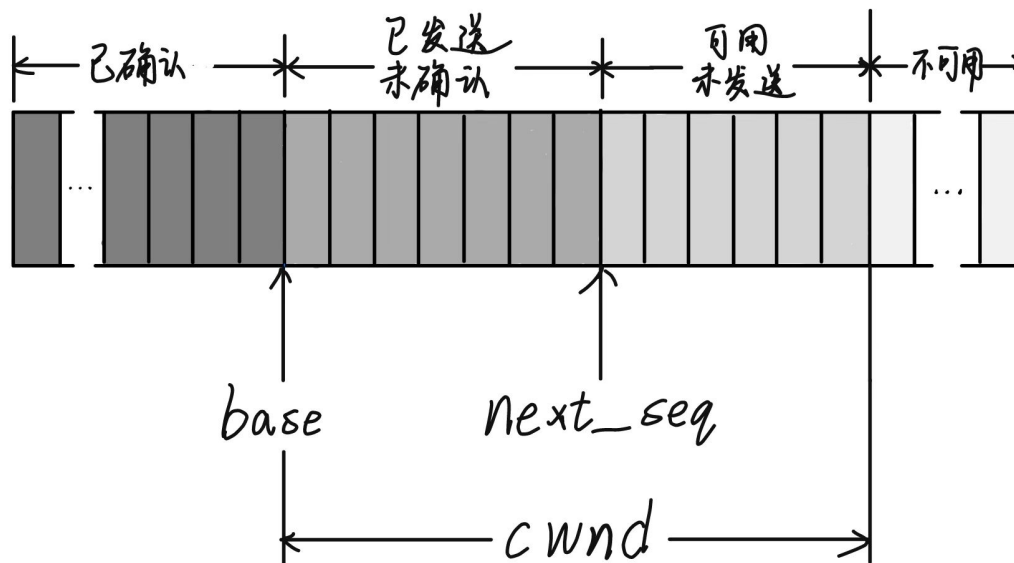
```

1. 流水线协议

停等机制在客户端发送数据包时, 必须等到服务器端返回 ack, 才可以继续发送下一个数据包, 这种机制效率低下, 会产生很长的等待时延。

流水线协议指客户端可以连续发送多个数据包, 在这期间无需等到服务器端的 ack, 从而提高了传输效率。

2. 滑动窗口



本次实验将流量控制机制从 3-1 中的停等机制改成了滑动窗口，假设滑动窗口大小为 $cwnd = 5$ ，那么客户端最多可以连续发送 5 个数据包，而在这期间无需等到服务器端的 ack 。

客户端每发送一个数据包， seq_next 自增 1，直到 seq_next 到达窗口右边界 $base + cwnd$ 。

客户端收到服务器端发送的 ack 报文后，会移除 ack 包含的 seq 之前的接收缓冲区内容以及计时器，将 $base$ 赋值为 ack 报文的内容。

```

1  const int cwnd = 5; // 窗口大小
2  int base = 1; // 窗口的起始序列号
3  int next_seq = 1; // 下一个待发送的序列号
4  mutex seq_mutex; // 序列号的互斥锁
5  bool send_over = false; // 传输完毕
6  map<int, Message> send_buffer; // 序列号及其报文的映射
7  map<int, clock_t> send_times; // 序列号及其发送时间的映射
8
9
10
11 // 发送线程
12 void send_thread(SOCKET socketClient, sockaddr_in& routerAddr, ifstream& in,
13     int filePtrLoc, int packetNum) {
14     Message sendMsg;
15     int count = 0;
16
17     // 窗口内发送数据
18     while (!send_over) {
19         {
20             unique_lock<mutex> lock(seq_mutex); // 加锁
21             while (next_seq <= packetNum && (next_seq - base) < cwnd) {
22                 if (next_seq == packetNum) {

```



```

22         in.read(sendMsg.data, filePtrLoc);
23         sendMsg.len = filePtrLoc;
24         sendMsg.setEND(); // 文件结束标志
25         filePtrLoc = 0;
26     }
27     else {
28         in.read(sendMsg.data, 1024); // 读取文件数据
29         sendMsg.len = 1024;
30         filePtrLoc -= 1024;
31     }
32
33     // 发送数据包
34     sendMsg.seq = next_seq;
35     sendMsg.setChecksum();
36     cout << "发送seq:" << next_seq << endl;
37     cout << "base: " << base << endl;
38     cout << "next_seq: " << next_seq << endl;
39     cout << "len:" << sendMsg.len << endl;
40     cout << "checksum: " << sendMsg.checksum << endl << endl;
41
42     if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (
43         SOCKADDR*)&routerAddr, sizeof(SOCKADDR)) == SOCKET_ERROR)
44     {
45         cout << "发送数据包失败!" << endl;
46     }
47
48     // 记录发送时间
49     send_buffer[next_seq] = sendMsg;
50     send_times[next_seq] = clock();
51
52     next_seq++;
53 }
54 lock.unlock(); // 解锁
55 {
56     unique_lock<mutex> lock(seq_mutex); // 加锁
57     // 超时重传
58     for (auto it = send_buffer.begin(); it != send_buffer.end(); ) {
59         if (clock() - send_times[it->first] > TIMEOUT) {
60             cout << "应答超时, 重新发送已发送未确认的数据包" << endl;
61
62             Message sendMsg = it->second;
63             cout << "发送seq:" << it->first << endl;
64             cout << "base: " << base << endl;
65             cout << "next_seq: " << next_seq << endl;
66             cout << "len:" << sendMsg.len << endl;
67             cout << "checksum: " << sendMsg.checksum << endl << endl;

```

```

68
69         if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (
           SOCKADDR*)&routerAddr, sizeof(SOCKADDR)) ==
           SOCKET_ERROR) {
70             cout << "重传数据包失败!" << endl;
71         }
72         // 重置计时器
73         send_times[it->first] = clock();
74     }
75     it++;
76 }
77 lock.unlock(); // 解锁
78 }
79 }
80 }
81
82 // 接收线程
83 void recv_thread(SOCKET socketClient, int packetNum) {
84     Message recvMsg;
85     clock_t start;
86
87     while (!send_over) {
88         if (recvfrom(socketClient, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
           routerAddr, &len) != SOCKET_ERROR) {
89             unique_lock<mutex> lock(seq_mutex); // 加锁
90             if (recvMsg.isACK() && !recvMsg.packetIncorrection()) {
91                 cout << "接收ack: " << recvMsg.ack << endl;
92                 cout << "base: " << base << endl;
93
94                 // 移除已确认的数据包
95                 auto it = send_buffer.begin();
96                 while (it != send_buffer.end() && it->first < recvMsg.ack) {
97                     it = send_buffer.erase(it); // 使用 erase 的返回值更新迭
                       代器
98                     send_times.erase(recvMsg.ack); // 同时移除计时器
99                 }
100
101                 base = recvMsg.ack;
102
103
104                 // 展示窗口情况
105                 cout << "base=接收ack=" << base << endl;
106                 cout << "next_seq: " << next_seq << endl;
107                 cout << "窗口内已发送但未收到ack的包: " << next_seq - base <<
                       endl;
108                 cout << "窗口内未发送的包: " << cwnd - (next_seq - base) <<
                       endl << endl;
109                 // 如果所有文件传输结束

```

```

110         if (recvMsg.ack == packetNum + 1) {
111             cout << "文件传输完成!" << endl;
112             send_over = true;
113         }
114     }
115     lock.unlock(); // 解锁
116 }
117 }
118 }

```

3. 超时重传

给每个数据包绑定一个计时器，接收线程收到 ack 后，会移除 ack 包含的 seq 之前的接收缓冲区内容以及计时器。

```

1 // 移除已确认的数据包
2 auto it = send_buffer.begin();
3 while (it != send_buffer.end() && it->first < recvMsg.ack) {
4     it = send_buffer.erase(it); // 使用 erase 的返回值更新迭代器
5     send_times.erase(recvMsg.ack); // 同时移除计时器
6 }

```

没有移除计时器的数据包会在 TIMEOUT 的时间后进行超时重传，就重新发送窗口内已发送未确认的数据包。

```

1 {
2     unique_lock<mutex> lock(seq_mutex); // 加锁
3     // 超时重传
4     for (auto it = send_buffer.begin(); it != send_buffer.end(); ) {
5         if (clock() - send_times[it->first] > TIMEOUT) {
6             cout << "应答超时，重新发送已发送未确认的数据包" << endl;
7             Message sendMsg = it->second;
8             cout << "发送seq:" << it->first << endl;
9             cout << "base: " << base << endl;
10            cout << "next_seq: " << next_seq << endl;
11            cout << "len:" << sendMsg.len << endl;
12            cout << "checksum: " << sendMsg.checksum << endl << endl;
13            if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
14                route
15                cout << "重传数据包失败!" << endl;
16            }
17            // 重置计时器
18            send_times[it->first] = clock();
19        }
20        it++;
21    }
22    lock.unlock(); // 解锁
}

```

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             routerAddr, &len) != SOCKET_ERROR) {
15             // 接收第一次挥手信息
16             if (recvMsg.isFIN() && !recvMsg.packetIncorrection()) {
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.packetIncorrection()) {
25                 ZeroMemory(filePath, 20);
26                 memcpy(filePath, recvMsg.data, recvMsg.len);
27                 outputPath = "./output/" + string(filePath);
28                 out.open(outputPath, ios::out | ios::binary); //以写入模式、二
29                     进制模式打开文件
30
31                 cout << "文件名为：" << filePath << endl;
32                 cout << "接收到的seq：" << recvMsg.seq << endl;
33                 cout << "checksum：" << recvMsg.checksum << endl;
34
35                 if (!out.is_open()) {
36                     cout << "文件打开失败！！!" << endl;
37                     exit(1);
38                 }
39
40                 packetNum = recvMsg.num;
41                 cout << "文件" << filePath << "有" << packetNum << "个数据包"

```

```

38         << endl;
39         // 发送ack给客户端
40         sendMsg.setACK();
41         sendMsg.ack = recvMsg.seq + 1;
42         sendMsg.setChecksum();
43         if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (
            SOCKADDR*)&routerAddr, sizeof(SOCKADDR)) == SOCKET_ERROR)
            {
44             cout << "服务器端发送ack报文失败!" << endl;
45             cout << "当前网络状态不佳, 请稍后再试" << endl;
46             return;
47         }
48         break;
49     }
50 }
51 }
52
53 // 设置套接字为阻塞模式
54 int mode = 0;
55 ioctlsocket(socketServer, FIONBIO, (u_long FAR*) & mode);
56
57 // 开始接收文件内容
58 cout << "服务器端开始接收文件内容!" << endl << endl;
59 int expected_seq = 1;
60 bool first_recv_incorrect_seq = true;
61 start = clock();
62 while (1) {
63     if (recvfrom(socketServer, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
        routerAddr, &len) != SOCKET_ERROR) {
64         // 如果校验和错误
65         if (recvMsg.packetIncorection()) {
66             cout << "checksum错误!" << endl;
67             cout << "checksum: " << recvMsg.checksum << endl;
68             continue;
69         }
70         if (recvMsg.seq < expected_seq) { // 去重
71             continue;
72         }
73         else {
74             // 如果接收到的是期望接收到的seq
75             if (recvMsg.seq == expected_seq) {
76                 first_recv_incorrect_seq = true;
77                 expected_seq++;
78             }
79             // 如果接收到的不是期望接收到的seq
80             else {
81                 if (first_recv_incorrect_seq) {

```

```

82 // 发送累计确认的ack给客户端
83 cout << "期望接收seq:" << expected_seq << endl;
84
85 sendMsg.setACK();
86 sendMsg.ack = expected_seq;
87 sendMsg.setChecksum();
88 cout << "发送累计确认ack:" << sendMsg.ack << endl <<
    endl;
89 if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0,
    (SOCKADDR*)&routerAddr, sizeof(SOCKADDR)) ==
    SOCKET_ERROR) {
90     cout << "服务器端发送ack报文失败!" << endl;
91     cout << "当前网络状态不佳, 请稍后再试" << endl;
92     return;
93 }
94 }
95 first_rcv_incorrect_seq = false;
96 }
97 // 存储接收到的数据包
98 rcv_buffer[rcvMsg.seq] = rcvMsg;
99 cout << "接收seq:" << rcvMsg.seq << endl;
100 cout << "len:" << rcvMsg.len << endl;
101 cout << "checksum: " << rcvMsg.checksum << endl << endl;
102 }
103
104 // 判断接收缓冲区的seq是否按序排列且最后一个seq的标志位是isEND
105 bool isSeqInOrder = true;
106 int prevSeq = 0;
107 for (auto it = rcv_buffer.begin(); it != rcv_buffer.end(); ++it) {
108     if (prevSeq != 0 && it->first != prevSeq + 1) {
109         isSeqInOrder = false;
110         break;
111     }
112     prevSeq = it->first;
113 }
114
115 // 如果接收缓冲区满了或者接收到最后一组数据包
116 if (isSeqInOrder && (rcv_buffer.size() >= cwnd || rcv_buffer.
    rbegin()->first == packetNum)) {
117     cout << "接收缓冲区写入文件" << endl << endl;
118     // 以追加模式打开文件, 并写入文件
119     ofstream out(outputPath, ios::app | std::ios::binary);
120     // 遍历接收缓冲区, 按序列号从小到大顺序将数据包写入文件
121     for (auto it = rcv_buffer.begin(); it != rcv_buffer.end();
        ++it) {
122         const Message& bufferRcvMsg = it->second;
123         out.write(bufferRcvMsg.data, bufferRcvMsg.len); // 写入

```

```

                                数据到文件
124         dataAmount += bufferRecvMsg.len;
125     }
126     out.close();
127
128     // 发送ack给客户端
129     sendMsg.setACK();
130     sendMsg.ack = recv_buffer.rbegin()->first + 1;
131     sendMsg.setChecksum();
132     cout << "发送ack:" << sendMsg.ack << endl << endl;
133     if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (
        SOCKADDR*)&routerAddr, sizeof(SOCKADDR)) == SOCKET_ERROR)
    {
134         cout << "服务器端发送ack报文失败!" << endl;
135         cout << "当前网络状态不佳, 请稍后再试" << endl;
136         return;
137     }
138
139     // 检查文件传输是否结束
140     if (recv_buffer.rbegin()->first == packetNum) {
141         end = clock();
142         cout << "接收文件成功!" << endl;
143         out.close();
144         out.clear();
145
146         double TotalTime = (double)(end - start) / CLOCKS_PER_SEC
            ;
147         cout << "传输总时间" << TotalTime << "s" << endl;
148         cout << "吞吐量" << (double)dataAmount / TotalTime << "
            bytes/s" << endl << endl;
149
150         return;
151     }
152     recv_buffer.clear(); // 清空接收缓冲区
153
154     expected_seq = sendMsg.ack;
155 }
156
157 }
158 }
159 }

```

1. 累积确认

累积确认指服务器端发送的 ack 报文中包含了一个序号, 表示前面的数据包已经成功接收, 并准备好了下一个期望收到的数据包的序号。

客户端发送一组数据包, 服务器端接收后先检查 seq, 如果在之前接收过就丢弃。如果接收

到的是期望接收到的 seq, 就让期望接收的 seq 自增 1; 如果接收到的不是期望接收到的 seq, 就将期望接收到的 seq 作为 ack 报文的内容发送给客户端。

将接收到的数据包全部存到接收缓冲区。判断接收缓冲区的 seq 是否有序, 如果有序, 且接收缓冲区已满, 或者接收到最后一组数据包 (长度可能不足窗口大小), 则尝试写入文件, 并发送 ack 给客户端。

```

1  if (recvfrom(socketServer, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&routerAddr
    , &le
2      // 如果校验和错误
3      if (recvMsg.packetIncorection()) {
4          cout << "checksum 错误! " << endl;
5          cout << "checksum: " << recvMsg.checksum << endl;
6          continue;
7      }
8  if (recvMsg.seq < expected_seq) { // 去重
9      continue;
10 }
11 else {
12     // 如果接收到的是期望接收到的seq
13     if (recvMsg.seq == expected_seq) {
14         first_recv_incorrect_seq = true;
15         expected_seq++;
16     }
17     // 如果接收到的不是期望接收到的seq
18     else{
19         if (first_recv_incorrect_seq) {
20             // 发送累计确认的ack给客户端
21             cout << "期望接收seq:" << expected_seq << endl;
22             sendMsg.setACK();
23             sendMsg.ack = expected_seq;
24             sendMsg.setChecksum();
25             cout << "发送累计确认ack:" << sendMsg.ack << endl << endl;
26             if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
                r
27                 cout << "服务器端发送ack报文失败!" << endl;
28                 cout << "当前网络状态不佳, 请稍后再试" << endl;
29                 return;
30             }
31         }
32         first_recv_incorrect_seq = false;
33     }
34     // 存储接收到的数据包
35     recv_buffer[recvMsg.seq] = recvMsg;
36     cout << "接收seq:" << recvMsg.seq << endl;
37     cout << "len:" << recvMsg.len << endl;
38     cout << "checksum: " << recvMsg.checksum << endl << endl;
39 }
40 // 判断接收缓冲区的seq是否按序排列且最后一个seq的标志位是isEND

```



```

41 bool isSeqInOrder = true;
42 int prevSeq = 0;
43 for (auto it = recv_buffer.begin(); it != recv_buffer.end(); ++it) {
44     if (prevSeq != 0 && it->first != prevSeq + 1) {
45         isSeqInOrder = false;
46         break;
47     }
48     prevSeq = it->first;
49 }
50 // 如果接收缓冲区满了或者接收到最后一组数据包
51 if (isSeqInOrder && (recv_buffer.size() >= cwnd || recv_buffer.rbegin()->
    first
52     cout << "接收缓冲区写入文件" << endl << endl;
53     // 以追加模式打开文件，并写入文件
54     ofstream out(outputPath, ios::app | std::ios::binary);
55     // 遍历接收缓冲区，按序列号从小到大顺序将数据包写入文件
56     for (auto it = recv_buffer.begin(); it != recv_buffer.end(); ++it) {
57         const Message& bufferRecvMsg = it->second;
58         out.write(bufferRecvMsg.data, bufferRecvMsg.len); // 写入数据到文件
59         dataAmount += bufferRecvMsg.len;
60     }
61     out.close();
62     // 发送ack给客户端
63     sendMsg.setACK();
64     sendMsg.ack = recv_buffer.rbegin()->first + 1;
65     sendMsg.setChecksum();
66     cout << "发送ack:" << sendMsg.ack << endl << endl;
67     if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
        routerAddr
68         cout << "服务器端发送ack报文失败!" << endl;
69         cout << "当前网络状态不佳，请稍后再试" << endl;
70         return;
71     }
72     // 检查文件传输是否结束
73     if (recv_buffer.rbegin()->first == packetNum) {
74         end = clock();
75         cout << "接收文件成功!" << endl;
76         out.close();
77         out.clear();
78         double TotalTime = (double)(end - start) / CLOCKS_PER_SEC;
79         cout << "传输总时间" << TotalTime << "s" << endl;
80         cout << "吞吐率" << (double)dataAmount / TotalTime << "bytes/s" <<
            en
81         return;
82     }
83     recv_buffer.clear(); // 清空接收缓冲区
84     expected_seq = sendMsg.ack;
85 }

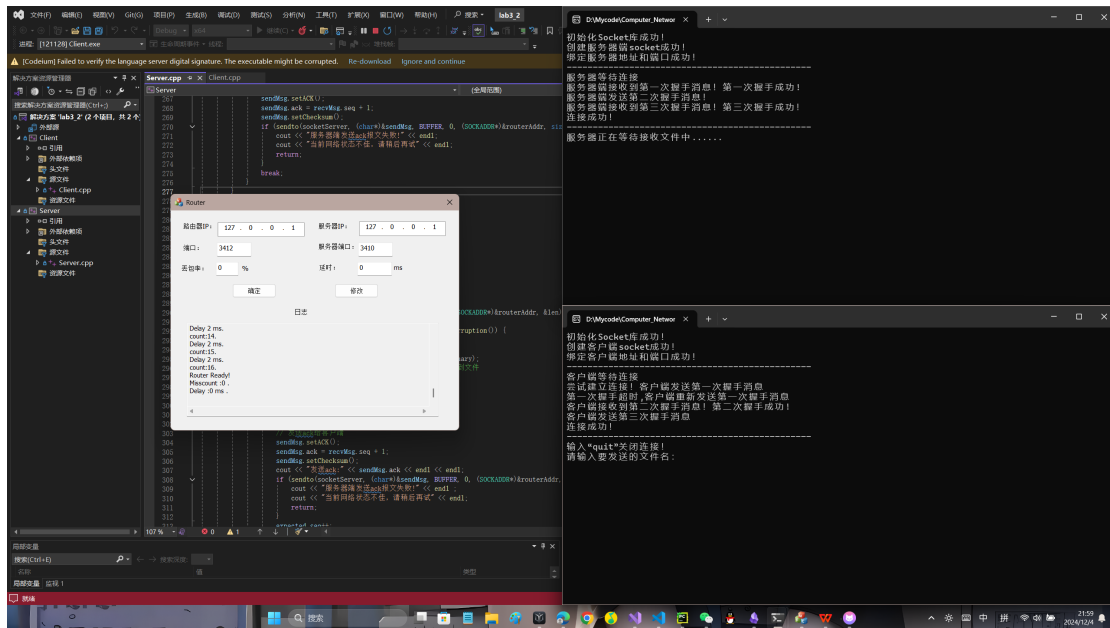
```

八、 测试

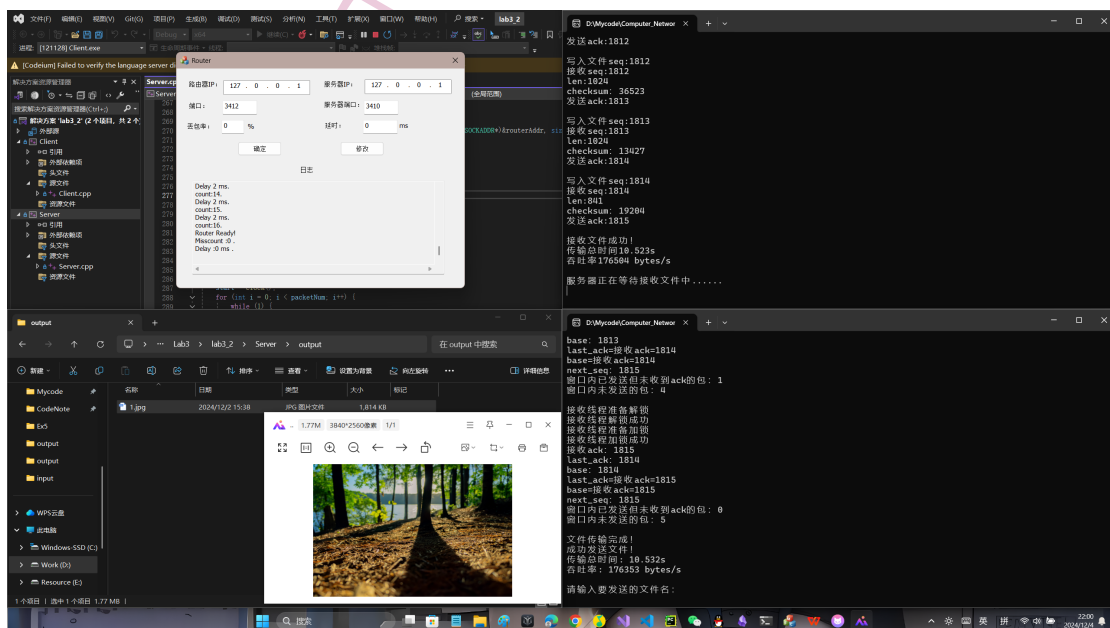
我们分别定义了服务器端、客户端、router 的端口号，IP 统一使用 127.0.0.1。

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

打开路由程序，设置 0% 的丢包率和 0ms 延迟，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。



接下来测试滑动窗口的超时重传，打开路由程序，设置 3% 的丢包率和 2ms 延迟，开始运行。

流水线发送

```

D:\Mycode\Computer_Network >
初始化Socket库成功！
创建服务端socket成功！
绑定服务端地址和端口成功！

服务端等待连接
服务端接收到第一次握手消息！ 第一次握手成功！
服务端发送第二次握手消息！
服务端接收到第二次握手消息！ 第二次握手成功！
连接成功！

服务端正在等待接收文件中.....
文件名为: 1.jpg
接收到的 seq:8
checksum: 23802
文件1.jpg有1814个数据包
服务端开始接收文件内容！

接收 seq:1
len:1024
checksum: 42749

接收 seq:2
len:1024
checksum: 4037

接收 seq:3
len:1024
checksum: 55587

接收 seq:4
len:1024
checksum: 20859

接收 seq:5
len:1024
checksum: 1678

接收缓冲区写入文件
发送 ack:6
接收 seq:6
len:1024
checksum: 46566

接收 seq:7
len:1024
checksum: 33968

接收 seq:8
len:1024
checksum: 45406

接收 seq:9
len:1024
checksum: 28885

D:\Mycode\Computer_Network >
初始化Socket库成功！
创建客户端socket成功！
绑定客户端地址和端口成功！

客户端等待连接
去连接立连接！ 客户端发送第一次握手消息
客户端接收到第二次握手消息！ 第二次握手成功！
客户端发送第三次握手消息
连接成功！

输入"quit"关闭连接！
请输入要发送的文件名: 1.jpg
文件1.jpg有1814个数据包
客户端发送文件名
发送 seq:0
base: 1
next_seq: 1
len:5
checksum: 23802
客户端发送文件名成功！
客户端开始发送文件内容！

发送 seq:1
base: 1
next_seq: 1
len:1024
checksum: 42749

发送 seq:2
base: 1
next_seq: 2
len:1024
checksum: 4037

发送 seq:3
base: 1
next_seq: 3
len:1024
checksum: 55587

发送 seq:4
base: 1
next_seq: 4
len:1024
checksum: 20859

发送 seq:5
base: 1
next_seq: 5
len:1024
checksum: 1678

接收 ack: 6
base: 1
base=接收ack=6
next_seq: 6

```

超时重传。客户端发送的过程中发生丢包，seq 为 1782，客户端会尝试重新发送窗口内已发送未确认的数据包。

```

D:\Mycode\Computer_Network >
len:1024
checksum: 34934

接收 seq:1777
len:1024
checksum: 34126

接收 seq:1778
len:1024
checksum: 45183

接收缓冲区写入文件
发送 ack:1779
接收 seq:1779
len:1024
checksum: 15734

接收 seq:1780
len:1024
checksum: 53467

接收 seq:1781
len:1024
checksum: 46791

删除接收 seq:1782
发送 seq:1783
len:1024
checksum: 34837

接收 seq:1782
len:1024
checksum: 51119

接收缓冲区写入文件
发送 ack:1784
接收 seq:1784
len:1024
checksum: 115

接收 seq:1785
len:1024
checksum: 49243

接收 seq:1786
len:1024
checksum: 18336

接收 seq:1787
len:1024

D:\Mycode\Computer_Network >
发送 seq:1779
base: 1779
next_seq: 1779
len:1024
checksum: 15734

发送 seq:1780
base: 1779
next_seq: 1780
len:1024
checksum: 53467

发送 seq:1781
base: 1779
next_seq: 1781
len:1024
checksum: 46791

发送 seq:1782
base: 1779
next_seq: 1782
len:1024
checksum: 51119

发送 seq:1783
base: 1779
next_seq: 1783
len:1024
checksum: 34837

接收 ack: 1782
base: 1779
base=接收ack=1782
next_seq: 1784
窗口内已发送但未收到ack的包: 2
窗口内未发送的包: 3

超时时，重新发送已发送未确认的数据包
发送 seq:1782
base: 1782
next_seq: 1784
len:1024
checksum: 51119

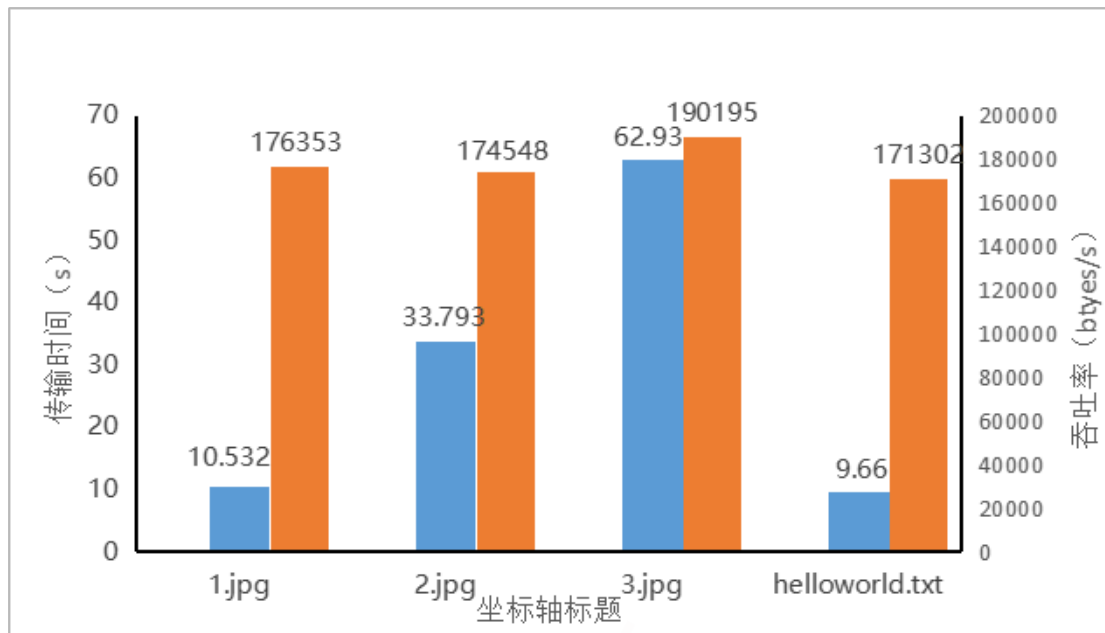
发送 seq:1784
base: 1782
next_seq: 1784
len:1024
checksum: 115

发送 seq:1785
base: 1782
next_seq: 1785
len:1024
checksum: 49243

```

累计确认。服务器端返回的 ack 报文中包含的 seq 为 1782，表示 seq 为 1782 前面的数据包已经成功接收，并准备好了下一个期望收到的数据包的 seq 为 1782。

固定窗口大小为 5，设置 0% 的丢包率和 0ms 延迟，分析不同文件的传输时间和吞吐率。



文件名	1.jpg	2.jpg	3.jpg	helloworld.txt
传输时间 (s)	10.532	33.793	62.93	9.66
吞吐率 (bytes/s)	176353	174548	190195	171302

以“1.jpg”为例，设置 0% 的丢包率和 0ms 延迟，分析不同窗口大小的传输时间和吞吐率。
断开连接：

