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 3.2 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

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

```
class Message {
public:
    u_long flag;          // 伪首部
    u_short seq;          // 序列号
    u_short ack;          // 确认号
    u_long len;           // 数据部分长度
    u_long num;           // 数据包个数
    u_short checksum;     // 校验和
    char data[1024];      // 数据

    Message() { memset(this, 0, sizeof(Message)); }

    bool isSYN() { return this->flag & 1; }

    bool isACK() { return this->flag & 2; }

    bool isFIN() { return this->flag & 4; }

    bool isSTART() { return this->flag & 8; }

    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。

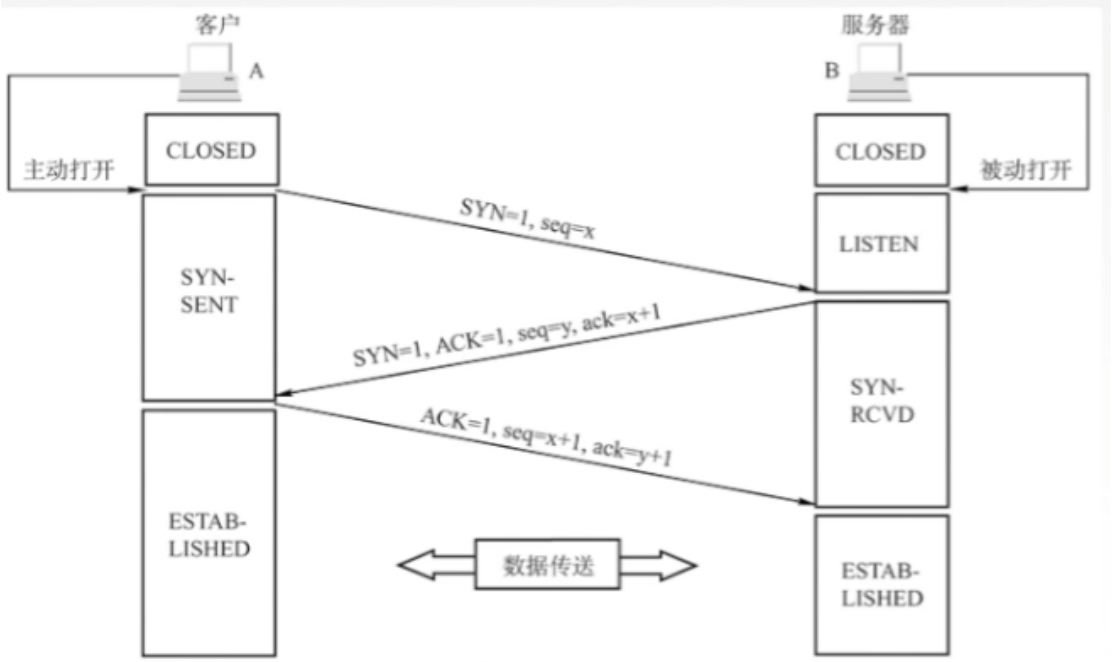
```
void setSYN() { this->flag |= 1; }
void setACK() { this->flag |= 2; }
void setFIN() { this->flag |= 4; }
void setSTART() { this->flag |= 8; }
void setEND() { this->flag |= 16; }
```

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。

```
    void setChecksum() {
        int sum = 0;
        u_char* temp = (u_char*)this;
        for (int i = 0; i < 8; i++) {
            sum += (temp[i << 1] << 8) + temp[i << 1 | 1];
            while (sum >= 0x10000) {
                // 溢出
                int t = sum >> 16; // 将最高位回滚添加至最低位
                sum += t;
            }
        }
        this->checksum = ~(u_short)sum; // 按位取反，方便校验计算
    }

    bool packetIncorruption() {
        int sum = 0;
        u_char* temp = (u_char*)this;
        for (int i = 0; i < 8; i++) {
            sum += (temp[i << 1] << 8) + temp[i << 1 | 1];
            while (sum >= 0x10000) {
                // 溢出
                int t = sum >> 16; // 计算方法与设置校验和相同
                sum += t;
            }
        }
        // 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
        if (checksum + (u_short)sum == 65535)
            return false;
        return true;
    }
```

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:

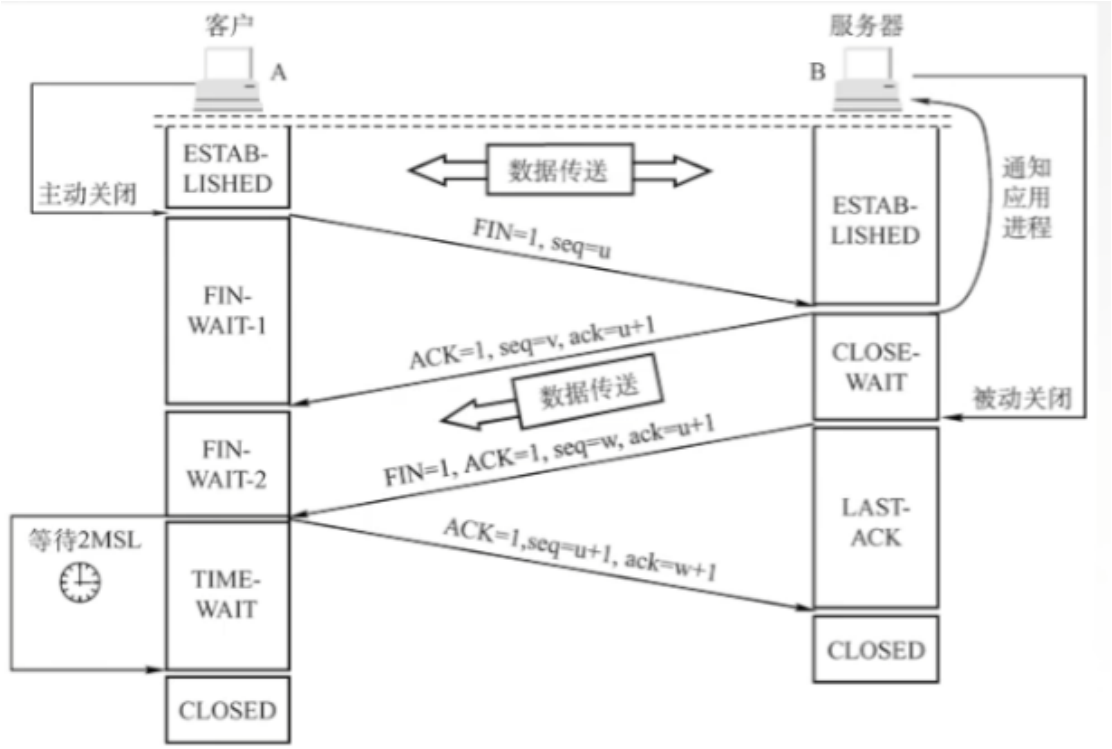
```
//Server.cpp
bool waitConnect() {
    // 设置套接字为非阻塞模式
    int mode = 1;
    ioctlsocket(socketServer, FIONBIO, (u_long FAR*) & mode);

    Message sendMsg, recvMsg;
    clock_t start;

    // 接收第一次握手消息
    while (1) {
        if (recvfrom(socketServer, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
            clientAddr, &len) != SOCKET_ERROR) {
            if (recvMsg.isSYN() && !recvMsg.packetIncorruption()) {
                cout << "服务器端接收到第一次握手消息! 第一次握手成功!" <<
                    endl;
```

```
                break;
            }
        }
    }

    // 发送第二次握手消息
    cout << "服务器端发送第二次握手消息! " << endl;
    sendMsg.setSYN();
    sendMsg.setACK();
    sendMsg.seq = 1000;
    sendMsg.ack = recvMsg.seq + 1;
    sendMsg.setChecksum();
    if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
        clientAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
        cout << "服务器端发送第二次握手消息失败!" << endl;
        cout << "当前网络状态不佳, 请稍后再试" << endl;
        return 0;
    }

    // 接收第三次握手消息, 超时重传
    start = clock();
    while (1) {
        if (recvfrom(socketServer, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
            clientAddr, &len) != SOCKET_ERROR) {
            if (recvMsg.isACK() && recvMsg.ack == sendMsg.seq + 1 && !recvMsg
                .packetIncorruption()) {
                cout << "服务器端接收到第三次握手消息! 第三次握手成功! " <<
                    endl;
                break;
            }
        }
        if (clock() - start > RTO) {
            cout << "第二次握手超时,服务器端重新发送第二次握手消息" << endl;
            if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
                clientAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
                cout << "服务器端发送第二次握手消息失败!" << endl;
                cout << "当前网络状态不佳, 请稍后再试" << endl;
                return 0;
            }
            start = clock();
        }
    }

    // 设置套接字为阻塞模式
    mode = 0;
    ioctlsocket(socketServer, FIONBIO, (u_long FAR*) & mode);

    return 1;
```

```
}

//Client.cpp
bool waitConnect() {
    Message sendMsg, recvMsg;
    clock_t start;

    // 设置套接字为非阻塞模式
    int mode = 1;
    ioctlsocket(socketClient, FIONBIO, (u_long FAR*) & mode);

    // 发送第一次握手消息
    cout << "尝试建立连接! 客户端发送第一次握手消息" << endl;
    sendMsg.setSYN();
    sendMsg.seq = 2000;
    sendMsg.setChecksum();
    if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
        serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
        cout << "客户端发送第一次握手消息失败!" << endl;
        cout << "当前网络状态不佳, 请稍后再试" << endl;
        return 0;
    }

    // 接收第二次握手消息, 超时重传
    start = clock();
    while (1) {
        if (recvfrom(socketClient, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
            serverAddr, &len) != SOCKET_ERROR) {
            if (recvMsg.isSYN() && recvMsg.isACK() && recvMsg.ack == sendMsg.
                seq + 1 && !recvMsg.packetIncorruption()) {
                cout << "客户端接收到第二次握手消息! 第二次握手成功!" << endl
                    ;
                break;
            }
        }
        if (clock() - start > RTO) {
            cout << "第一次握手超时,客户端重新发送第一次握手消息" << endl;
            if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
                serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
                cout << "客户端发送第一次握手消息失败!" << endl;
                cout << "当前网络状态不佳, 请稍后再试" << endl;
                return 0;
            }
            start = clock();
        }
    }

    // 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*)&
            serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
107         cout << "客户端发送第三次握手消息失败!" << endl;
108         cout << "当前网络状态不佳，请稍后再试" << endl;
109         return 0;
110     }
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:

```cpp
//Server.cpp
bool closeConnect(Message recvMsg) {
    Message sendMsg;
    clock_t start;

    /*
        第一次挥手在recv_file函数里面处理
    */

    // 设置套接字为非阻塞模式
    int mode = 1;
    ioctlsocket(socketServer, FIONBIO, (u_long FAR*) & mode);

    // 发送第二次挥手消息
    cout << "服务器端发送第二次挥手消息! " << endl;
    sendMsg.setACK();
    sendMsg.seq = 4000;
    sendMsg.ack = recvMsg.seq + 1;
    sendMsg.setChecksum();
    if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
        clientAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
        cout << "服务器端发送第二次挥手消息失败!" << endl;
        cout << "当前网络状态不佳, 请稍后再试" << endl;
        return 0;
    }

    // 发送第三次挥手消息
    cout << "服务器端发送第三次挥手消息! " << endl;
    sendMsg.setFIN();
    sendMsg.setACK();
    sendMsg.seq = 5000;
    sendMsg.ack = recvMsg.seq + 1;
    sendMsg.setChecksum();
    if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
        clientAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
        cout << "服务器端发送第三次挥手消息失败!" << endl;
        cout << "当前网络状态不佳, 请稍后再试" << endl;
        return 0;
    }

    // 接收第四次挥手消息, 超时重传
    start = clock();
    while (1) {
        if (recvfrom(socketServer, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
            clientAddr, &len) != SOCKET_ERROR) {
            if (recvMsg.isACK() && recvMsg.ack == sendMsg.seq + 1 && !recvMsg
                .packetIncorruption()) {
```

```
44                  cout << "服务器端接收到第四次挥手消息! 第四次挥手成功! " <<
                        endl;
45                  break;
46              }
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*)&
                    clientAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
51                  cout << "服务器端发送第三次挥手消息失败!" << endl;
52                  cout << "当前网络状态不佳, 请稍后再试" << endl;
53                  return 0;
54              }
55              start = clock();
56          }
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*)&
            serverAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
73          cout << "客户端发送第一次挥手消息失败!" << endl;
74          cout << "当前网络状态不佳, 请稍后再试" << endl;
75          return 0;
76      }
77
78      // 接收第二次挥手消息, 超时重传
79      start = clock();
80      while (1) {
81          if (recvfrom(socketClient, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
                serverAddr, &len) != SOCKET_ERROR) {
82              if (recvMsg.isACK() && recvMsg.ack == sendMsg.seq + 1 && !recvMsg
                    .packetIncorruption()) {
83                  cout << "客户端接收到第二次挥手消息! 第二次挥手成功! " <<
                        endl;
84                  break;
85              }
```

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

```
    }

    return 1;
}
```

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:

```
void setChecksum() {
    this->checksum = 0;// 清0校验和字段
    int dataLen = this->len;// 数据部分长度
    int paddingLen = (16 - (dataLen % 16)) % 16;// 数据部分需要填0
    char* paddedData = new char[dataLen + paddingLen];// 填充后数据
    memcpy(paddedData, this->data, dataLen);
    memset(paddedData + dataLen, 0, paddingLen);
    // 分段求和，并处理溢出
    u_short* buffer = (u_short*)this;
    int sum = 0;
    for (int i = 0; i < (sizeof(Message) + paddingLen) / 2; i++) {
        sum += buffer[i];
        if (sum > 0xFFFF) {
            sum = (sum & 0xFFFF) + (sum >> 16);
        }
    }
    // 计算结果取反写入校验和字段
    this->checksum = ~sum;
    // 释放动态分配的内存
    delete[] paddedData;
}
bool packetIncorruption() {
    // 计算数据长度并填充
    int dataLen = this->len;
    int paddingLen = (16 - (dataLen % 16)) % 16;
    // 使用动态内存分配
    char* paddedData = new char[dataLen + paddingLen];
    memcpy(paddedData, this->data, dataLen);
    memset(paddedData + dataLen, 0, paddingLen);
```

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

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`。

```
void send_file() {
    Message sendMsg, recvMsg;
    clock_t start, end;
    char filePath[20];
    ifstream in;
    int filePtrLoc;
    int dataAmount;
    int packetNum;
    int checksum;

    cout << "请输入要发送的文件名: ";
    memset(filePath, 0, 20);
    string temp;
    cin >> temp;
    string inputPath = "./input/" + temp;

    if (temp == "quit") {
        closeConnect();
        quit = true;
        return;
    }
```

```
22      else if (temp == "1.jpg" || temp == "2.jpg" || temp == "3.jpg" || temp ==
            "helloworld.txt") {
23          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 << "发送seq:" << sendMsg.seq << endl;
46      cout << "base: " << base << endl;
47      cout << "next_seq: " << next_seq << endl;
48      cout << "len:" << sendMsg.len << endl;
49      cout << "checksum: " << sendMsg.checksum << endl;
50      if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
            routerAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
51          cout << "客户端发送文件名失败!" << endl;
52          return;
53      }
54
55      start = clock();
56      while (1) {
57          if (recvfrom(socketClient, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
                routerAddr, &len) != SOCKET_ERROR) {
58              if (recvMsg.isACK() && recvMsg.ack == sendMsg.seq + 1 && !recvMsg
                    .packetIncorrection()) {
59                  cout << "客户端发送文件名成功!" << endl;
60                  break;
61              }
62          }
63          if (clock() - start > TIMEOUT) {
64              cout << "应答超时, 客户端重新发送文件名" << endl;
```

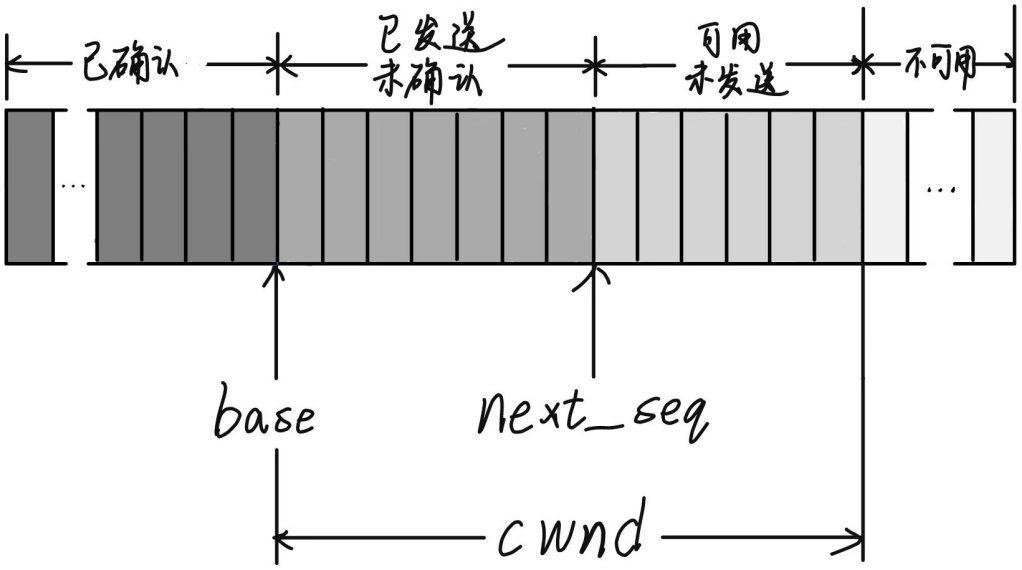
```
65              cout << "checksum: " << sendMsg.checksum << endl << endl;
66              if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
                    routerAddr, sizeof(SOCKADDR)) == SOCKET_ERROR) {
67                  cout << "客户端发送文件名失败!" << endl;
68                  cout << "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。

```
const int cwnd = 5; // 窗口大小
int base = 1; // 窗口的起始序列号
int next_seq = 1; // 下一个待发送的序列号
mutex seq_mutex; // 序列号的互斥锁
bool send_over = false; // 传输完毕
map<int, Message> send_buffer; // 序列号及其报文的映射
map<int, clock_t> send_times;  // 序列号及其发送时间的映射



// 发送线程
void send_thread(SOCKET socketClient, sockaddr_in& routerAddr, ifstream& in,
    int filePtrLoc, int packetNum) {
    Message sendMsg;
    int count = 0;

    // 窗口内发送数据
    while (!send_over) {
        {
            unique_lock<mutex> lock(seq_mutex);// 加锁
            while (next_seq <= packetNum && (next_seq - base) < cwnd) {
                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, (
                    SOCKADDR*)&routerAddr, sizeof(SOCKADDR)) == SOCKET_ERROR)
                     {
43                  cout << "发送数据包失败!" << endl;
44              }
45
46              // 记录发送时间
47              send_buffer[next_seq] = sendMsg;
48              send_times[next_seq] = clock();
49
50              next_seq++;
51          }
52          lock.unlock(); // 解锁
53      }
54      {
55          unique_lock<mutex> lock(seq_mutex);// 加锁
56          // 超时重传
57          for (auto it = send_buffer.begin(); it != send_buffer.end(); ) {
58              if (clock() - send_times[it->first] > TIMEOUT) {
59                  cout << "应答超时, 重新发送已发送未确认的数据包" << endl;
60
61                  Message sendMsg = it->second;
62                  cout << "发送seq:" << it->first << endl;
63                  cout << "base: " << base << endl;
64                  cout << "next_seq: " << next_seq << endl;
65                  cout << "len:" << sendMsg.len << endl;
66                  cout << "checksum: " << sendMsg.checksum << endl << endl;
67
```

```
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
```

```
                if (recvMsg.ack == packetNum + 1) {
                    cout << "文件传输完成! " << endl;
                    send_over = true;
                }
            }
            lock.unlock(); // 解锁
        }
    }
}
```

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。

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

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。

```
{
    unique_lock<mutex> lock(seq_mutex);// 加锁
    // 超时重传
    for (auto it = send_buffer.begin(); it != send_buffer.end(); ) {
        if (clock() - send_times[it->first] > TIMEOUT) {
            cout << "应答超时，重新发送已发送未确认的数据包" << endl;
            Message sendMsg = it->second;
            cout << "发送seq:" << it->first << endl;
            cout << "base: " << base << endl;
            cout << "next_seq: " << next_seq << endl;
            cout << "len:" << sendMsg.len << endl;
            cout << "checksum: " << sendMsg.checksum << endl << endl;
            if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
                route
                cout << "重传数据包失败!" << endl;
            }
            // 重置计时器
            send_times[it->first] = clock();
        }
        it++;
    }
    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。

```
void recv_file() {
    cout << "服务器正在等待接收文件中......" << endl;
    Message recvMsg, sendMsg;
    clock_t start, end;
    char filePath[20];
    string outputPath;
    ofstream out;
    int dataAmount = 0;
    int packetNum;

    // 接收文件名
    while (1) {
        if (recvfrom(socketServer, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
            routerAddr, &len) != SOCKET_ERROR) {
            // 接收第一次挥手信息
            if (recvMsg.isFIN() && !recvMsg.packetIncorrection()) {
                cout << "客户端准备断开连接! 进入挥手模式! " << endl;
                cout << "服务器端接收到第一次挥手消息! 第一次挥手成功!" <<
                    endl;
                closeConnect(recvMsg);
                quit = true;
                return;
            }
            if (recvMsg.isSTART() && !recvMsg.packetIncorrection()) {
                ZeroMemory(filePath, 20);
                memcpy(filePath, recvMsg.data, recvMsg.len);
                outputPath = "./output/" + string(filePath);
                out.open(outputPath, ios::out | ios::binary);//以写入模式、二
                    进制模式打开文件
                cout << "文件名为: " << filePath << endl;
                cout << "接收到的seq:" << recvMsg.seq << endl;
                cout << "checksum: " << recvMsg.checksum << endl;

                if (!out.is_open()) {
                    cout << "文件打开失败! ! ! " << endl;
                    exit(1);
                }

                packetNum = recvMsg.num;
                cout << "文件" << filePath << "有" << packetNum << "个数据包"
```

```
                    << endl;

                // 发送ack给客户端
                sendMsg.setACK();
                sendMsg.ack = recvMsg.seq + 1;
                sendMsg.setChecksum();
                if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (
                    SOCKADDR*)&routerAddr, sizeof(SOCKADDR)) == SOCKET_ERROR)
                     {
                    cout << "服务器端发送ack报文失败!" << endl;
                    cout << "当前网络状态不佳，请稍后再试" << endl;
                    return;
                }
                break;
            }
        }
    }

    // 设置套接字为阻塞模式
    int mode = 0;
    ioctlsocket(socketServer, FIONBIO, (u_long FAR*) & mode);

    // 开始接收文件内容
    cout << "服务器端开始接收文件内容! " << endl << endl;
    int expected_seq = 1;
    bool first_recv_incorrect_seq = true;
    start = clock();
    while (1) {
        if (recvfrom(socketServer, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
            routerAddr, &len) != SOCKET_ERROR) {
            // 如果校验和错误
            if (recvMsg.packetIncorrection()) {
                cout << "checksum错误! " << endl;
                cout << "checksum: " << recvMsg.checksum << endl;
                continue;
            }
            if (recvMsg.seq < expected_seq) {// 去重
                continue;
            }
            else {
                // 如果接收到的是期望接收到的seq
                if (recvMsg.seq == expected_seq) {
                    first_recv_incorrect_seq = true;
                    expected_seq++;
                }
                // 如果接收到的不是期望接收到的seq
                else{
                    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_recv_incorrect_seq = false;
96              }
97              // 存储接收到的数据包
98              recv_buffer[recvMsg.seq] = recvMsg;
99              cout << "接收seq:" << recvMsg.seq << endl;
100             cout << "len:" << recvMsg.len << endl;
101             cout << "checksum: " << recvMsg.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 = recv_buffer.begin(); it != recv_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 && (recv_buffer.size() >= cwnd || recv_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 = recv_buffer.begin(); it != recv_buffer.end();
                    ++it) {
122                 const Message& bufferRecvMsg = it->second;
123                 out.write(bufferRecvMsg.data, bufferRecvMsg.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 << "吞吐率" << (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

累计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，就让期望接的 seq 自增 1；如果接收到的不是期望接收到的 seq，就将期望收到的 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。

```
if (recvfrom(socketServer, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&routerAddr, &le
    // 如果校验和错误
    if (recvMsg.packetIncorrection()) {
        cout << "checksum错误！" << endl;
        cout << "checksum: " << recvMsg.checksum << endl;
        continue;
    }
if (recvMsg.seq < expected_seq) {// 去重
    continue;
}
else {
    // 如果接收到的是期望接收到的seq
    if (recvMsg.seq == expected_seq) {
        first_recv_incorrect_seq = true;
        expected_seq++;
    }
    // 如果接收到的不是期望接收到的seq
    else{
        if (first_recv_incorrect_seq) {
            // 发送累计确认的ack给客户端
            cout << "期望接收seq:" << expected_seq << endl;
            sendMsg.setACK();
            sendMsg.ack = expected_seq;
            sendMsg.setChecksum();
            cout << "发送累计确认ack:" << sendMsg.ack << endl << endl;
            if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
                r
                cout << "服务器端发送ack报文失败!" << endl;
                cout << "当前网络状态不佳，请稍后再试" << endl;
                return;
            }
        }
        first_recv_incorrect_seq = false;
    }
    // 存储接收到的数据包
    recv_buffer[recvMsg.seq] = recvMsg;
    cout << "接收seq:" << recvMsg.seq << endl;
    cout << "len:" << recvMsg.len << endl;
    cout << "checksum: " << recvMsg.checksum << endl << endl;
}
// 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
```

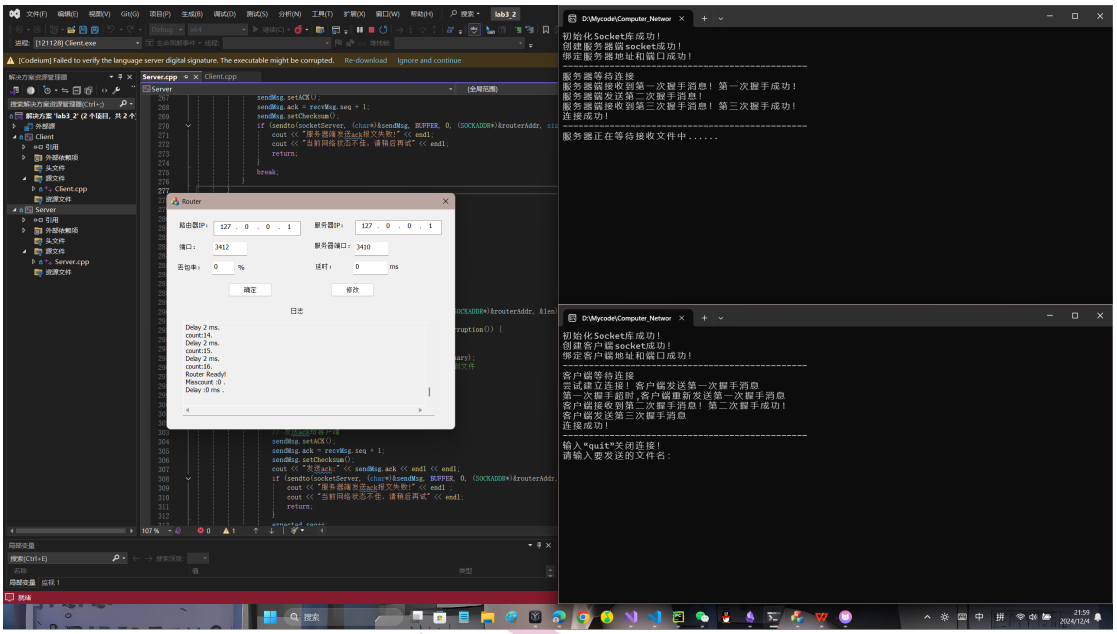
```
bool isSeqInOrder = true;
int prevSeq = 0;
for (auto it = recv_buffer.begin(); it != recv_buffer.end(); ++it) {
    if (prevSeq != 0 && it->first != prevSeq + 1) {
        isSeqInOrder = false;
        break;
    }
    prevSeq = it->first;
}
// 如果接收缓冲区满了或者接收到最后一组数据包
if (isSeqInOrder && (recv_buffer.size() >= cwnd || recv_buffer.rbegin()->
    first
    cout << "接收缓冲区写入文件" << endl << endl;
    // 以追加模式打开文件，并写入文件
    ofstream out(outputPath, ios::app | std::ios::binary);
    // 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
    for (auto it = recv_buffer.begin(); it != recv_buffer.end(); ++it) {
        const Message& bufferRecvMsg = it->second;
        out.write(bufferRecvMsg.data, bufferRecvMsg.len);// 写入数据到文件
        dataAmount += bufferRecvMsg.len;
    }
    out.close();
    // 发送ack给客户端
    sendMsg.setACK();
    sendMsg.ack = recv_buffer.rbegin()->first + 1;
    sendMsg.setChecksum();
    cout << "发送ack:" << sendMsg.ack << endl << endl;
    if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (SOCKADDR*)&
        routerAdd
        cout << "服务器端发送ack报文失败!" << endl;
        cout << "当前网络状态不佳，请稍后再试" << endl;
        return;
    }
    // 检查文件传输是否结束
    if (recv_buffer.rbegin()->first == packetNum) {
        end = clock();
        cout << "接收文件成功! " << endl;
        out.close();
        out.clear();
        double TotalTime = (double)(end - start) / CLOCKS_PER_SEC;
        cout << "传输总时间" << TotalTime << "s" << endl;
        cout << "吞吐率" << (double)dataAmount / TotalTime << "␣bytes/s" <<
            en
        return;
    }
    recv_buffer.clear(); // 清空接收缓冲区
    expected_seq = sendMsg.ack;
}
```

八、 测试

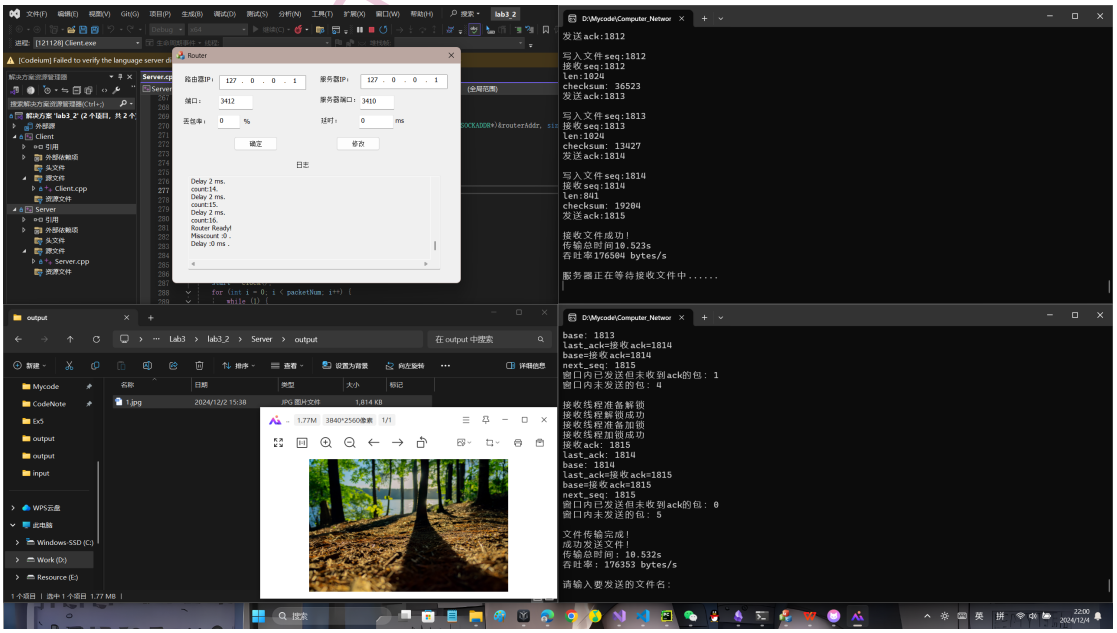
我们分别定义了服务器端、客户端、router 的端口号，IP 统一使用 127.0.0.1。

```
#define SERVER_PORT 3410
#define CLIENT_PORT 3411
#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。

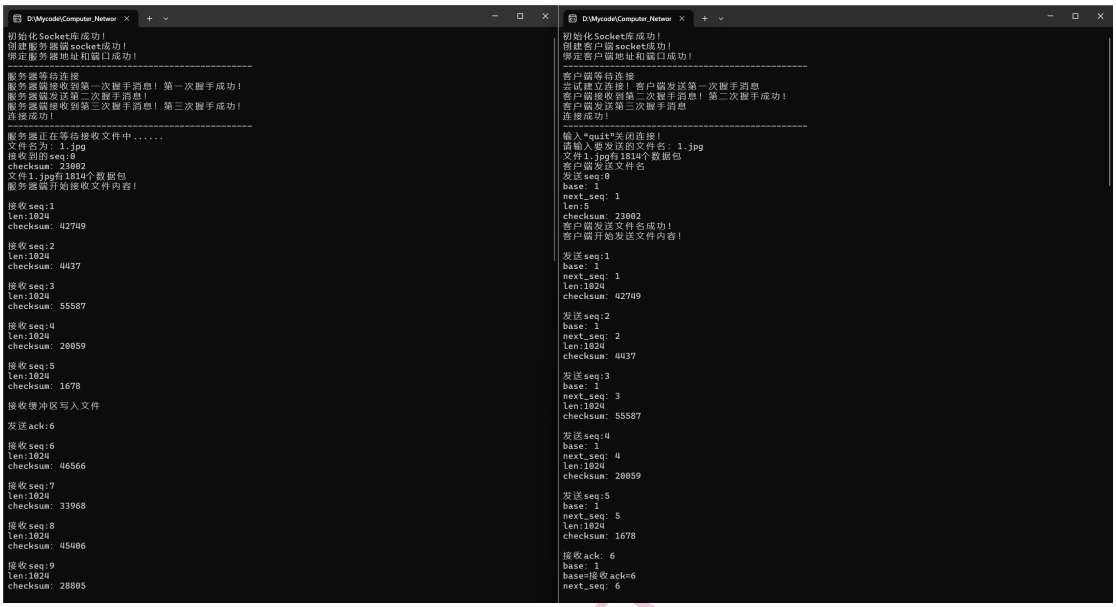


接下来发送文件“1.jpg”。检查 Server output 目录，出现文件“1.jpg”，且与客户端发送的文件大小相等。

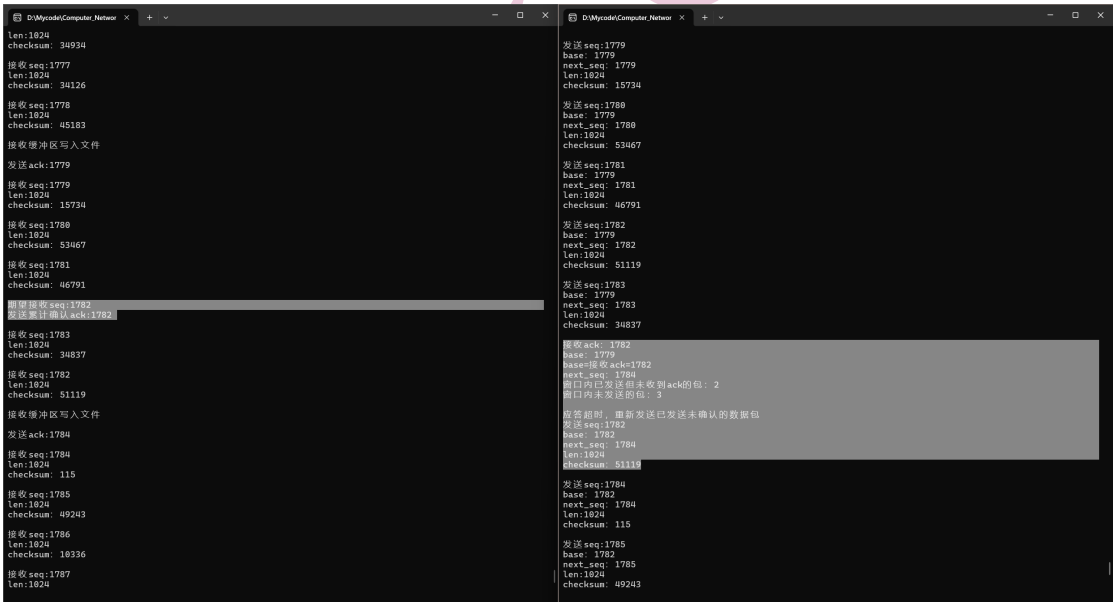


接下来测试滑动窗口的超时重传，打开路由程序，设置 3% 的丢包率和 2ms 延迟，开始运行。

流水线发送

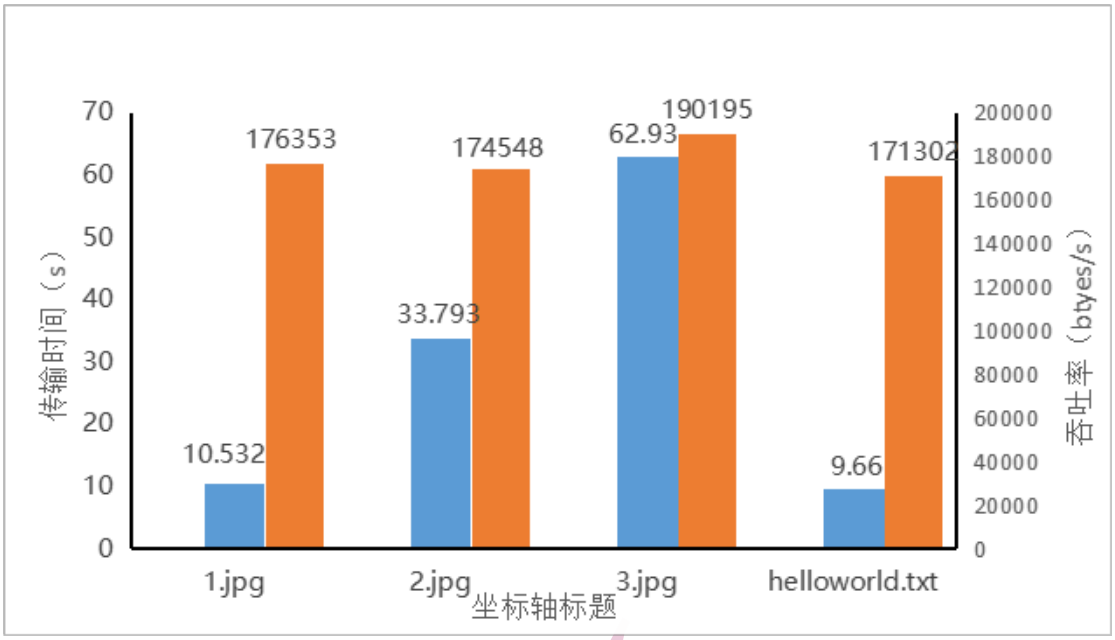


超时重传。客户端发送的过程中发生丢包，seq 为 1782，客户端会尝试重新发送窗口内已发送未确认的数据包。



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（btyes/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：