



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 12 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. 慢启动算法	15
2. 拥塞避免算法	15
3. 快重传算法	15
4. 快恢复算法	16
七、 接收文件	16
1. 确认重传	20
八、 测试	21

一、 实验目的

实验 3-3: 在实验 3-2 的基础上, 选择实现一种拥塞控制算法, 也可以是改进的算法, 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。

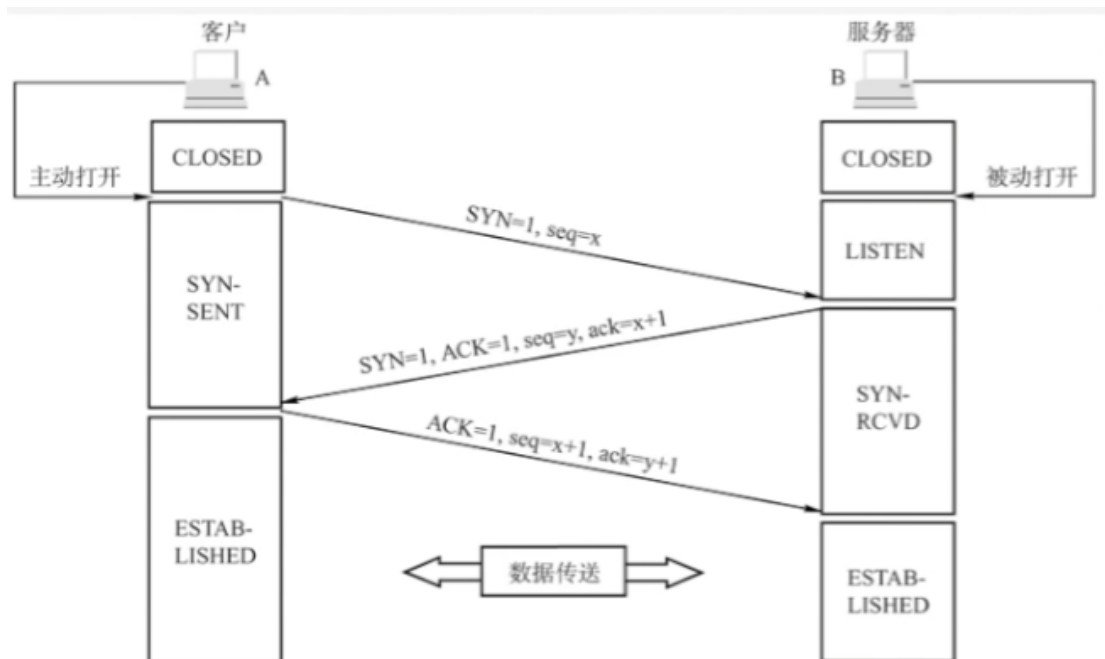
```
1 void setSYN() { this->flag |= 1; }
2 void setACK() { this->flag |= 2; }
3 void setFIN() { this->flag |= 4; }
4 void setSTART() { this->flag |= 8; }
5 void setEND() { this->flag |= 16; }
```

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;
```

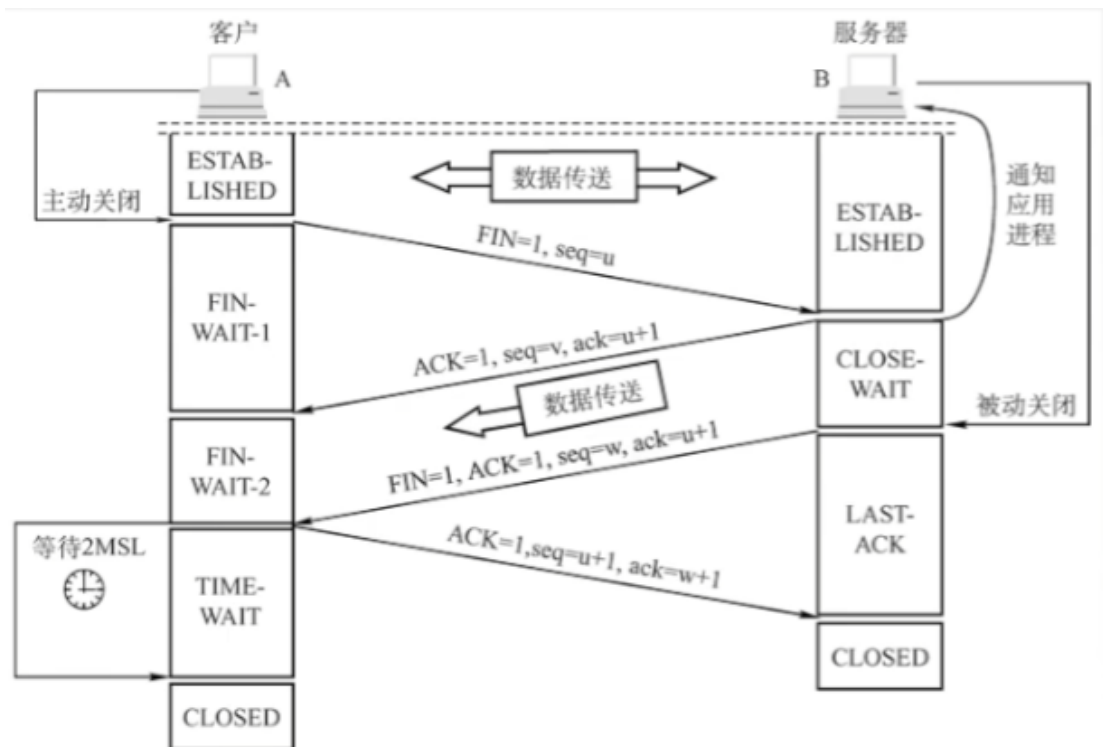
```
58 }
59
60 //Client.cpp
61 bool waitConnect() {
62     Message sendMsg, recvMsg;
63     clock_t start;
64
65     // 设置套接字为非阻塞模式
66     int mode = 1;
67     ioctlsocket(socketClient, FIONBIO, (u_long FAR*) & mode);
68
69     // 发送第一次握手消息
70     cout << "尝试建立连接! 客户端发送第一次握手消息" << endl;
71     sendMsg.setSYN();
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 // 发送线程
2 void send_thread(SOCKET socketClient, sockaddr_in& routerAddr, ifstream& in,
3     int filePtrLoc, int packetNum) {
4     Message sendMsg;
5
6     // 窗口内发送数据
7     while (!send_over) {
8         {
9             unique_lock<mutex> lock(seq_mutex); // 加锁
10            if (next_seq <= packetNum && (next_seq - base) < min(cwnd, rwnd))
11                {
12                    if (next_seq == packetNum) {
13                        in.read(sendMsg.data, filePtrLoc);
14                        sendMsg.len = filePtrLoc;
15                        sendMsg.setEND();
16                        filePtrLoc = 0;
17                    }
18                    else {
19                        in.read(sendMsg.data, 1024);
20                        sendMsg.len = 1024;

```

```

20         filePtrLoc -= 1024;
21     }
22
23     // 发送数据包
24     sendMsg.seq = next_seq;
25     sendMsg.setChecksum();
26     if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (
        SOCKADDR*)&routerAddr, sizeof(SOCKADDR)) == SOCKET_ERROR)
    {
27         cout << "发送数据包失败!" << endl;
28     }
29     // 记录发送时间
30     send_buffer[next_seq] = sendMsg;
31     send_times[next_seq] = clock();
32
33     cout << "发送seq:" << next_seq << endl;
34     cout << "checksum:" << sendMsg.checksum << endl;
35     cout << "ssthresh:" << ssthresh << endl;
36     cout << "swnd:" << min(cwnd, rwnd) << endl;
37     cout << "base:" << base << endl;
38
39     next_seq++;
40
41     cout << "next_seq:" << next_seq << endl;
42     cout << "top:" << base + min(cwnd, rwnd) << endl << endl;
43 }
44 lock.unlock(); // 解锁
45 }
46 {
47     unique_lock<mutex> lock(seq_mutex); // 加锁
48     // 超时重传
49     for (auto it = send_buffer.begin(); it != send_buffer.end(); it
        ++){
50         if (clock() - send_times[it->first] > TIMEOUT) {
51             cout << "应答超时, 重新发送已发送未确认的数据包" << endl;
52
53             Message sendMsg = it->second;
54             if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (
                SOCKADDR*)&routerAddr, sizeof(SOCKADDR)) ==
                SOCKET_ERROR) {
55                 cout << "重传数据包失败!" << endl;
56             }
57             // 重置计时器
58             send_times[it->first] = clock();
59
60             cout << "发送seq:" << it->first << endl;
61             cout << "checksum:" << sendMsg.checksum << endl;
62             cout << "ssthresh:" << ssthresh << endl;

```

```

63         cout << "swnd:" << min(cwnd, rwnd) << endl;
64         cout << "base: " << base << endl;
65         cout << "next_seq: " << next_seq << endl;
66         cout << "top: " << base + min(cwnd, rwnd) << endl << endl
        ;
67
68         cout << "重新发送完毕" << endl;
69     }
70 }
71 lock.unlock(); // 解锁
72 }
73
74 }
75 }
76
77 // 接收线程
78 void recv_thread(SOCKET socketClient, int packetNum) {
79     Message recvMsg;
80     clock_t start;
81     int expected_ack = 2; // 期望收到的ack
82     int prev_ack = 0; // 上一次收到的确认号
83
84     while (!send_over) {
85         if (recvfrom(socketClient, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
86             routerAddr, &len) != SOCKET_ERROR) {
87             {
88                 unique_lock<mutex> lock(seq_mutex); // 加锁
89                 if (recvMsg.isACK() && !recvMsg.packetIncorrection()) {
90                     if (recvMsg.ack > prev_ack) { // 新的 ack
91                         cout << "接收新的ack:" << recvMsg.ack << endl;
92                         dup_ack_count = 0; // 重置重复 ACK 计数
93                         if (recvMsg.ack >= expected_ack) {
94                             // 移除已确认的数据包
95                             auto it = send_buffer.begin();
96                             while (it != send_buffer.end() && it->first <
97                                 recvMsg.ack) {
98                                 it = send_buffer.erase(it); // 从发送缓冲区中
99                                     删去对应报文
100                                 send_times.erase(recvMsg.ack); // 同时移除计
101                                     时器
102                             }
103                             base = recvMsg.ack;
104
105                             if (cwnd < ssthresh) {
106                                 // 慢启动
107                                 cout << "慢启动:" << endl;
108                                 cwnd *= 2;
109                             }
110                         }
111                     }
112                 }
113             }
114         }
115     }
116 }

```

```

106         else {
107             // 拥塞避免
108             cout << "拥塞避免:" << endl;
109             cwnd += 1;
110         }
111         expected_ack += min(cwnd, rwnd);
112     }
113     prev_ack = recvMsg.ack;
114
115     // 展示窗口情况
116     cout << "ssthresh:" << ssthresh << endl;
117     cout << "swnd:" << min(cwnd, rwnd) << endl;
118     cout << "base:" << base << endl;
119     cout << "next_seq:" << next_seq << endl;
120     cout << "top:" << base + min(cwnd, rwnd) << endl <<
        endl;
121
122     }
123     else if (recvMsg.ack == prev_ack) { //重复 ack
124         if (dup_ack_count < 3) {
125             dup_ack_count++;
126             cout << "接收到第" << dup_ack_count << "个重复ack
                : " << recvMsg.ack << endl;
127             cwnd += 1;
128         }
129         if (dup_ack_count == 3) { // 收到第3个重复 ACK
130             // 快重传
131             cout << "快重传:" << endl;
132             Message sendMsg = send_buffer.find(prev_ack)->
                second;
133             if (sendto(socketClient, (char*)&sendMsg, BUFFER,
                0, (SOCKADDR*)&routerAddr, sizeof(SOCKADDR))
                == SOCKET_ERROR) {
134                 cout << "重传数据包失败!" << endl;
135             }
136             cout << "发送seq:" << sendMsg.seq << endl;
137             cout << "checksum:" << sendMsg.checksum << endl;
138             cout << "ssthresh:" << ssthresh << endl << endl;
139
140             // 快恢复
141             cout << "快恢复:" << endl;
142             ssthresh = cwnd / 2;
143             cwnd = ssthresh + 3;
144
145             // 展示窗口情况
146             cout << "ssthresh:" << ssthresh << endl;
147             cout << "swnd:" << min(cwnd, rwnd) << endl;
148             cout << "base:" << base << endl;

```



```

149         cout << "next_seq:" << next_seq << endl;
150         cout << "top:" << base + min(cwnd, rwnd) << endl
151             << endl;
152
153         dup_ack_count = INT_MAX;
154     }
155     // 如果所有文件传输结束
156     if (recvMsg.ack == packetNum + 1) {
157         cout << "文件传输完成!" << endl;
158         send_over = true;
159     }
160 }
161 lock.unlock(); // 解锁
162 }
163 }
164 }
165 }

```

1. 慢启动算法

当 $cwnd < ssthresh$ 时, 使用慢启动算法

先发送少量数据试探一下网络的拥塞程度, 每个传输轮次拥塞窗口 $cwnd$ 按指数增长。

```

1 if (cwnd < ssthresh) {
2     // 慢启动
3     cout << "慢启动:" << endl;
4     cwnd *= 2;
5 }

```

2. 拥塞避免算法

当 $cwnd > ssthresh$ 时, 停止使用慢启动算法而改用拥塞避免算法。

```

1 else {
2     // 拥塞避免
3     cout << "拥塞避免:" << endl;
4     cwnd += 1;
5 }

```

3. 快重传算法

有时个别报文段会在网络中丢失, 但实际上网络并未发生拥塞。这将导致发送方超时重传, 并误认为网络发生了拥塞, 把拥塞窗口 $cwnd$ 又设置为最小值 1, 错误地启动慢启动算法, 因而降低了传输效率。

快重传算法要求接收方收到数据, 立即发送确认。接收方如果收到失序的报文段, 发出对已收到的报文段的重复确认。发送方一旦收到 3 个连续的重复确认, 就将相应的报文段立即重传

```

1 else if (recvMsg.ack == prev_ack) { //重复 ack
2     if (dup_ack_count < 3) {
3         dup_ack_count++;
4         cout << "接收到第" << dup_ack_count << "个重复ack:" << re
5         cwnd += 1;
6     }
7     if (dup_ack_count == 3) { // 收到第3个重复 ACK
8         // 快重传
9         cout << "快重传:" << endl;
10        Message sendMsg = send_buffer.find(prev_ack)->second;
11        if (sendto(socketClient, (char*)&sendMsg, BUFFER, 0, (SOC
12            cout << "重传数据包失败!" << endl;
13        }
14        cout << "发送seq:" << sendMsg.seq << endl;
15        cout << "checksum:" << sendMsg.checksum << endl;
16        cout << "sssthresh:" << sssthresh << endl << endl;

```

4. 快恢复算法

发送方一旦收到 3 个重复确认, 就知道现在只是丢失了个别的报文段。于是不启动慢启动算法, 而执行快恢复算法。

```

1 // 快恢复
2 cout << "快恢复:" << endl;
3 sssthresh = cwnd / 2;
4 cwnd = sssthresh + 3;

```

加 3 是因为发送方收到 3 个重复的确认, 就表明有 3 个数据报文段已经离开了网络, 这 3 个报文段不再消耗网络资源而是停留在接收方的接收缓存中, 可见现在网络中不是堆积了报文段而是减少了 3 个报文段。因此可以适当把拥塞窗口扩大些。

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                  cout << "文件名为: " << filePath << endl;
31                  cout << "接收到的seq:" << recvMsg.seq << endl;
32                  cout << "checksum: " << recvMsg.checksum << endl;
33
34                  if (!out.is_open()) {
35                      cout << "文件打开失败!!! " << endl;
36                      exit(1);
37                  }
38
39                  packetNum = recvMsg.num;
40                  cout << "文件" << filePath << "有" << packetNum << "个数据包"
41                      << endl;
42
43                  // 发送ack给客户端
44                  sendMsg.setACK();
45                  sendMsg.ack = recvMsg.seq + 1;
46                  sendMsg.setChecksum();
47                  if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (
48                      SOCKADDR*)&routerAddr, sizeof(SOCKADDR)) == SOCKET_ERROR)
49                      {
50                          cout << "服务器端发送ack报文失败!" << endl;
51                          cout << "当前网络状态不佳, 请稍后再试" << endl;
52                          return;
53                      }
54                  break;
55              }
56          }
57      }
```

```
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     start = clock();
61     while (1) {
62         if (recvfrom(socketServer, (char*)&recvMsg, BUFFER, 0, (SOCKADDR*)&
63             routerAddr, &len) != SOCKET_ERROR) {
64             // 如果校验和错误
65             if (recvMsg.packetIncorrection()) {
66                 cout << "checksum错误!" << endl;
67                 cout << "checksum: " << recvMsg.checksum << endl;
68                 continue;
69             }
70             // 存储接收到的数据包
71             recv_buffer[recvMsg.seq] = recvMsg;
72             cout << "接收seq:" << recvMsg.seq << endl;
73             cout << "checksum:" << recvMsg.checksum << endl;
74
75             if (recvMsg.seq == expected_seq) {
76
77                 auto last_seq = recv_buffer.rbegin()->first;
78
79                 // 查找下一个期望的序列号
80                 bool foundGap = false;
81                 for (int i = expected_seq + 1; i <= last_seq; ++i) {
82                     if (recv_buffer.find(i) == recv_buffer.end()) {
83                         // 找到第一个gap, 即下一个期望的序列号
84                         expected_seq = i;
85                         foundGap = true;
86                         break;
87                     }
88                 }
89                 if (!foundGap) {
90                     // 如果没有找到gap, 那么expected_seq就是最后一个序列号加1
91                     expected_seq = last_seq + 1;
92                 }
93
94                 // 发送ack给客户端
95                 sendMsg.setACK();
96                 sendMsg.ack = expected_seq;
97                 sendMsg.setChecksum();
98                 cout << "发送ack:" << sendMsg.ack << endl << endl;
```

```

98         if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (
          SOCKADDR*)&routerAddr, sizeof(SOCKADDR)) == SOCKET_ERROR)
          {
99             cout << "服务器端发送ack报文失败!" << endl;
100             cout << "当前网络状态不佳, 请稍后再试" << endl;
101             return;
102         }
103     }
104     // 如果接收到的不是期望接收的seq
105     else {
106         // 发送累计确认的ack给客户端
107         cout << "期望接收seq:" << expected_seq << endl;
108
109         sendMsg.setACK();
110         sendMsg.ack = expected_seq;
111         sendMsg.setChecksum();
112         cout << "发送累计确认ack:" << sendMsg.ack << endl << endl;
113         if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (
          SOCKADDR*)&routerAddr, sizeof(SOCKADDR)) == SOCKET_ERROR)
          {
114             cout << "服务器端发送ack报文失败!" << endl;
115             cout << "当前网络状态不佳, 请稍后再试" << endl;
116             return;
117         }
118     }
119
120     // 如果接收到所有数据包
121     if (expected_seq == packetNum + 1) {
122         cout << "接收缓冲区写入文件" << endl << endl;
123         // 以追加模式打开文件, 并写入文件
124         ofstream out(outputPath, ios::app | std::ios::binary);
125         // 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
126         for (auto it = recv_buffer.begin(); it != recv_buffer.end();
          ++it) {
127             const Message& bufferRecvMsg = it->second;
128             out.write(bufferRecvMsg.data, bufferRecvMsg.len);
129             dataAmount += bufferRecvMsg.len;
130         }
131         out.close();
132         recv_buffer.clear(); // 清空接收缓冲区
133
134
135         end = clock();
136         cout << "接收文件成功!" << endl;
137         out.close();
138         out.clear();
139
140         double TotalTime = (double)(end - start) / CLOCKS_PER_SEC;

```

```

141         cout << "传输总时间" << TotalTime << "s" << endl;
142         cout << "吞吐率" << (double)dataAmount / TotalTime << "bytes
           /s" << endl << endl;
143
144         return;
145     }
146
147     }
148 }
149 }

```

1. 确认重传

服务器端先将接收到的数据包存到接收缓冲区，然后判断 seq 是否为期望的序列号。

如果是期望的序列号，就寻找下一个期望的序列号（接收缓冲区的第一个 gap 或者接收缓冲区最后一个序列号加 1）。最后将期望的序列号作为 ack 的内容发送给客户端。

如果不是期望的序列号，说明发送丢包，就将期望收到的序列号作为 ack 报文的内容发送给客户端。

如果接收到所有数据包，则尝试写入文件，并计算传输总时间和吞吐率。

```

1  if (recvMsg.seq == expected_seq) {
2      auto last_seq = recv_buffer.rbegin()->first;
3      // 查找下一个期望的序列号
4      bool foundGap = false;
5      for (int i = expected_seq + 1; i <= last_seq; ++i) {
6          if (recv_buffer.find(i) == recv_buffer.end()) {
7              // 找到第一个gap，即下一个期望的序列号
8              expected_seq = i;
9              foundGap = true;
10             break;
11         }
12     }
13     if (!foundGap) {
14         // 如果没有找到gap，那么expected_seq就是最后一个序列
15         expected_seq = last_seq + 1;
16     }
17     // 发送ack给客户端
18     sendMsg.setACK();
19     sendMsg.ack = expected_seq;
20     sendMsg.setChecksum();
21     cout << "发送ack:" << sendMsg.ack << endl << endl;
22     if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (SOCKET)0, 0) < 0) {
23         cout << "服务器端发送ack报文失败!" << endl;
24         cout << "当前网络状态不佳，请稍后再试" << endl;
25         return;
26     }
27 }
28 // 如果接收到的不是期望接收的seq

```

```

29 else {
30     // 发送累计确认的ack给客户端
31     cout << "期望接收seq:" << expected_seq << endl;
32     sendMsg.setACK();
33     sendMsg.ack = expected_seq;
34     sendMsg.setChecksum();
35     cout << "发送累计确认ack:" << sendMsg.ack << endl << endl
36     if (sendto(socketServer, (char*)&sendMsg, BUFFER, 0, (SOC
37         cout << "服务器端发送ack报文失败!" << endl;
38         cout << "当前网络状态不佳, 请稍后再试" << endl;
39         return;
40     }
41 }

```

八、 测试

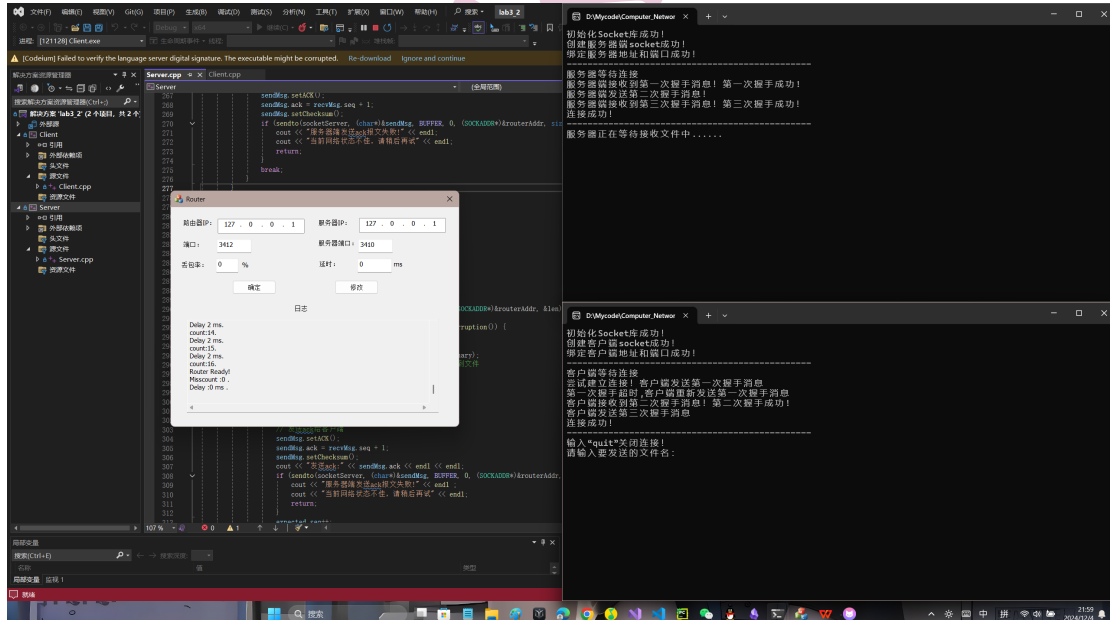
我们分别定义了服务器端、客户端、router 的端口号, IP 统一使用 127.0.0.1。

```

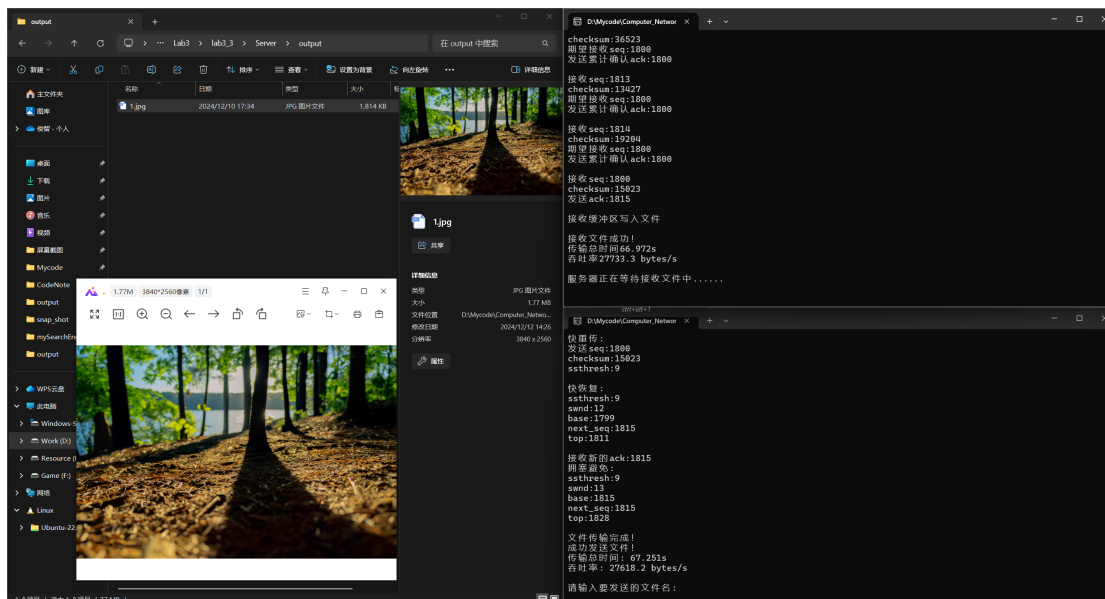
1 #define SERVER_PORT 3410
2 #define CLIENT_PORT 3411
3 #define ROUTER_PORT 3412

```

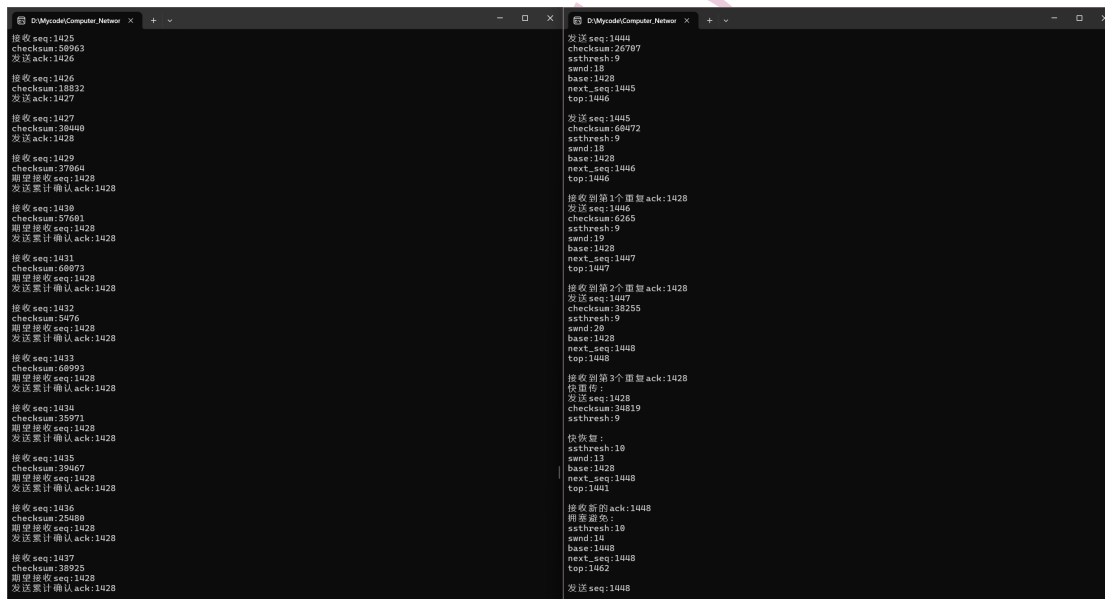
打开路由程序, 设置 0% 的丢包率和 0ms 延迟, 开始运行, 首先建立连接。



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。



接下来测试快重传，打开路由程序，设置 3% 的丢包率和 2ms 延迟，开始运行。



客户端发送的过程中发生丢包，seq 为 1428，服务器端再接受数据包，均返回 ack1428。

客户端收到 3 个冗余 ack1428 后，立刻执行快重传算法，重传 seq 为 1428 的数据包，而不是等待到超时重传。

执行快重传算法后，不启动慢启动算法，而执行快恢复算法。


```

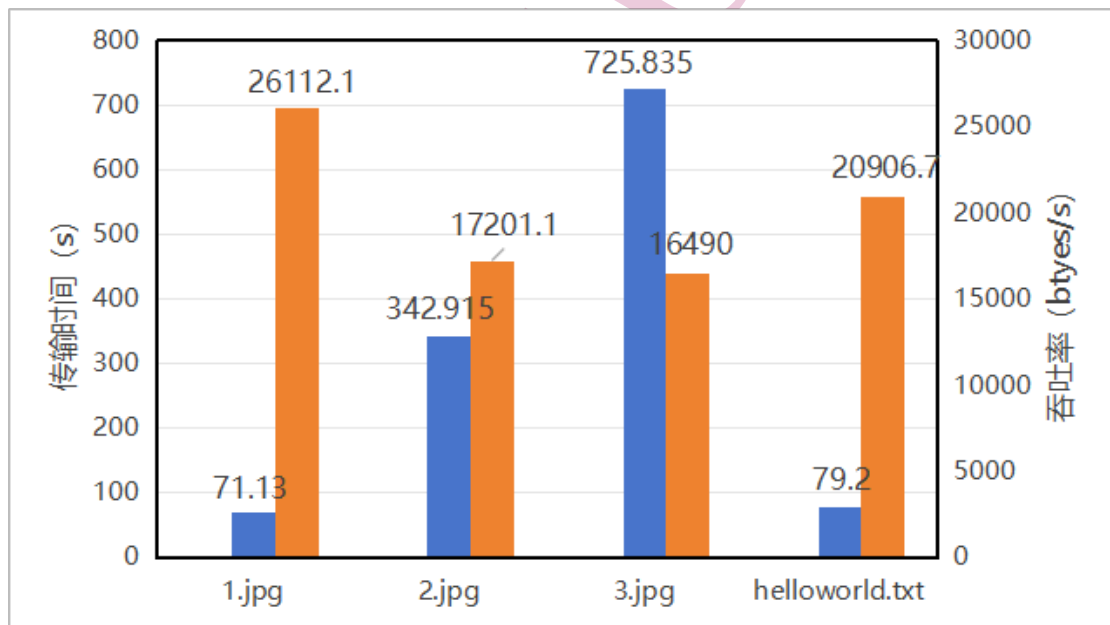
D:\MyCode\Computer_Network
发送累计确认ack:1428
接收seq:1428
checksum:52648
期望接收seq:1428
发送累计确认ack:1428
接收seq:1443
checksum:3923
期望接收seq:1428
发送累计确认ack:1428
接收seq:1444
checksum:26797
期望接收seq:1428
发送累计确认ack:1428
接收seq:1445
checksum:68472
期望接收seq:1428
发送累计确认ack:1428
接收seq:1446
checksum:6265
期望接收seq:1428
发送累计确认ack:1428
接收seq:1447
checksum:38255
期望接收seq:1428
发送累计确认ack:1428
接收seq:1428
checksum:34819
期望接收seq:1448
发送seq:1448
checksum:62803
发送ack:1449
接收seq:1449
checksum:46583
发送ack:1450
接收seq:1450
checksum:45877
发送ack:1451
接收seq:1451
checksum:56423
发送ack:1452
接收seq:1452
checksum:44959
发送ack:1453

D:\MyCode\Computer_Network
base:1428
next_seq:1445
top:1446
发送seq:1445
checksum:68472
ssthresh:9
swnd:18
base:1428
next_seq:1446
top:1446
接收第1个期望ack:1428
发送seq:1446
checksum:6265
ssthresh:9
swnd:19
base:1428
next_seq:1447
top:1447
接收第2个期望ack:1428
发送seq:1447
checksum:38255
ssthresh:9
swnd:20
base:1428
next_seq:1448
top:1448
接收第3个期望ack:1428
休眠中
发送seq:1428
checksum:34819
ssthresh:9
休眠中
ssthresh:18
swnd:13
base:1428
next_seq:1448
top:1441
接收新的ack:1448
期望接收:
ssthresh:18
swnd:14
base:1448
next_seq:1448
top:1462
发送seq:1448
checksum:62803
ssthresh:18
swnd:14
base:1448

```

服务器端接收到 seq 为 1428 的数据包，返回 ack1448，表明 seq 为 1448 之前的数据包都已经成功接收，并准备好了下一个期望收到的数据包的 seq 为 1448。

设置 3% 的丢包率和 2ms 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)	71.13	342.915	725.835	79.2
吞吐率 (bytes/s)	26112.1	17201.1	16490	20906.7

断开连接：

