



南開大學
Nankai University

计算机学院
计算机网络实验报告

Lab3-2 : 基于 UDP 服务设计可靠传输
协议并编程实现

姓名：何禹姗
学号：2211421
专业：计算机科学与技术

2024 年 11 月 29 日

目录

1 实验要求	2
2 实验知识	2
2.1 滑动窗口	2
2.2 GBN (回退 N 重传协议)	2
3 实验设计	2
3.1 数据报套接字 UDP	2
3.2 协议设计	2
3.3 差错检验	3
3.4 建立连接	4
3.5 断开连接	10
3.6 超时重传	16
3.7 数据传输	17
4 运行结果	24
4.1 连接建立	24
4.2 数据传输	24
4.3 断开连接	27

1 实验要求

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，发送窗口和接收窗口采用相同大小，支持累积确认，使用 GBN 算法，完成给定测试文件的传输。

2 实验知识

2.1 滑动窗口

滑动窗口是一种在数据流处理、网络通信以及算法设计中广泛使用的技术。它主要用来处理连续的数据流或序列，通过一个固定大小的窗口来查看或处理数据的一部分，然后根据需要向前移动这个窗口。利用滑动窗口可以提高数据传输的效率。

在基于 UDP 服务设计可靠传输协议实现中，建立滑动窗口意味着在主机 A 未收到主机 B 发来的确认时，可将窗口内的所有数据包全部发送出去，主机 B 对主机 A 进行累积确认并进行流控。主机 A 每收到一个 ACK 就会将窗口向后滑动，同时将窗口内未发生的数据包全部发送。如果主机 A 发送给主机 B 的数据包丢失，则会从丢失的数据包开始将窗口内的所有数据包发送。

2.2 GBN (回退 N 重传协议)

在 GBN 协议下，发送方可以同时发送多个报文段，这些报文段都有一个特定的序号，接收方必须按照序号顺序接收这些报文。如果发送方发送了一批报文，但是其中有部分报文在传输过程中丢失，那么接收方将只会收到一些连续而又正确的数据包，而接收不到中间丢失的数据包和后面序列号不正确的数据包。当发送超时，发送方就会重发所有未确认数据包（即还没有收到对应的 ACK 确认报文），重发错误数据包以及其后所有在窗口内的数据包。

接收端只允许按顺序接收数据包，为了减少开销，累积确认允许在连续收到好几个正确的数据包后，只对最后一个数据包发送确认信息，所以当发送端收到接收端发回的某一数据包的确认信息时，表明该数据包及其之前的所有数据包都准确无误的收到了。

GBN 协议的接收窗口为 1，可以保证按序接收数据包。若采用 n 个比特对数据包编号，发送窗口的尺寸应满足小于 $2^{(n-1)}$ 。

3 实验设计

3.1 数据报套接字 UDP

用户数据报协议是一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务。

3.2 协议设计

报文格式每个 UDP 报文分为 UDP 报头和 UDP 数据区两部分。报头由 4 个 16 位长（2 字节）字段组成，分别说明该报文的源端口、目的端口、报文长度和校验值。



图 3.1: 报文格式

在我的程序中，我的数据报由两部分组成，第一部分是 Packethead，即数据报头，里面包含 16 位的序列号，校验和，数据部分总长度，确认号和标志位。第二部分是数据报 Packet，里面包含数据报头和所携带的数据部分，长度上限为 2048 字节。

```

1  //数据报头
2  struct Packethead {
3      uint16_t seq;      // 序列号 16 位
4      uint16_t Check;    // 校验 16 位
5      uint16_t len;      // 数据部分总长度
6      uint16_t ack;      // 确认号
7      unsigned char flags; // 标志位
8
9      Packethead() : seq(0), Check(0), len(0), flags(0), ack(0) {}
10 };
11 //数据报
12 struct Packet {
13     Packethead head;
14     char data[2048]; // 数据部分
15
16     Packet() : head(), data() {}
17 };

```

3.3 差错检验

为了判断传输的数据是否正确，我们利用校验和进行差错检验。

UDP 校验和的计算方法是：

(1) 按每 16 位求和得出一个 32 位的数，如果这个 32 位的数，高 16 位不为 0，则高 16 位加低 16 位再得到一个 32 位的数；即如果结果超过 16 位，则将最高位的值加到最低位上，形成一个新的 16 位数。

(2) 重复上述步骤 (1) 直到高 16 位为 0, 将低 16 位取反, 得到校验和。如果最终结果为全 1 (即 0xFFFF), 则校验和为 0; 否则, 取反最终结果得到校验和。

将计算出的校验和值放入 UDP 头部的校验和字段中。

```
1 //差错检测
2 u_short packetcheck(u_short* packet, int packlength)
3 {
4     register u_long sum = 0;
5     int count = (packlength + 1) / 2; //两个字节的计算
6     u_short* buf = new u_short[packlength + 1];
7     memset(buf, 0, packlength + 1);
8     memcpy(buf, packet, packlength);
9     while (count--)
10    {
11        sum += *buf++;
12        if (sum & 0xFFFF0000)
13        {
14            sum &= 0xFFFF;
15            sum++;
16        }
17    }
18    return ~(sum & 0xFFFF);
19 }
```

3.4 建立连接

在程序中我基于 TCP 的三次握手协议建立连接。

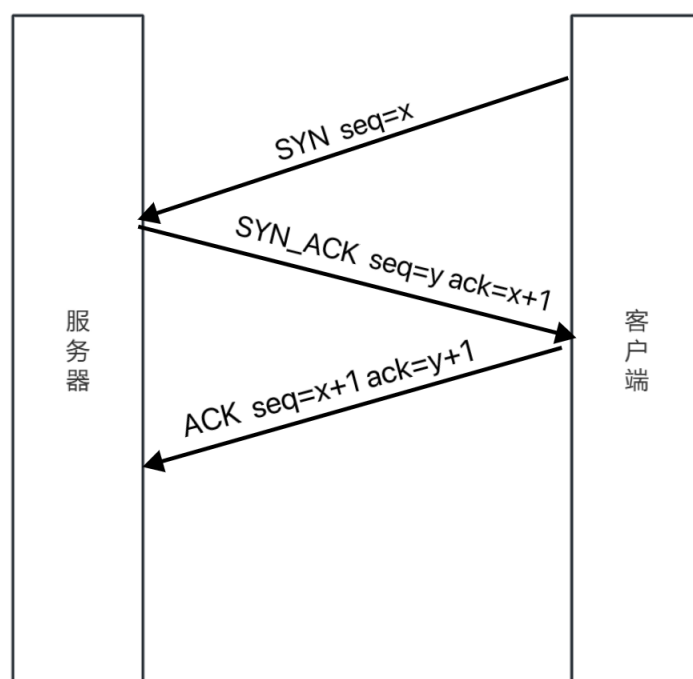


图 3.2: 三次握手示意图

第一次握手客户端向服务器发送数据报头标志位为 SYN 的数据包，表示想与服务器建立连接。(序列号 $seq=x$)

第二次握手服务器端向客户端发送数据报头标志位为 SYN_ACK 的数据包。(序列号 $seq=y$ ，确认号 $ack=x+1$)

标志位为 SYN（即第一次握手）的数据包发送至服务器端，由服务器端判断数据报序列号，标志位，校验和是否正确。如果正确，服务器端接收，且发送 SYN_ACK（即第二次握手）回客户端；如果不正确，服务器端不接收，客户端一段时间后未收到由服务器端发送的 SYN_ACK（即第二次握手），则重发标志位为 SYN（即第一次握手）的数据包。

同理，若数据包发送过程中被丢包，服务器端也无法顺利接收标志位为 SYN（即第一次握手）的数据包，此时客户端一段时间后未收到由服务器端发送的 SYN_ACK（即第二次握手），则重发标志位为 SYN（即第一次握手）的数据包。

第三次握手客户端向服务端发送标志位为 ACK 的数据包，与第一次握手同理，服务器端判断数据报序列号，标志位，校验和是否正确，如果正确服务器端则顺利接收数据包，此时握手完成，连接建立。(序列号 $seq=x+1$ ， $ack=y+1$)

客户端建立连接代码如下：

```

1  int clientHandshake(SOCKET s, sockaddr_in& clientAddr, int& sockLen) {
2  // 设置为非阻塞模式，避免卡在 recvfrom
3  u_long iMode = 1; // 0: 阻塞
4  ioctlsocket(s, FIONBIO, &iMode); // 非阻塞设置
5
6  Packet packet1;
  
```

```
7
8 // 发送 SYN 包
9 packet1.head.seq = 0;
10 packet1.head.flags = FLAG_SYN;
11 packet1.head.Check = packetcheck((u_short*)&packet1, sizeof(packet1));
12 int flag1 = 0;
13
14 //发送缓冲区
15 char* buffer1 = new char[sizeof(packet1)];
16 memcpy(buffer1, &packet1, sizeof(packet1));
17 flag1 = sendto(s, buffer1, sizeof(packet1), 0, (sockaddr*)&clientAddr,
18 sockLen);
19
20 if (flag1 == -1) { // 发送失败
21     std::cout << "[Client]: Failed to send SYN packet." << std::endl;
22     return 0;
23 }
24 clock_t start = clock(); //记录发送第一次握手时间
25 std::cout << "[Client]: SYN packet sent successfully." << std::endl;
26
27 // 等待 SYN_ACK 包
28 Packet packet2;
29
30 //缓冲区
31 char* buffer2 = new char[sizeof(packet2)];
32
33 bool synAckReceived = false;
34
35 while (recvfrom(s, buffer2, sizeof(packet2), 0, (sockaddr*)&clientAddr,
36 &sockLen)<=0) {
37     if (clock() - start > MAX_TIME) //超时, 重新传输第一次握手
38     {
39         std::cout << "[Client]:Timeout Retransmission,resending SYN
40 packet..." << std::endl;
41         sendto(s, buffer1, sizeof(packet1), 0, (sockaddr*)&clientAddr,
42 sockLen);
43         start = clock();
44     }
45 }
46 memcpy(&packet2, buffer2, sizeof(packet2));
47 u_short res = packetcheck((u_short*)&packet2, sizeof(packet2));
48 if (packet2.head.flags == FLAG_SYN_ACK && packet2.head.seq == 1 && res ==
```

```
49 0) {
50     std::cout << "[Client]: Received SYN_ACK packet with seq=" <<
51     packet2.head.seq << std::endl;
52     synAckReceived = true;
53 }
54 else {
55     std::cout << "[Client]: Failed to receive SYN_ACK packet." <<
56     std::endl;
57     return 0;
58 }
59 start = clock();
60
61 // 发送 ACK 包
62 Packet packet3;
63
64 packet3.head.seq = 2;
65 packet3.head.flags = FLAG_ACK;
66 packet3.head.Check = packetcheck((u_short*)&packet3, sizeof(packet3));
67 int flag3 = 0;
68
69 //发送缓冲区
70 char* buffer3 = new char[sizeof(packet3)];
71 memcpy(buffer3, &packet3, sizeof(packet3));
72
73 bool ackSentSuccessfully = false;
74
75 flag3 = sendto(s, buffer3, sizeof(packet3), 0, (sockaddr*)&clientAddr,
76 sockLen);
77
78 if (flag3 == -1) { // 发送失败
79     std::cout << "[Client]: Failed to send ACK packet." << std::endl;
80     return 0;
81 }
82
83 std::cout << "[Client]: ACK packet sent successfully." << std::endl;
84
85 iMode = 0; //0: 阻塞
86 ioctlsocket(s, FIONBIO, &iMode); //恢复成阻塞模式
87 return 1;
88 }
```

服务器端建立连接代码如下:


```

1  int serverHandshake(SOCKET s, sockaddr_in& ServerAddr, int& sockLen) {
2  //设置为非阻塞模式, 避免卡在 recvfrom
3  u_long iMode = 1; //0: 阻塞
4  ioctlsocket(s, FIONBIO, &iMode); //非阻塞设置
5
6  Packet packet1; //储存接收到的数据包
7  int flag1 = 0;
8
9  bool synReceived = false; //标记是否收到 SYN 包
10
11  char* buffer1 = new char[sizeof(packet1)];
12  // 等待 SYN 包
13  while (1) {
14
15      flag1 = recvfrom(s, buffer1, sizeof(packet1), 0,
16                      (sockaddr*)&ServerAddr, &sockLen);
17      //没收到就一直循环
18      if (flag1 <= 0)
19      {
20          //cout << "error" << endl;
21          continue;
22      }
23      //如果接收成功
24      memcpy(&(packet1), buffer1, sizeof(packet1.head));
25      u_short res = packetcheck((u_short*)&packet1, sizeof(packet1));
26      //cout << "success" << endl;
27      //检查接收到的是否为 SYN 包
28      //标志位为 FLAG_SYN 且序列号为 1, 且数据包正确
29      if (packet1.head.flags == FLAG_SYN && packet1.head.seq == 0 && res ==
30          0) {
31          std::cout << "[Server]: Received SYN packet with seq=" <<
32              packet1.head.seq << std::endl;
33          synReceived = true;
34          break;
35      }
36  }
37
38  //如果未收到 SYN 包, 输出一条消息表示失败
39  if (!synReceived) {
40      std::cout << "[Server]: Failed to receive SYN packet." << std::endl;
41      //return 0;

```

```
42 }
43
44 // 发送 SYN_ACK 包
45 Packet packet2;
46 packet2.head.seq = 1;
47 packet2.head.flags = FLAG_SYN_ACK;
48 packet2.head.Check = packetcheck((u_short*)&packet2, sizeof(packet2));
49
50 char* buffer2 = new char[sizeof(packet2)];
51 memcpy(buffer2, &packet2, sizeof(packet2));
52
53 int flag2 = sendto(s, buffer2, sizeof(packet2), 0,
54 (sockaddr*)&ServerAddr, sockLen);
55 if (flag2 == -1) { // 发送失败
56     std::cout << "[Server]: Failed to send SYN_ACK packet." << std::endl;
57     return 0;
58 }
59 clock_t start = clock(); // 记录第二次握手发送时间
60 std::cout << "[Server]: SYN_ACK packet sent successfully." << std::endl;
61
62
63 bool ackReceived = false; // 标记是否收到 ACK 包
64
65 // 等待 ACK 包
66 Packet packet3;
67 char* buffer3 = new char[sizeof(packet3)];
68 int flag3 = 0;
69
70 while (recvfrom(s, buffer3, sizeof(packet3), 0, (sockaddr*)&ServerAddr,
71 &sockLen) <= 0) {
72     if (clock() - start > MAX_TIME) // 超时, 重新传输第一次握手
73     {
74         return 1;
75         /*std::cout << "[Server]: Timeout Retransmission, resending SYN_ACK
76 packet..." << std::endl;
77         sendto(s, buffer2, sizeof(packet2), 0, (sockaddr*)&ServerAddr,
78 sockLen);
79         start = clock();*/
80     }
81 }
82 // 如果接收成功
83 memcpy(&(packet3), buffer3, sizeof(packet3.head));
```

```

84  u_short res = packetcheck((u_short*)&packet3, sizeof(packet3));
85
86  if (packet3.head.flags == FLAG_ACK && packet3.head.seq == 2 && res == 0) {
87      std::cout << "[Server]: Received ACK packet with seq=" <<
88      packet3.head.seq << std::endl;
89  }
90
91  iMode = 0; //0: 阻塞
92  ioctlsocket(s, FIONBIO, &iMode); //恢复成阻塞模式
93  return 1;
94  }

```

3.5 断开连接

在程序中我基于 TCP 的四次挥手协议断开连接。

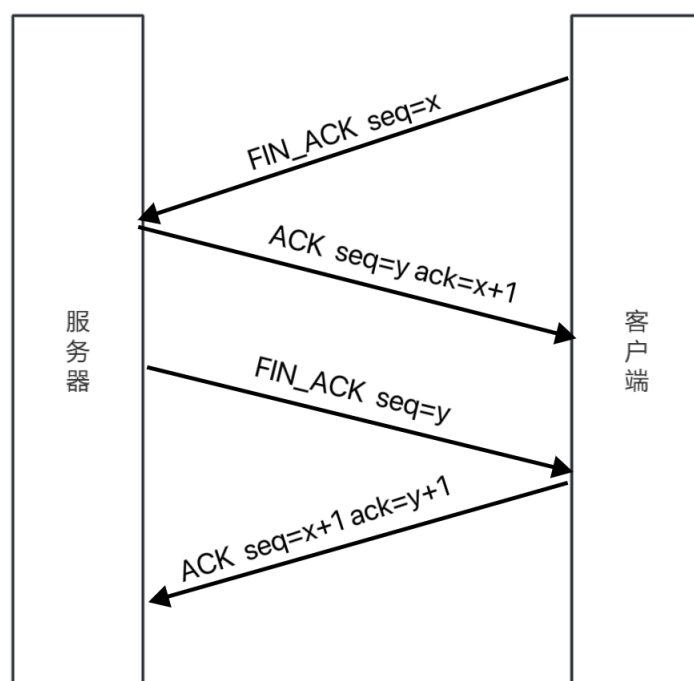


图 3.3: 四次挥手示意图

第一次挥手客户端向服务器发送数据报头标志位为 FIN_ACK 的数据包，表示想与服务器断开连接。(序列号 seq=x)

第二次挥手服务器端向客户端发送数据报头标志位为 ACK 的数据包。(序列号 seq=y，确认号 ack=x+1)

标志位为 FIN_ACK (即第一次挥手) 的数据包发送至服务器端，由服务器端判断数据报序列号，标志位，校验和是否正确。如果正确，服务器端接收，且发送 ACK (即第二次挥手) 回客户端；如果不正确，服务器端不接收，客户端一段时间后未收到由服务器端发送的 ACK (即第二次挥手)，则重

发标志位为 FIN_ACK（即第一次挥手）的数据包。

同理，若数据包发送过程中被丢包，服务器端也无法顺利接收标志位为 FIN_ACK（即第一次挥手）的数据包，此时客户端一段时间后未收到由服务器端发送的 ACK（即第二次挥手），则重发标志位为 FIN_ACK（即第一次挥手）的数据包。

第三、四次挥手第三、四次挥手与一、二次相同，只不过改为服务器端发送数据报头标志位为 FIN_ACK 的数据包，客户端发送数据报头标志位为 ACK 的数据包。（第三次挥手序列号 seq=y；第四次挥手序列号 seq=x+1，确认号 ack=y+1）

客户端断开连接代码如下：

```

1  int clientCloseConnection(SOCKET s, sockaddr_in& ClientAddr, int&
2  sockLen) {
3  // 设置为非阻塞模式，避免卡在 recvfrom
4  u_long iMode = 1; // 0: 阻塞
5  ioctlsocket(s, FIONBIO, &iMode); // 非阻塞设置
6
7  Packet packet1;
8  int flag1 = 0;
9
10 // 发送 FIN_ACK 包
11 packet1.head.seq = 0;
12 packet1.head.flags = FLAG_FIN_ACK;
13 packet1.head.Check = packetcheck((u_short*)&packet1, sizeof(packet1));
14 char* buffer1 = new char[sizeof(packet1)];
15 memcpy(buffer1, &packet1, sizeof(packet1));
16
17 flag1 = sendto(s, buffer1, sizeof(packet1), 0, (sockaddr*)&ClientAddr,
18 sockLen);
19 if (flag1 == -1) {
20     std::cout << "[Client]: Failed to send FIN_ACK packet." << std::endl;
21     return 0;
22 }
23 clock_t start = clock();
24 std::cout << "[Client]: FIN_ACK packet sent successfully." << std::endl;
25
26 // 等待服务器发送的 ACK 包
27 Packet packet2;
28 char* buffer2 = new char[sizeof(packet2)];
29
30 bool ackReceived = false;
31
32 while (recvfrom(s, buffer2, sizeof(packet2), 0, (sockaddr*)&ClientAddr,
33 &sockLen)<=0) {

```

```

34     if (clock() - start > MAX_TIME)//超时, 重新传输第一次握手
35     {
36         std::cout << "[Client]:Timeout Retransmission,resending FIN_ACK
37         packet..." << std::endl;
38         sendto(s, buffer1, sizeof(packet1), 0, (sockaddr*)&ClientAddr,
39         sockLen);
40         start = clock();
41     }
42 }
43 memcpy(&packet2, buffer2, sizeof(packet2));
44 u_short res = packetcheck((u_short*)&packet2, sizeof(packet2));
45
46 if (packet2.head.flags == FLAG_ACK && packet2.head.seq == 1 && res == 0) {
47     std::cout << "[Client]: Received ACK packet with seq=" <<
48     packet2.head.seq << std::endl;
49     ackReceived = true;
50 }
51 else{
52     std::cout << "[Client]: Failed to receive ACK packet." << std::endl;
53     //return 0;
54 }
55
56 // 等待服务器发送的 FIN_ACK 包
57 bool finackReceived = false;
58 Packet packet3;
59 char* buffer3 = new char[sizeof(packet3)];
60
61 while(1) {
62     u_short res = packetcheck((u_short*)&packet3, sizeof(packet3));
63
64     if (recvfrom(s, buffer3, sizeof(packet3), 0, (sockaddr*)&ClientAddr,
65     &sockLen)<=0) {
66         //cout << " 未收到 FIN_ACK 包 " << endl;
67     }
68     memcpy(&packet3, buffer3, sizeof(packet3));
69     if (packet3.head.flags == FLAG_FIN_ACK && packet3.head.seq == 2 &&
70     res == 0) {
71         std::cout << "[Client]: Received FIN_ACK packet with seq=" <<
72         packet3.head.seq << std::endl;
73         finackReceived = true;
74         break;
75     }

```

```
76
77 }
78 start = clock();
79
80 if (!finackReceived) {
81     std::cout << "[Client]: Failed to receive FIN_ACK packet." <<
82     std::endl;
83     return 0;
84 }
85
86 // 发送 ACK 包
87 Packet packet4;
88 packet4.head.seq = 3;
89 packet4.head.flags = FLAG_ACK;
90 packet4.head.Check = packetcheck((u_short*)&packet4, sizeof(packet4));
91 char* buffer4 = new char[sizeof(packet4)];
92 memcpy(buffer4, &packet4, sizeof(packet4));
93 int flag4 = 0;
94
95 bool ackSentSuccessfully = false;
96
97 while (!ackSentSuccessfully) {
98     flag4 = sendto(s, buffer4, sizeof(packet4), 0,
99     (sockaddr*)&ClientAddr, sockLen);
100     if (flag4 != -1) { // 发送成功
101         std::cout << "[Client]: ACK packet sent successfully." <<
102         std::endl;
103         ackSentSuccessfully = true;
104     }
105     else {
106         std::cout << "[Client]: Failed to send ACK packet, retrying..."
107         << std::endl;
108         return 0;
109     }
110 }
111
112 iMode = 0; // 0: 阻塞
113 ioctlsocket(s, FIONBIO, &iMode); // 恢复成阻塞模式
114 return 1;
115 }
```

服务器端断开连接代码如下：

```
1  int serverCloseConnection(SOCKET s, sockaddr_in& ServerAddr, int&
2  sockLen) {
3  // 设置为非阻塞模式, 避免卡在 recvfrom
4  u_long iMode = 1; // 0: 阻塞
5  ioctlsocket(s, FIONBIO, &iMode); // 非阻塞设置
6
7  Packet packet1;
8  char* buffer1 = new char[sizeof(packet1)];
9  int flag = 0;
10
11 // 等待客户端发送的 FIN_ACK 包
12 bool finackReceived = false;
13
14 while(1) {
15     if (recvfrom(s, buffer1, sizeof(packet1), 0, (sockaddr*)&ServerAddr,
16         &sockLen) > 0) {
17         memcpy(&(packet1), buffer1, sizeof(packet1));
18         u_short res = packetcheck((u_short*)&packet1, sizeof(packet1));
19         if (packet1.head.flags == FLAG_FIN_ACK && packet1.head.seq == 0
20             && res == 0) {
21             std::cout << "[Server]: Received FIN_ACK packet with seq=" <<
22                 packet1.head.seq << std::endl;
23             finackReceived = true;
24             break;
25         }
26     }
27     else {
28         continue;
29     }
30 }
31
32 if (!finackReceived) {
33     std::cout << "[Server]: Failed to receive FIN_ACK packet." <<
34         std::endl;
35     return 0;
36 }
37
38 // 发送 ACK 包
39 Packet packet2;
40 packet2.head.seq = 1;
41 packet2.head.flags = FLAG_ACK;
```

```
42 packet2.head.Check = packetcheck((u_short*)&packet2, sizeof(packet2));
43 char* buffer2 = new char[sizeof(packet2)];
44 memcpy(buffer2, &packet2, sizeof(packet2));
45
46 flag = sendto(s, buffer2, sizeof(packet2), 0, (sockaddr*)&ServerAddr,
47 sockLen);
48 if (flag != -1) { // 发送成功
49     std::cout << "[Server]: ACK packet sent successfully." << std::endl;
50 }
51 else {
52     std::cout << "[Server]: Failed to send ACK packet." << std::endl;
53 }
54
55 // 发送 FIN_ACK 包
56 Packet packet3;
57 packet3.head.seq = 2;
58 packet3.head.flags = FLAG_FIN_ACK;
59 packet3.head.Check = packetcheck((u_short*)&packet3, sizeof(packet3));
60 char* buffer3 = new char[sizeof(packet3)];
61 memcpy(buffer3, &packet3, sizeof(packet3));
62
63 bool finackSentSuccessfully = false;
64
65 while (!finackSentSuccessfully) {
66     flag = sendto(s, buffer3, sizeof(packet3), 0, (sockaddr*)&ServerAddr,
67 sockLen);
68     if (flag != -1) { // 发送成功
69         std::cout << "[Server]: FIN_ACK packet sent successfully." <<
70 std::endl;
71         finackSentSuccessfully = true;
72     }
73     else {
74         std::cout << "[Server]: Failed to send FIN_ACK packet,
75 retrying..." << std::endl;
76         std::this_thread::sleep_for(std::chrono::seconds(1));
77     }
78 }
79 clock_t start = clock();
80
81 if (flag == -1) { // 发送失败
82     std::cout << "[Server]: Failed to send FIN_ACK packet." << std::endl;
83     return 0;
```



```

84 }
85
86 // 等待客户端发送的 ACK 包
87 Packet packet4;
88 char* buffer4 = new char[sizeof(packet4)];
89 int flag4 = 0;
90
91 while (recvfrom(s, buffer4, sizeof(packet4), 0, (sockaddr*)&ServerAddr,
92 &sockLen) <= 0) {
93     if (clock() - start > MAX_TIME)//超时
94     {
95         return 1;
96         /*std::cout << "[Server]:Timeout Retransmission,resending ACK
97 packet..." << std::endl;
98 sendto(s, buffer3, sizeof(packet3), 0, (sockaddr*)&ServerAddr,
99 sockLen);
100 start = clock();*/
101     }
102 }
103 //如果接收成功
104 memcpy(&(packet4), buffer4, sizeof(packet4.head));
105 u_short res = packetcheck((u_short*)&packet4, sizeof(packet4));
106
107 if (packet4.head.flags == FLAG_ACK && packet4.head.seq == 3 && res == 0) {
108     std::cout << "[Server]: Received ACK packet with seq=" <<
109     packet4.head.seq << std::endl;
110 }
111
112 iMode = 0; // 0: 阻塞
113 ioctlsocket(s, FIONBIO, &iMode); // 恢复成阻塞模式
114 return 1;
115 }

```

3.6 超时重传

本次实验基于滑动窗口设计了超时重传协议，每个数据都有一个唯一的序列号，用于标识数据包的顺序，接收方发送的确认信息（ACK）中包含确认号，表示已成功接收数据包的序列号，发送方为每个未确认的数据包设置一个超时计时器，如果在超时时间内没有收到确认消息，发送方会从该数据包开始将窗口内所有数据包重新发送。

3.7 数据传输

数据传输中发送端和接收端均采用 rdt3.0 和 GBN。面对出现差错的情况，接收端会重发正确的最后一个数据包来进行确认，此时发送方会重发出现差错的数据包以及其后在窗口内的所有数据包，即确认重传。对于数据包丢失的情况，接收端接收不到数据包即不会返回 ACK，设置一个超时计时器，此时发送端在超时时间内未收到对应的 ACK 即会被判为超时，此时发送方会重发丢失的数据包以及其后在窗口内的所有数据包。

对于发送方 client:

1. 维护一个发送窗口结构体，记录此时窗口起始 start 和窗口结束 end，设定窗口大小 size。

```

1  struct sendwindow {
2      int size;
3      int start;
4      int end;
5
6      sendwindow():size(20),start(-1),end(0){}
7  };

```

2. 发送窗口内的所有数据包，数据包序列号递增，发送完第一个包就开始计时。

3. 当收到从接收端发回的 ACK 时，检查判断接收到的数据包的正确性，如果正确且 $\text{int}(\text{packet2.head.seq}) \geq \text{window.start} \% 256$ 即接收回的 ACK 序列号大于窗口起始号，则说明现在窗口内的数据包存在被确认的，此时窗口向后移，移到未被确认的第一个数据包，同时将现在窗口内所有未发送的数据包发送，重新开始计时；如果接收到的 ACK 错误，或者接收到的 ACK 不大于窗口起始号，即收到的确认数据包已不在窗口内，则不操作，等待下一个 ACK 确认数据包发回。

4. 如果发送端在设置的超时时间内未收到 ACK 确认数据包，则将窗口内的所有数据包重传，并重新开始计时。

5. 当 window.end 等于 packagenum 的时候，说明窗口上限已到达发送的最大数据包，接下来窗口上限不再进行移动，只需要修改 window.start 即可，对 window.start 进行判断，如果其等于 packagenum-1 时，则说明已经发送完所有数据包（window.start 初始化为-1），即可以退出循环，否则继续循环发送数据包，直到发送完毕为止。

6. 为了提高效率和资源利用率设置了多线程，即将发送端 ACK 的接收设置单独线程，不影响主线程的发送。此时还需设置线程结束的条件，即当 window.start 等于 packagenum-1 时，ACK 接收线程结束，此时可以保证所有 ACK 都被接收到。因为维护了全局变量窗口的结构体，所以同时还需在 ACK 接收线程和主线程加了 `std::lock_guard<std::mutex>` 互斥锁，并在对象析构时自动释放该锁，可以确保在多线程环境下对共享资源的安全访问。

发送端代码 ACK 接收线程如下:

```

1  void handleAckReception(SOCKET& socketClient, SOCKADDR_IN& servAddr, int&
2  servAddrlen, int packagenum)
3  {
4      Packet packet2;
5      char* buffer2 = new char[sizeof(packet2)];

```

```

6  while (window.start < packagenum - 1)
7  {
8      if (recvfrom(socketClient, buffer2, sizeof(packet2), 0,
9          (sockaddr*)&servAddr, &servAddrlen) > 0)
10     {
11         memcpy(&packet2, buffer2, sizeof(packet2));
12         u_short check2 = packetcheck((u_short*)&packet2, sizeof(packet2));
13
14         std::lock_guard<std::mutex> lock(windowMutex); // 加锁
15         if (int(check2) != 0)
16         {
17             window.end = window.start + 1;
18
19             //std::lock_guard<std::mutex> guarderror(coutMutex);
20             cout << " 接收到错误的 ACK" << endl;
21             continue;
22         }
23         else
24         {
25             if (int(packet2.head.seq) >= window.start % 256)
26             {
27                 window.start = window.start + int(packet2.head.seq) -
28                 window.start % 256;
29
30                 std::lock_guard<std::mutex> guardack(coutMutex);
31                 cout << " 接收到 ACK, 确认数据包发送成功  Flag:" <<
32                 int(packet2.head.flags) << " SEQ:" <<
33                 int(packet2.head.seq) << " ACK:" << int(packet2.head.ack)
34                 << " CHECK:" << int(packet2.head.Check) << " 窗口起始: " <<
35                 window.start << " 窗口结束: " << window.end << " 窗口大小: " <
36                 < window.size << endl;
37
38                 //cout << " 窗口起始: " << window.start << " 窗口结束: " <<
39                 window.end << " 窗口大小: " << window.size << endl;
40                 //std::lock_guard<std::mutex> lock(ackMutex);
41                 //--ackCounter;
42                 //ackCv.notify_one(); // 每次成功处理一个 ACK, 通知一次
43             }
44             else if (window.start % 256 > 256 - window.size - 1 &&
45                 int(packet2.head.seq) < window.size)
46             {
47                 window.start = window.start + 256 - window.start % 256 +

```

```

48         int(packet2.head.seq);
49
50     }
51 }
52 //锁自动释放
53 }
54 }
55 delete[] buffer2;
56 }

```

发送端代码发送数据包线程如下:

```

1  void send(SOCKET& socketClient, SOCKADDR_IN& servAddr, int& servAddrLen,
2  char* message, int messagelen)
3  {
4      u_long mode = 1;
5      ioctlsocket(socketClient, FIONBIO, &mode);
6      int seqnum = 0;
7      int packagenum = (messagelen / MAXSIZE) + (messagelen % MAXSIZE != 0);
8      cout << packagenum << endl;
9      int len = 0;
10     //sendwindow window;
11     window.start = -1;
12     window.size = 20;
13     window.end = 0;
14     clock_t start=clock();
15
16     ackCounter = packagenum;
17     // 启动 ACK 接收线程
18     std::thread ackThread(handleAckReception, std::ref(socketClient),
19     std::ref(servAddr), std::ref(servAddrLen), std::ref(packagenum));
20     //ackThread.detach();
21
22     //std::lock_guard<std::mutex> lock(windowMutex);
23     while (window.start < packagenum - 1)
24     {
25         if (window.end - window.start <= window.size && window.end !=
26         packagenum)
27         {
28             seqnum = window.end % 256;
29             len = ((window.end == (packagenum - 1)) ? (messagelen -
30             ((packagenum - 1) * MAXSIZE)) : MAXSIZE);

```

```

31     Packet packet1;
32     char* buffer1 = new char[len + sizeof(packet1)];
33     packet1.head.len = len;
34     packet1.head.seq = seqnum; //序列号
35     packet1.head.Check = 0;
36     u_short check = packetcheck((u_short*)&packet1, sizeof(packet1)); //
37     计算校验和
38     packet1.head.Check = check;
39     packet1.head.flags = 0;
40     packet1.head.ack = 0;
41     memcpy(buffer1, &packet1, sizeof(packet1));
42     char* mes = message + window.end * MAXSIZE;
43     memcpy(buffer1 + sizeof(packet1), mes, len); //将数据复制到缓冲区的后部
44     分
45     sendto(socketClient, buffer1, len + sizeof(packet1), 0,
46     (sockaddr*)&servAddr, servAddrlen); //发送
47
48     std::lock_guard<std::mutex> guardsend(coutMutex);
49     cout << " 发送文件大小为 " << len << " bytes!" << " Flag:" <<
50     int(packet1.head.flags) << " SEQ:" << int(packet1.head.seq) << "
51     ACK:" << int(packet1.head.ack) << " CHECK:" <<
52     int(packet1.head.Check) << endl;
53     start = clock();
54     window.end++;
55 }
56 if (clock() - start > MAX_TIME)
57 {
58     window.end = window.start + 1;
59
60     //std::lock_guard<std::mutex> guardre(coutMutex);
61     cout << " 超时重传" << endl;
62 }
63 if (window.start >= packagenum - 1)
64 {
65     break;
66 }
67 else {
68     continue;
69 }
70 }
71 ackThread.join();
72

```

```

73 // 等待所有 ACK
74 /*std::unique_lock<std::mutex> lock(ackMutex);
75 std::this_thread::sleep_for(std::chrono::milliseconds(100));*/
76
77 //发送结束信息
78 Packet packet5;
79 char* Buffer5 = new char[sizeof(packet5)];
80 packet5.head.flags = OVER; //结束信息
81 packet5.head.Check = 0;
82 u_short temp = packetcheck((u_short*)&packet5, sizeof(packet5));
83 packet5.head.Check = temp;
84
85 memcpy(Buffer5, &packet5, sizeof(packet5));
86 sendto(socketClient, Buffer5, sizeof(packet5), 0, (sockaddr*)&servAddr,
87 servAddrlen);
88 //cout << " 发送文件完毕! " << endl;
89 start = clock();
90
91 mode = 0;
92 ioctlsocket(socketClient, FIONBIO, &mode); //改回阻塞模式
93
94 return;
95 }

```

对于接收方 server: 1. 声明一个变量 seq 记录希望得到的数据包序列号，对于接收到的数据包进行判断是否正确，如果收到的数据包序列号正确且校验和正确，则将此数据包写入文件缓冲区，发送 ack=seq+1 回发送端，同时 seq+1。

2. 如果收到的数据包校验和错误或序列号错误，那么丢弃此数据包且不发送 ACK，此时发送端收不到对应的 ACK 便会重传。

3. 直到收到发送端发送的 over 数据包表示文件传输结束，返回 ACK 确认，接收文件完毕。

4. 由于路由器有发送文件个数的限制，影响数据传输速率，我在服务器端设置了模拟丢包。

接收端代码如下：

```

1 int RecvMessage(SOCKET& sockServ, SOCKADDR_IN& ClientAddr, int&
2 ClientAddrLen, char* message)
3 {
4     u_long mode = 1;
5     ioctlsocket(sockServ, FIONBIO, &mode); // 非阻塞模式
6     int filesize = 0; //文件长度
7
8     int seq = 0; //序列号
9     int ack = 1;

```

```
10
11  srand(time(0)); // 初始化随机数生成器
12
13  while (1)
14  {
15      Packet packet1;
16      char* Buffer1 = new char[MAXSIZE + sizeof(packet1)];
17
18      while (recvfrom(sockServ, Buffer1, sizeof(packet1) + MAXSIZE, 0,
19          (sockaddr*)&ClientAddr, &ClientAddrLen) <= 0); //接收报文长度
20
21      memcpy(&packet1, Buffer1, sizeof(packet1));
22
23      // 模拟丢包, 假设丢包概率
24      if (rand() % 100 < 5) {
25          delete[] Buffer1;
26          continue;
27      }
28
29      //判断是否是结束
30      if (packet1.head.flags == OVER && packetcheck((u_short*)&packet1,
31          sizeof(packet1)) == 0)
32      {
33          cout << " 文件接收完毕" << endl;
34          break;
35      }
36      //处理数据报文
37      else if (packet1.head.flags == 0 &&
38          packetcheck((u_short*)&packet1, sizeof(packet1)) == 0)
39      {
40          //判断是否接受的是别的包
41          if (seq != int(packet1.head.seq))
42          {
43              //cout << "error" << endl;
44              continue; //丢弃该数据包
45          }
46
47          //cout << "seq: " << seq << "head.seq: " << packet1.head.seq
48          << endl;
49          //取出 buffer 中的内容
50          int curr_size = packet1.head.len;
51
```

```

52     cout << " 接收到文件大小为 " << curr_size << " bytes! Flag:" <<
53     int(packet1.head.flags) << " SEQ : " << int(packet1.head.seq)
54     << " ACK : " << int(packet1.head.ack) << " CHECK : " <<
55     int(packet1.head.Check) << endl;
56
57     memcpy(message + filesize, Buffer1 + sizeof(packet1),
58     curr_size);
59     //cout << "size" << sizeof(message) << endl;
60     filesize = filesize + curr_size;
61
62     //返回 ACK
63     Packet packet2;
64     char* Buffer2 = new char[sizeof(packet2)];
65
66     packet2.head.flags = FLAG_ACK;
67     //packet2.head.len = 0;
68     packet2.head.seq = seq++;
69     packet2.head.ack = ack++;
70     packet2.head.Check = 0;
71     packet2.head.Check = packetcheck((u_short*)&packet2,
72     sizeof(packet2));
73     memcpy(Buffer2, &packet2, sizeof(packet2));
74     //重发该包的 ACK
75     sendto(sockServ, Buffer2, sizeof(packet2), 0,
76     (sockaddr*)&ClientAddr, ClientAddrLen);
77
78     cout << " 发送 ACK, 确认数据包发送成功 Flag:" <<
79     int(packet2.head.flags) << " SEQ : " << int(packet2.head.seq)
80     << " ACK : " << int(packet2.head.ack) << " CHECK : " <<
81     int(packet2.head.Check) << endl;
82     if (seq > 255)
83     {
84         seq = seq - 256;
85     }
86     if (ack > 255)
87     {
88         ack = ack - 256;
89     }
90 }
91 else {
92     if (packetcheck((u_short*)&packet1, sizeof(packet1)) != 0) {
93         cout << "error" << endl;

```

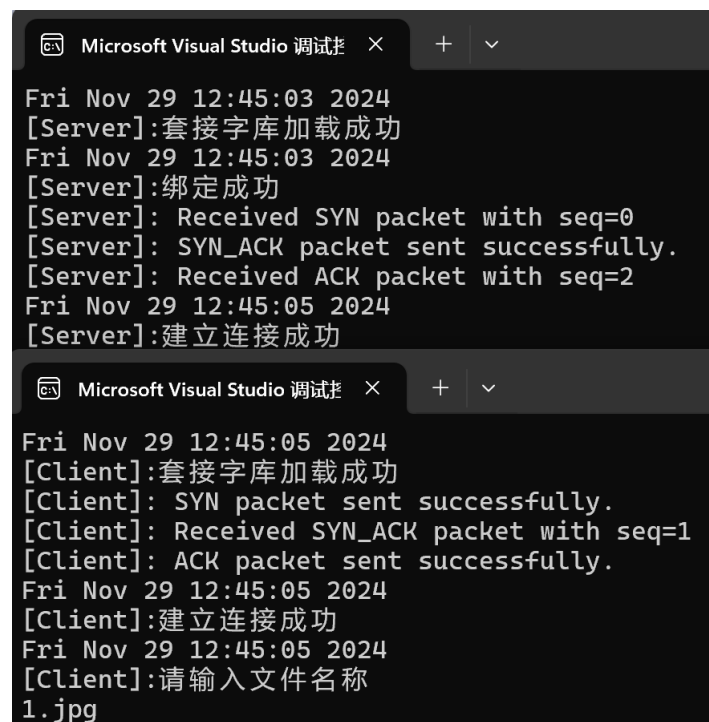


```
94         system("pause");
95     }
96     cout << " 错误" << endl;
97 }
98 }
99
100 mode = 0;
101 ioctlsocket(sockServ, FIONBIO, &mode); // 阻塞模式
102
103 return filesize;
104 }
```

4 运行结果

4.1 连接建立

经过三次握手连接建立运行结果如下：



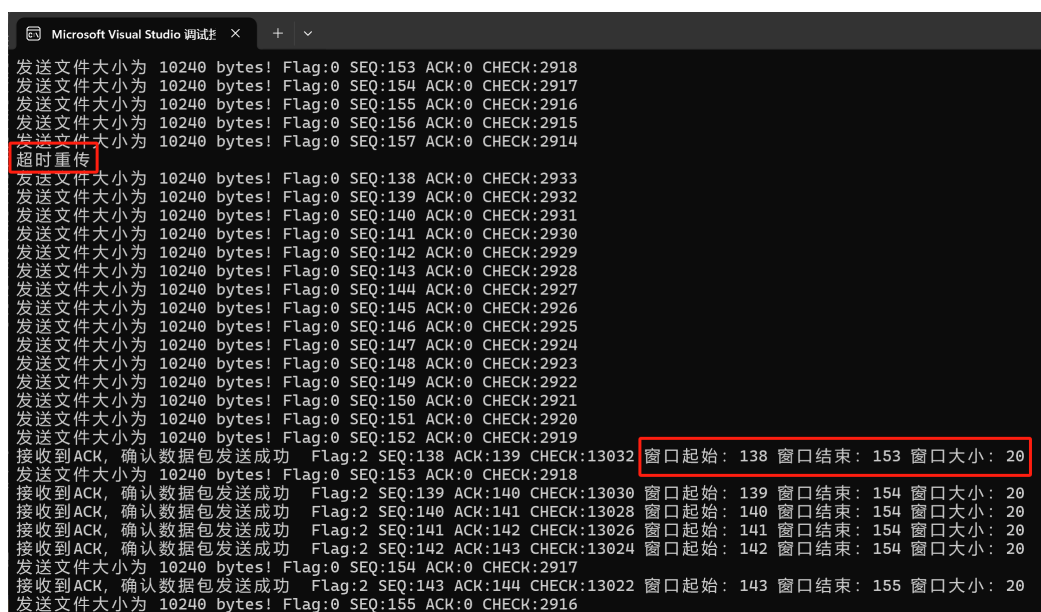
```
Microsoft Visual Studio 调试  x + v
Fri Nov 29 12:45:03 2024
[Server]:套接字库加载成功
Fri Nov 29 12:45:03 2024
[Server]:绑定成功
[Server]: Received SYN packet with seq=0
[Server]: SYN_ACK packet sent successfully.
[Server]: Received ACK packet with seq=2
Fri Nov 29 12:45:05 2024
[Server]:建立连接成功

Microsoft Visual Studio 调试  x + v
Fri Nov 29 12:45:05 2024
[Client]:套接字库加载成功
[Client]: SYN packet sent successfully.
[Client]: Received SYN_ACK packet with seq=1
[Client]: ACK packet sent successfully.
Fri Nov 29 12:45:05 2024
[Client]:建立连接成功
Fri Nov 29 12:45:05 2024
[Client]:请输入文件名称
1.jpg
```

图 4.4: 连接建立

4.2 数据传输

客户端发送数据结果如下：



```
Microsoft Visual Studio 调试
发送文件大小为 10240 bytes! Flag:0 SEQ:153 ACK:0 CHECK:2918
发送文件大小为 10240 bytes! Flag:0 SEQ:154 ACK:0 CHECK:2917
发送文件大小为 10240 bytes! Flag:0 SEQ:155 ACK:0 CHECK:2916
发送文件大小为 10240 bytes! Flag:0 SEQ:156 ACK:0 CHECK:2915
发送文件大小为 10240 bytes! Flag:0 SEQ:157 ACK:0 CHECK:2914
超时重传
发送文件大小为 10240 bytes! Flag:0 SEQ:138 ACK:0 CHECK:2933
发送文件大小为 10240 bytes! Flag:0 SEQ:139 ACK:0 CHECK:2932
发送文件大小为 10240 bytes! Flag:0 SEQ:140 ACK:0 CHECK:2931
发送文件大小为 10240 bytes! Flag:0 SEQ:141 ACK:0 CHECK:2930
发送文件大小为 10240 bytes! Flag:0 SEQ:142 ACK:0 CHECK:2929
发送文件大小为 10240 bytes! Flag:0 SEQ:143 ACK:0 CHECK:2928
发送文件大小为 10240 bytes! Flag:0 SEQ:144 ACK:0 CHECK:2927
发送文件大小为 10240 bytes! Flag:0 SEQ:145 ACK:0 CHECK:2926
发送文件大小为 10240 bytes! Flag:0 SEQ:146 ACK:0 CHECK:2925
发送文件大小为 10240 bytes! Flag:0 SEQ:147 ACK:0 CHECK:2924
发送文件大小为 10240 bytes! Flag:0 SEQ:148 ACK:0 CHECK:2923
发送文件大小为 10240 bytes! Flag:0 SEQ:149 ACK:0 CHECK:2922
发送文件大小为 10240 bytes! Flag:0 SEQ:150 ACK:0 CHECK:2921
发送文件大小为 10240 bytes! Flag:0 SEQ:151 ACK:0 CHECK:2920
发送文件大小为 10240 bytes! Flag:0 SEQ:152 ACK:0 CHECK:2919
接收到ACK, 确认数据包发送成功 Flag:2 SEQ:138 ACK:139 CHECK:13032 窗口起始: 138 窗口结束: 153 窗口大小: 20
发送文件大小为 10240 bytes! Flag:0 SEQ:153 ACK:0 CHECK:2918
接收到ACK, 确认数据包发送成功 Flag:2 SEQ:139 ACK:140 CHECK:13030 窗口起始: 139 窗口结束: 154 窗口大小: 20
接收到ACK, 确认数据包发送成功 Flag:2 SEQ:140 ACK:141 CHECK:13028 窗口起始: 140 窗口结束: 154 窗口大小: 20
接收到ACK, 确认数据包发送成功 Flag:2 SEQ:141 ACK:142 CHECK:13026 窗口起始: 141 窗口结束: 154 窗口大小: 20
接收到ACK, 确认数据包发送成功 Flag:2 SEQ:142 ACK:143 CHECK:13024 窗口起始: 142 窗口结束: 154 窗口大小: 20
发送文件大小为 10240 bytes! Flag:0 SEQ:154 ACK:0 CHECK:2917
接收到ACK, 确认数据包发送成功 Flag:2 SEQ:143 ACK:144 CHECK:13022 窗口起始: 143 窗口结束: 155 窗口大小: 20
发送文件大小为 10240 bytes! Flag:0 SEQ:155 ACK:0 CHECK:2916
```

图 4.5: 客户端发送数据

客户端发送数据运行输出发送文件大小, 标志位, 序列号, 确认号, 校验和; 接收到 ACK 输出标志位, 序列号, 确认号, 校验和进行比对, 确保发送的数据包正确, 同时输出窗口起始、窗口结束和窗口大小。发生丢包时, 显示“超时重传”, 将窗口内的所有数据包进行重传, 在运行结果中可以看出, 重传的数据包为此时窗口内的所有数据包, 序列号一致, 运行结果正确。

服务器端接收数据结果如下:

```
Microsoft Visual Studio 调试 × + v
接收到文件大小为 10240 bytes! Flag:0 SEQ : 171 ACK : 0 CHECK : 2900
发送ACK, 确认数据包发送成功 Flag:2 SEQ : 171 ACK : 172 CHECK : 12966
接收到文件大小为 10240 bytes! Flag:0 SEQ : 172 ACK : 0 CHECK : 2899
发送ACK, 确认数据包发送成功 Flag:2 SEQ : 172 ACK : 173 CHECK : 12964
接收到文件大小为 10240 bytes! Flag:0 SEQ : 173 ACK : 0 CHECK : 2898
发送ACK, 确认数据包发送成功 Flag:2 SEQ : 173 ACK : 174 CHECK : 12962
接收到文件大小为 10240 bytes! Flag:0 SEQ : 174 ACK : 0 CHECK : 2897
发送ACK, 确认数据包发送成功 Flag:2 SEQ : 174 ACK : 175 CHECK : 12960
接收到文件大小为 10240 bytes! Flag:0 SEQ : 175 ACK : 0 CHECK : 2896
发送ACK, 确认数据包发送成功 Flag:2 SEQ : 175 ACK : 176 CHECK : 12958
接收到文件大小为 10240 bytes! Flag:0 SEQ : 176 ACK : 0 CHECK : 2895
发送ACK, 确认数据包发送成功 Flag:2 SEQ : 176 ACK : 177 CHECK : 12956
接收到文件大小为 10240 bytes! Flag:0 SEQ : 177 ACK : 0 CHECK : 2894
发送ACK, 确认数据包发送成功 Flag:2 SEQ : 177 ACK : 178 CHECK : 12954
接收到文件大小为 10240 bytes! Flag:0 SEQ : 178 ACK : 0 CHECK : 2893
发送ACK, 确认数据包发送成功 Flag:2 SEQ : 178 ACK : 179 CHECK : 12952
接收到文件大小为 10240 bytes! Flag:0 SEQ : 179 ACK : 0 CHECK : 2892
发送ACK, 确认数据包发送成功 Flag:2 SEQ : 179 ACK : 180 CHECK : 12950
接收到文件大小为 10240 bytes! Flag:0 SEQ : 180 ACK : 0 CHECK : 2891
发送ACK, 确认数据包发送成功 Flag:2 SEQ : 180 ACK : 181 CHECK : 12948
接收到文件大小为 3913 bytes! Flag:0 SEQ : 181 ACK : 0 CHECK : 9217
发送ACK, 确认数据包发送成功 Flag:2 SEQ : 181 ACK : 182 CHECK : 12946
文件接收完毕
Fri Nov 29 12:45:15 2024
[Server]:文件名接收成功
Fri Nov 29 12:45:15 2024
[Server]:文件内容接收成功
```

图 4.6: 服务器端接收数据

服务器端接收数据运行输出接收到文件大小, 标志位, 序列号, 确认号, 校验和, 接收完毕后输出文件接收完毕。

接收文件结果如下:

名称	修改日期	类型	大小
x64	2024/11/21 16:27	文件夹	
1	2024/11/29 12:45	JPG 文件	1,814 KB
2	2024/11/29 0:41	JPG 文件	5,761 KB
3	2024/11/29 0:38	JPG 文件	11,689 KB
helloworld	2024/11/29 12:48	文本文档	1,617 KB
Lab3-2 server.cpp	2024/11/29 0:40	C++ Source	16 KB
Lab3-2 server.vcxproj	2024/11/21 16:27	VCXPROJ 文件	7 KB
Lab3-2 server.vcxproj.filters	2024/11/21 14:00	VC++ Project Filter...	1 KB
Lab3-2 server.vcxproj.user	2024/11/21 14:00	Per-User Project O...	1 KB

图 4.7: 本地接收文件结果

文件成功传输到服务器端, 并且文件大小一致。

丢包设置由于发现路由器有传输文件个数的限制, 会影响程序传输速率, 我在服务器端进行手动丢包。

```

1 // 模拟丢包，假设丢包概率
2 if (rand() % 100 < 5) {
3     delete[] Buffer1;
4     continue;
5 }

```

此时丢包率为 5%。

4.3 断开连接

经过四次挥手连接断开运行结果如下：

```

Microsoft Visual Studio 调试  x  +  v
Microsoft Visual Studio 调试  x  +  v
接收到文件大小为 10240 bytes! Flag:0 SEQ : 174  接收到ACK, 确认数据包发送成功  Flag:2 SEQ:169 ACK:170 CHECK:12970 窗口起始: 169 窗口结束: 182 窗口大小: 20
发送ACK, 确认数据包发送成功  Flag:2 SEQ : 174  接收到ACK, 确认数据包发送成功  Flag:2 SEQ:170 ACK:171 CHECK:12968 窗口起始: 170 窗口结束: 182 窗口大小: 20
接收到文件大小为 10240 bytes! Flag:0 SEQ : 175  接收到ACK, 确认数据包发送成功  Flag:2 SEQ:171 ACK:172 CHECK:12966 窗口起始: 171 窗口结束: 182 窗口大小: 20
发送ACK, 确认数据包发送成功  Flag:2 SEQ : 175  接收到ACK, 确认数据包发送成功  Flag:2 SEQ:172 ACK:173 CHECK:12964 窗口起始: 172 窗口结束: 182 窗口大小: 20
接收到文件大小为 10240 bytes! Flag:0 SEQ : 176  接收到ACK, 确认数据包发送成功  Flag:2 SEQ:173 ACK:174 CHECK:12962 窗口起始: 173 窗口结束: 182 窗口大小: 20
发送ACK, 确认数据包发送成功  Flag:2 SEQ : 176  接收到ACK, 确认数据包发送成功  Flag:2 SEQ:174 ACK:175 CHECK:12960 窗口起始: 174 窗口结束: 182 窗口大小: 20
接收到文件大小为 10240 bytes! Flag:0 SEQ : 177  接收到ACK, 确认数据包发送成功  Flag:2 SEQ:175 ACK:176 CHECK:12958 窗口起始: 175 窗口结束: 182 窗口大小: 20
发送ACK, 确认数据包发送成功  Flag:2 SEQ : 177  接收到ACK, 确认数据包发送成功  Flag:2 SEQ:176 ACK:177 CHECK:12956 窗口起始: 176 窗口结束: 182 窗口大小: 20
接收到文件大小为 10240 bytes! Flag:0 SEQ : 178  接收到ACK, 确认数据包发送成功  Flag:2 SEQ:177 ACK:178 CHECK:12954 窗口起始: 177 窗口结束: 182 窗口大小: 20
发送ACK, 确认数据包发送成功  Flag:2 SEQ : 178  超时重传
接收到文件大小为 10240 bytes! Flag:0 SEQ : 179  发送文件大小为 10240 bytes! Flag:0 SEQ:178 ACK:0 CHECK:2893
发送ACK, 确认数据包发送成功  Flag:2 SEQ : 179  发送文件大小为 10240 bytes! Flag:0 SEQ:179 ACK:0 CHECK:2892
接收到文件大小为 10240 bytes! Flag:0 SEQ : 180  发送文件大小为 10240 bytes! Flag:0 SEQ:180 ACK:0 CHECK:2891
发送ACK, 确认数据包发送成功  Flag:2 SEQ : 180  发送文件大小为 3913 bytes! Flag:0 SEQ:181 ACK:0 CHECK:9217
接收到文件大小为 3913 bytes! Flag:0 SEQ : 181  接收到ACK, 确认数据包发送成功  Flag:2 SEQ:178 ACK:179 CHECK:12952 窗口起始: 178 窗口结束: 182 窗口大小: 20
发送ACK, 确认数据包发送成功  Flag:2 SEQ : 181  接收到ACK, 确认数据包发送成功  Flag:2 SEQ:179 ACK:180 CHECK:12950 窗口起始: 179 窗口结束: 182 窗口大小: 20
文件接收完毕  接收到ACK, 确认数据包发送成功  Flag:2 SEQ:180 ACK:181 CHECK:12948 窗口起始: 180 窗口结束: 182 窗口大小: 20
Fri Nov 29 12:45:15 2024  接收到ACK, 确认数据包发送成功  Flag:2 SEQ:181 ACK:182 CHECK:12946 窗口起始: 181 窗口结束: 182 窗口大小: 20
[Server]:文件名接收成功  Fri Nov 29 12:45:15 2024  [Client]:文件内容传输成功
[Server]:文件内容接收成功  传输总时间为:5s
[Server]: Received FIN_ACK packet with seq=0  吞吐率为:371471 byte/s
[Server]: ACK packet sent successfully.  [Client]: Received ACK packet with seq=1
[Server]: FIN_ACK packet sent successfully.  [Client]: Received FIN_ACK packet with seq=2
[Server]: Received ACK packet with seq=3  [Client]: ACK packet sent successfully.
Fri Nov 29 12:45:15 2024  Fri Nov 29 12:45:15 2024
[Server]:断开连接成功  [System]:断开连接成功
F:\大学\课程\大三上\计算机网络\实验\实验3-2\Lab3-2 client\x64\Debug\Lab3-2 client.exe (进程 24660)已退出, 代码
按任意键关闭此窗口. . .

```

图 4.8: 断开连接