计算机学院

计算机网络实验报告

Lab3-1：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 20 日

目录

1 实验要求

利用数据报套接字在用户空间实现面向连接的可靠数据传输，功能包括：建立连接、差错检测、接收确认、超时重传等。流量控制采用停等机制，完成给定测试文件的传输。

2 实验设计

2.1 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。

2.2 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。

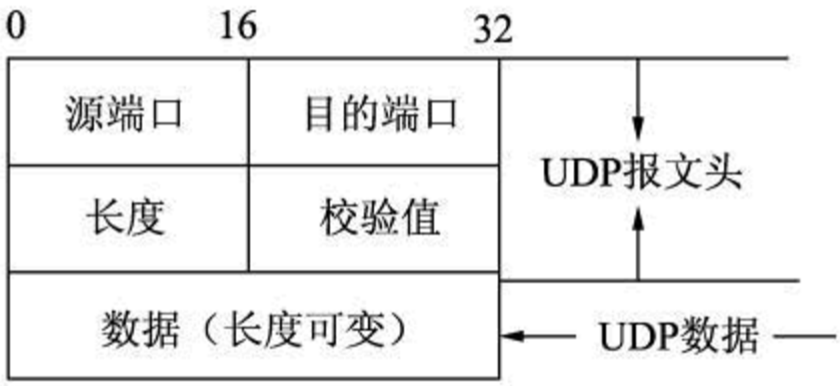


图 2.1: 报文格式

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。

```
//数据报头
struct Packethead {
    uint16_t seq;      // 序列号 16 位
    uint16_t Check;    // 校验 16 位
    uint16_t len;      // 数据部分总长度
    uint16_t ack;      //确认号
    unsigned char flags; // 标志位

    Packethead() : seq(0), Check(0), len(0), flags(0), ack(0) {}
};
//数据报
```

```
struct Packet {
    Packethead head;
    char data[2048]; // 数据部分

    Packet() : head(), data() {}
};
```

2.3 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。

```
//差错检测
u_short packetcheck(u_short* packet, int packelength)
{
    register u_long sum = 0;
    int count = (packelength + 1) / 2;//两个字节的计算
    u_short* buf = new u_short[packelength + 1];
    memset(buf, 0, packelength + 1);
    memcpy(buf, packet, packelength);
    while (count--)
    {
        sum += *buf++;
        if (sum & 0xFFFF0000)
        {
            sum &= 0xFFFF;
            sum++;
        }
    }
    return ~(sum & 0xFFFF);
}
```

2.4 建立连接

在程序中我基于 TCP 的三次握手协议建立连接。

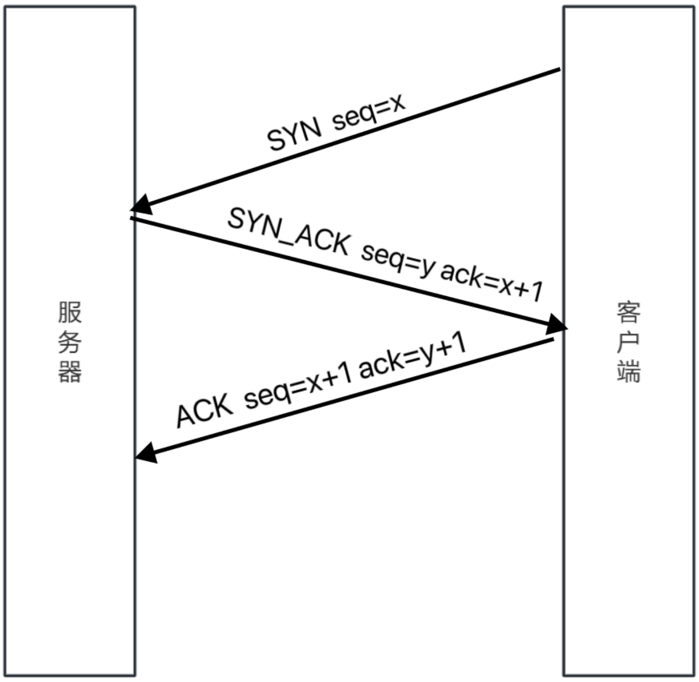


图 2.2: 三次握手示意图

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：

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

Packet packet1;
```

```

// 发送 SYN 包
packet1.head.seq = 0;
packet1.head.flags = FLAG_SYN;
packet1.head.Check = packetcheck((u_short*)&packet1, sizeof(packet1));
int flag1 = 0;

//发送缓冲区
char* buffer1 = new char[sizeof(packet1)];
memcpy(buffer1, &packet1, sizeof(packet1));
flag1 = sendto(s, buffer1, sizeof(packet1), 0, (sockaddr*)&clientAddr,
sockLen);

if (flag1 == -1) { // 发送失败
    std::cout << "[Client]: Failed to send SYN packet." << std::endl;
    return 0;
}
clock_t start = clock(); //记录发送第一次握手时间
std::cout << "[Client]: SYN packet sent successfully." << std::endl;

// 等待 SYN_ACK 包
Packet packet2;

//缓冲区
char* buffer2 = new char[sizeof(packet2)];

bool synAckReceived = false;

while (recvfrom(s, buffer2, sizeof(packet2), 0, (sockaddr*)&clientAddr,
&sockLen)<=0) {
    if (clock() - start > MAX_TIME)//超时，重新传输第一次握手
    {
        std::cout << "[Client]:Timeout Retransmission,resending SYN
        packet..." << std::endl;
        sendto(s, buffer1, sizeof(packet1), 0, (sockaddr*)&clientAddr,
        sockLen);
        start = clock();
    }
}
memcpy(&packet2, buffer2, sizeof(packet2));
u_short res = packetcheck((u_short*)&packet2, sizeof(packet2));
if (packet2.head.flags == FLAG_SYN_ACK && packet2.head.seq == 1 && res ==
0) {
    std::cout << "[Client]: Received SYN_ACK packet with seq=" <<
    packet2.head.seq << std::endl;
    synAckReceived = true;
}
else {
    std::cout << "[Client]: Failed to receive SYN_ACK packet." <<
    std::endl;
    return 0;
}
start = clock();

// 发送 ACK 包
Packet packet3;

packet3.head.seq = 2;
packet3.head.flags = FLAG_ACK;
packet3.head.Check = packetcheck((u_short*)&packet3, sizeof(packet3));
int flag3 = 0;

//发送缓冲区
char* buffer3 = new char[sizeof(packet3)];
memcpy(buffer3, &packet3, sizeof(packet3));

bool ackSentSuccessfully = false;

flag3 = sendto(s, buffer3, sizeof(packet3), 0, (sockaddr*)&clientAddr,
sockLen);

if (flag3 == -1) { // 发送失败
    std::cout << "[Client]: Failed to send ACK packet." << std::endl;
    return 0;
}

std::cout << "[Client]: ACK packet sent successfully." << std::endl;

iMode = 0; //0: 阻塞
ioctlsocket(s, FIONBIO, &iMode);//恢复成阻塞模式
return 1;
}
```

服务器端建立连接代码如下：

```
int serverHandshake(SOCKET s, sockaddr_in& ServerAddr, int& sockLen) {
//设置为非阻塞模式，避免卡在 recvfrom
u_long iMode = 1; //0: 阻塞
ioctlsocket(s, FIONBIO, &iMode);//非阻塞设置

Packet packet1;//储存接收到的数据包
int flag1 = 0;

bool synReceived = false;//标记是否收到 SYN 包

char* buffer1 = new char[sizeof(packet1)];
// 等待 SYN 包
while (1) {

    flag1 = recvfrom(s, buffer1, sizeof(packet1), 0,
    (sockaddr*)&ServerAddr, &sockLen);
    //没收到就一直循环
    if (flag1 <= 0)
    {
        //cout << "error" << endl;
        continue;
    }
    //如果接收成功
    memcpy(&(packet1), buffer1, sizeof(packet1.head));
    u_short res = packetcheck((u_short*)&packet1, sizeof(packet1));
    //cout << "success" << endl;
    //检查接收到的是否为 SYN 包
    //标志位为 FLAG_SYN 且序列号为 1，且数据包正确
    if (packet1.head.flags == FLAG_SYN && packet1.head.seq == 0 && res ==
    0) {
        std::cout << "[Server]: Received SYN packet with seq=" <<
        packet1.head.seq << std::endl;
        synReceived = true;
        break;
    }
}

//如果未收到 SYN 包，输出一条消息表示失败
if (!synReceived) {
    std::cout << "[Server]: Failed to receive SYN packet." << std::endl;
    //return 0;
}

// 发送 SYN_ACK 包
Packet packet2;
packet2.head.seq = 1;
packet2.head.flags = FLAG_SYN_ACK;
packet2.head.Check = packetcheck((u_short*)&packet2, sizeof(packet2));

char* buffer2 = new char[sizeof(packet2)];
memcpy(buffer2, &packet2, sizeof(packet2));

int flag2 = sendto(s, buffer2, sizeof(packet2), 0,
(sockaddr*)&ServerAddr, sockLen);
if (flag2 == -1) { // 发送失败
    std::cout << "[Server]: Failed to send SYN_ACK packet." << std::endl;
    return 0;
}
clock_t start = clock();//记录第二次握手发送时间
std::cout << "[Server]: SYN_ACK packet sent successfully." << std::endl;


bool ackReceived = false; // 标记是否收到 ACK 包

// 等待 ACK 包
Packet packet3;
char* buffer3 = new char[sizeof(packet3)];
int flag3 = 0;

while (recvfrom(s, buffer3, sizeof(packet3), 0, (sockaddr*)&ServerAddr,
&sockLen) <= 0) {
    if (clock() - start > MAX_TIME)//超时，重新传输第一次握手
    {
        return 1;
        /*std::cout << "[Server]:Timeout Retransmission,resending SYN_ACK
        packet..." << std::endl;
        sendto(s, buffer2, sizeof(packet2), 0, (sockaddr*)&ServerAddr,
        sockLen);
        start = clock();*/
    }
}
//如果接收成功
memcpy(&(packet3), buffer3, sizeof(packet3.head));
```

```
u_short res = packetcheck((u_short*)&packet3, sizeof(packet3));

if (packet3.head.flags == FLAG_ACK && packet3.head.seq == 2 && res == 0) {
    std::cout << "[Server]: Received ACK packet with seq=" <<
    packet3.head.seq << std::endl;
}

iMode = 0; //0: 阻塞
ioctlsocket(s, FIONBIO, &iMode);//恢复成阻塞模式
return 1;
}
```

2.5 断开连接

在程序中我基于 TCP 的四次挥手协议断开连接。

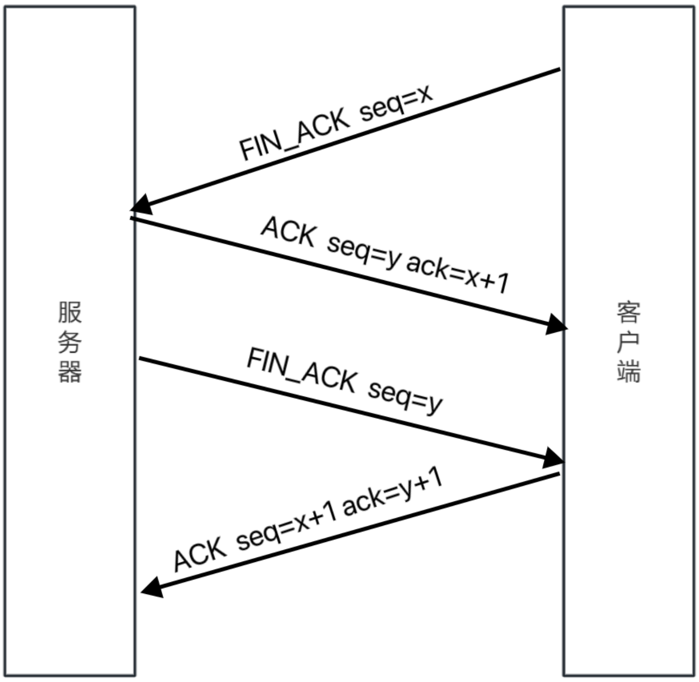


图 2.3: 四次挥手示意图

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：

```
int clientCloseConnection(SOCKET s, sockaddr_in& ClientAddr, int&
sockLen) {
// 设置为非阻塞模式，避免卡在 recvfrom
u_long iMode = 1; // 0: 阻塞
ioctlsocket(s, FIONBIO, &iMode); // 非阻塞设置

Packet packet1;
int flag1 = 0;

// 发送 FIN_ACK 包
packet1.head.seq = 0;
packet1.head.flags = FLAG_FIN_ACK;
packet1.head.Check = packetcheck((u_short*)&packet1, sizeof(packet1));
char* buffer1 = new char[sizeof(packet1)];
memcpy(buffer1, &packet1, sizeof(packet1));

flag1 = sendto(s, buffer1, sizeof(packet1), 0, (sockaddr*)&ClientAddr,
sockLen);
if (flag1 == -1) {
    std::cout << "[Client]: Failed to send FIN_ACK packet." << std::endl;
    return 0;
}
clock_t start = clock();
std::cout << "[Client]: FIN_ACK packet sent successfully." << std::endl;

// 等待服务器发送的 ACK 包
Packet packet2;
char* buffer2 = new char[sizeof(packet2)];

bool ackReceived = false;

while (recvfrom(s, buffer2, sizeof(packet2), 0, (sockaddr*)&ClientAddr,
&sockLen)<=0) {
        if (clock() - start > MAX_TIME)//超时，重新传输第一次握手
        {
            std::cout << "[Client]:Timeout Retransmission,resending FIN_ACK
            packet..." << std::endl;
            sendto(s, buffer1, sizeof(packet1), 0, (sockaddr*)&ClientAddr,
            sockLen);
            start = clock();
        }
    }
    memcpy(&packet2, buffer2, sizeof(packet2));
    u_short res = packetcheck((u_short*)&packet2, sizeof(packet2));

    if (packet2.head.flags == FLAG_ACK && packet2.head.seq == 1 && res == 0) {
        std::cout << "[Client]: Received ACK packet with seq=" <<
        packet2.head.seq << std::endl;
        ackReceived = true;
    }
    else{
        std::cout << "[Client]: Failed to receive ACK packet." << std::endl;
        //return 0;
    }

    // 等待服务器发送的 FIN_ACK 包
    bool finackReceived = false;
    Packet packet3;
    char* buffer3 = new char[sizeof(packet3)];

    while(1) {
        u_short res = packetcheck((u_short*)&packet3, sizeof(packet3));

        if (recvfrom(s, buffer3, sizeof(packet3), 0, (sockaddr*)&ClientAddr,
        &sockLen)<=0) {
            //cout << " 未收到 FIN_ACK 包" << endl;
        }
        memcpy(&packet3, buffer3, sizeof(packet3));
        if (packet3.head.flags == FLAG_FIN_ACK && packet3.head.seq == 2 &&
        res == 0) {
            std::cout << "[Client]: Received FIN_ACK packet with seq=" <<
            packet3.head.seq << std::endl;
            finackReceived = true;
            break;
        }
```

```

}
start = clock();

if (!finackReceived) {
    std::cout << "[Client]: Failed to receive FIN_ACK packet." <<
    std::endl;
    return 0;
}

// 发送 ACK 包
Packet packet4;
packet4.head.seq = 3;
packet4.head.flags = FLAG_ACK;
packet4.head.Check = packetcheck((u_short*)&packet4, sizeof(packet4));
char* buffer4 = new char[sizeof(packet4)];
memcpy(buffer4, &packet4, sizeof(packet4));
int flag4 = 0;

bool ackSentSuccessfully = false;

while (!ackSentSuccessfully) {
    flag4 = sendto(s, buffer4, sizeof(packet4), 0,
    (sockaddr*)&ClientAddr, sockLen);
    if (flag4 != -1) { // 发送成功
        std::cout << "[Client]: ACK packet sent successfully." <<
        std::endl;
        ackSentSuccessfully = true;
    }
    else {
        std::cout << "[Client]: Failed to send ACK packet, retrying..."
        << std::endl;
        return 0;
    }
}

iMode = 0; // 0: 阻塞
ioctlsocket(s, FIONBIO, &iMode); // 恢复成阻塞模式
return 1;
}
```

服务器端断开连接代码如下：

```
int serverCloseConnection(SOCKET s, sockaddr_in& ServerAddr, int&
sockLen) {
// 设置为非阻塞模式，避免卡在 recvfrom
u_long iMode = 1; // 0：阻塞
ioctlsocket(s, FIONBIO, &iMode); // 非阻塞设置

Packet packet1;
char* buffer1 = new char[sizeof(packet1)];
int flag = 0;

// 等待客户端发送的 FIN_ACK 包
bool finackReceived = false;

while(1) {
    if (recvfrom(s, buffer1, sizeof(packet1), 0, (sockaddr*)&ServerAddr,
    &sockLen) > 0) {
        memcpy(&(packet1), buffer1, sizeof(packet1));
        u_short res = packetcheck((u_short*)&packet1, sizeof(packet1));
        if (packet1.head.flags == FLAG_FIN_ACK && packet1.head.seq == 0
        && res == 0) {
            std::cout << "[Server]: Received FIN_ACK packet with seq=" <<
            packet1.head.seq << std::endl;
            finackReceived = true;
            break;
        }
    }
    else {
        continue;
    }
}

if (!finackReceived) {
    std::cout << "[Server]: Failed to receive FIN_ACK packet." <<
    std::endl;
    return 0;
}

// 发送 ACK 包
Packet packet2;
packet2.head.seq = 1;
packet2.head.flags = FLAG_ACK;
```

```
packet2.head.Check = packetcheck((u_short*)&packet2, sizeof(packet2));
char* buffer2 = new char[sizeof(packet2)];
memcpy(buffer2, &packet2, sizeof(packet2));

flag = sendto(s, buffer2, sizeof(packet2), 0, (sockaddr*)&ServerAddr,
sockLen);
if (flag != -1) { // 发送成功
    std::cout << "[Server]: ACK packet sent successfully." << std::endl;
}
else {
    std::cout << "[Server]: Failed to send ACK packet." << std::endl;
}

// 发送 FIN_ACK 包
Packet packet3;
packet3.head.seq = 2;
packet3.head.flags = FLAG_FIN_ACK;
packet3.head.Check = packetcheck((u_short*)&packet3, sizeof(packet3));
char* buffer3 = new char[sizeof(packet3)];
memcpy(buffer3, &packet3, sizeof(packet3));

bool finackSentSuccessfully = false;

while (!finackSentSuccessfully) {
    flag = sendto(s, buffer3, sizeof(packet3), 0, (sockaddr*)&ServerAddr,
    sockLen);
    if (flag != -1) { // 发送成功
        std::cout << "[Server]: FIN_ACK packet sent successfully." <<
        std::endl;
        finackSentSuccessfully = true;
    }
    else {
        std::cout << "[Server]: Failed to send FIN_ACK packet,
        retrying..." << std::endl;
        std::this_thread::sleep_for(std::chrono::seconds(1));
    }
}
clock_t start = clock();

if (flag == -1) { // 发送失败
    std::cout << "[Server]: Failed to send FIN_ACK packet." << std::endl;
    return 0;
}

// 等待客户端发送的 ACK 包
Packet packet4;
char* buffer4 = new char[sizeof(packet4)];
int flag4 = 0;

while (recvfrom(s, buffer4, sizeof(packet4), 0, (sockaddr*)&ServerAddr,
&sockLen) <= 0) {
    if (clock() - start > MAX_TIME)//超时
    {
        return 1;
        /*std::cout << "[Server]:Timeout Retransmission,resending ACK
        packet..." << std::endl;
        sendto(s, buffer3, sizeof(packet3), 0, (sockaddr*)&ServerAddr,
        sockLen);
        start = clock();*/
    }
}
//如果接收成功
memcpy(&(packet4), buffer4, sizeof(packet4.head));
u_short res = packetcheck((u_short*)&packet4, sizeof(packet4));

if (packet4.head.flags == FLAG_ACK && packet4.head.seq == 3 && res == 0) {
    std::cout << "[Server]: Received ACK packet with seq=" <<
    packet4.head.seq << std::endl;
}

iMode = 0; // 0: 阻塞
ioctlsocket(s, FIONBIO, &iMode); // 恢复成阻塞模式
return 1;
}
```

2.6 超时重传

本次实验基于 rdt3.0 以及停等机制设计了协议，rdt3.0 是一种改进的可靠数据传输协议，它在停等协议的基础上进行了扩展，增加了更多的功能和机制，以提高数据传输的效率和可靠性。每个数据包都有一个唯一的序列号，用于标识数据包的顺序，接收方发送的确认消息（ACK）中包含确认号，表示已成功接收的数据包的序列号。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，发送方会重新发送该数据包。

2.7 数据传输

数据传输中发送端和接收端均采用 rdt3.0。数据在传输时，将一个文件分为数个包进行分段传输，每个包的内容为数据头 + 数据，我在程序中设置每个包最大为 1024 字节。

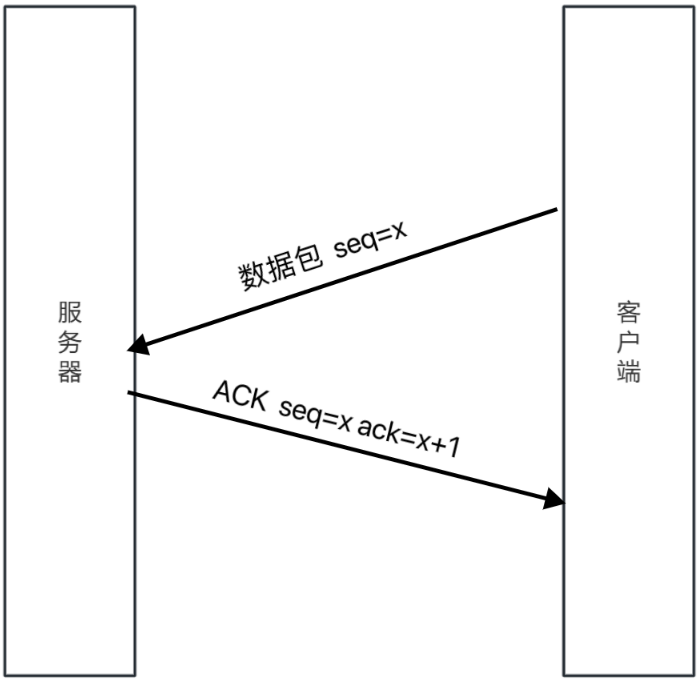


图 2.4: 数据传输示意图

数据传输流程如下：

客户端向服务器端发送序列号 seq=x 的数据包（标志位为 0 表示为数据包），数据在传输过程中有以下几种情况：

（1）数据包成功传输，服务器端通过判断序列号，标志位，校验和确认数据包是否正确，数据包正确，服务器端发送标志位为 ACK、序列号 seq=x，确认号 ack=x+1 的数据包回客户端，表示已经成功接收该数据包。此时客户端比较收到的确认号 ack 是否等于发送确认号 seq+1，如果是，则确认文件发送成功，继续发送下一个数据包。

（2）数据包成功传输，服务器端通过判断序列号，标志位，校验和确认数据包是否正确，若其中一项不正确，则服务器端不发送 ACK，客户端在设置的 MAX_TIME（此实验中我设置为 0.5s）内未收到由服务器端发回的 ACK 确认信息，则重传该序列号 seq=x 的数据包。

（3）数据包未成功传输，可能在传输过程中发生了丢失，此时服务器端未收到该数据包，不会发送 ACK 回客户端确认。客户端在设置的 MAX_TIME（此实验中我设置为 0.5s）内未收到由服务器端发回的 ACK 确认信息，则重传该序列号 seq=x 的数据包。

该数据包所有数据传输完毕后，客户端会发送一个标志位为 OVER 的数据包给服务器端表示发送完毕，服务器端收到后发送一个标志位为 ACK 的数据包回客户端表示确认，至此文件发送完毕。

客户端数据传输代码如下：

```
void send(SOCKET& socketClient, SOCKADDR_IN& servAddr,
int& servAddrlen, char* message, int messagelen)
{
u_long mode = 1;
ioctlsocket(socketClient, FIONBIO, &mode);
int seqnum = 0;
int packagenum = (messagelen / MAXSIZE)+ (messagelen % MAXSIZE != 0);
cout << packagenum << endl;
int len = 0;

for (int i = 0; i < packagenum; i++){
    Packet packet1;

    len = ((i == (packagenum - 1)) ? (messagelen - ((packagenum - 1) *
    MAXSIZE)) : MAXSIZE);
    //cout << len << endl;
    //cout << len - ((packagenum - 1) * MAXSIZE) << endl;
    char* buffer1 = new char[len + sizeof(packet1)];
    packet1.head.len = len;
    packet1.head.seq = seqnum;//序列号
    packet1.head.Check = 0;

    //计算校验和
    u_short check = packetcheck((u_short*)&packet1, sizeof(packet1));//计
    算校验和
    packet1.head.Check = check;
    packet1.head.flags = 0;
    packet1.head.ack = 0;
    memcpy(buffer1, &packet1, sizeof(packet1));
    char* mes = message + i * MAXSIZE;
    memcpy(buffer1 + sizeof(packet1), mes, len);//将数据复制到缓冲区的后部分
    sendto(socketClient, buffer1, len + sizeof(packet1), 0,
    (sockaddr*)&servAddr, servAddrlen);//发送
    cout << " 发送文件大小为 " << len << " bytes!" << " Flag:" <<
    int(packet1.head.flags) << " SEQ:" << int(packet1.head.seq) << "
    ACK:" << int(packet1.head.ack) << " CHECK:" <<
    int(packet1.head.Check) << endl;
    clock_t start = clock();//记录发送时间

    //接收 ACK
    Packet packet2;
    char* buffer2 = new char[sizeof(packet2)];
    //如果接受失败则继续等待
    while (recvfrom(socketClient, buffer2, sizeof(packet2), 0,
    (sockaddr*)&servAddr, &servAddrlen) <= 0)
    {
        //cout << "error" << endl;
        if (clock() - start > MAX_TIME)
        {
            //memcpy(buffer1, &packet1, sizeof(packet1));
            sendto(socketClient, buffer1, len + sizeof(packet1), 0,
            (sockaddr*)&servAddr, servAddrlen);//发送
            cout << " 发送超时, 重传文件大小为 " << len << " bytes! " << endl;
            cout << " 发送文件大小为 " << len << " bytes!" << " Flag:" <<
            int(packet1.head.flags) << " SEQ:" << int(packet1.head.seq)
            << " ACK:" << int(packet1.head.ack) << " CHECK:" <<
            int(packet1.head.Check) << endl;
            //system("pause");
            start = clock();//记录发送时间
        }
    }
    memcpy(&packet2, buffer2, sizeof(packet2));//缓冲区接收到信息, 读取
    u_short check2 = packetcheck((u_short*)&packet2, sizeof(packet2));
    //cout << "packet2.head.ack: " << packet2.head.ack << endl;
    //cout << "seqnum: " << seqnum << endl;
    if ((packet2.head.ack == (seqnum + 1) % 256) && packet2.head.flags ==
    FLAG_ACK && check2 == 0)
    {
        cout << " 接收到 ACK, 确认数据包发送成功  Flag:" <<
        int(packet2.head.flags) << " SEQ:" << int(packet2.head.seq) << "
        ACK:" << int(packet2.head.ack) << " CHECK:" <<
        int(packet2.head.Check) << endl;
    }
    else
    {//重传
        if (packet2.head.ack != seqnum + 1)
            cout << "1";
        if (packet2.head.flags != FLAG_ACK)
            cout << "2";
        if (check2 != 0)
            cout << "3";

        i--;
        cout << "error";
        //system("pause");
        continue;
    }
    seqnum = (seqnum + 1) % 256;
}

//发送结束信息
Packet packet3;
char* Buffer3 = new char[sizeof(packet3)];
packet3.head.flags = OVER;//结束信息
packet3.head.Check = 0;
u_short temp = packetcheck((u_short*)&packet3, sizeof(packet3));
packet3.head.Check = temp;
//发送结束信息
memcpy(Buffer3, &packet3, sizeof(packet3));
sendto(socketClient, Buffer3, sizeof(packet3), 0, (sockaddr*)&servAddr,
servAddrlen);
cout << " 接收文件完毕! " << endl;
clock_t start = clock();

Packet packet4;
char* Buffer4 = new char[sizeof(packet4)];

//接收结束确认
//如果接收失败
while (recvfrom(socketClient, Buffer4, sizeof(packet4), 0,
(sockaddr*)&servAddr, &servAddrlen) <= 0)
{
    if (clock() - start > MAX_TIME)
    {
        sendto(socketClient, Buffer3, sizeof(packet3), 0,
        (sockaddr*)&servAddr, servAddrlen);
        cout << " 结束信息超时重传" << endl;
        start = clock();
    }
}
memcpy(&packet4, Buffer4, sizeof(packet4));//缓冲区接收到信息，读取
u_short check4 = packetcheck((u_short*)&packet4, sizeof(packet4));
if (packet4.head.flags == OVER && check4 == 0)
{
    cout << " 对方已成功接收文件!" << endl;
```

```
}
else
{
    cout << " 未成功接收文件!" << endl;
}


mode = 0;
ioctlsocket(socketClient, FIONBIO, &mode);//改回阻塞模式
return;
}
```

服务器端接收数据代码如下：

```
int RecvMessage(SOCKET& sockServ, SOCKADDR_IN& ClientAddr, int&
ClientAddrLen, char* message)
{
u_long mode = 1;
ioctlsocket(sockServ, FIONBIO, &mode);  // 非阻塞模式
int filesize = 0;//文件长度

int seq = 0;//序列号
int ack = 1;

while (1)
{
    Packet packet1;
    char* Buffer1 = new char[MAXSIZE + sizeof(packet1)];

    while(recvfrom(sockServ, Buffer1, sizeof(packet1) + MAXSIZE, 0,
    (sockaddr*)&ClientAddr, &ClientAddrLen)<=0);//接收报文长度

    memcpy(&packet1, Buffer1, sizeof(packet1));
    //判断是否是结束
    if (packet1.head.flags == OVER && packetcheck((u_short*)&packet1,
    sizeof(packet1)) == 0)
    {
        cout << " 文件接收完毕" << endl;
        break;
    }
    //处理数据报文
    else if(packet1.head.flags == 0 && packetcheck((u_short*)&packet1,
    sizeof(packet1))==0)
    {
        //判断是否接受的是别的包
        if (seq != int(packet1.head.seq))
        {
            //cout << seq << endl;
            //cout << ack << endl;
            //cout << packet1.head.seq << endl;
            //说明出了问题，返回 ACK
            Packet packet4;
            packet4.head.flags = FLAG_ACK;
            packet4.head.len = 0;
            packet4.head.seq = seq;
            packet4.head.Check = 0;
            u_short temp = packetcheck((u_short*)&packet4,
            sizeof(packet4));
            packet4.head.Check = temp;
            char* buffer4 = new char[sizeof(packet4)];
            memcpy(buffer4, &packet4, sizeof(packet4));
            //重发该包的 ACK
            sendto(sockServ, buffer4, sizeof(packet4), 0,
            (sockaddr*)&ClientAddr, ClientAddrLen);

            cout << " 服务器接收的序列号不一致，ack 申请重传" << endl;
            //system("pause");
            continue;//丢弃该数据包
        }

        //cout << "seq: " << seq << "head.seq: " << packet1.head.seq <<
        endl;
        //取出 buffer 中的内容
        int curr_size = packet1.head.len;

        cout << " 接收到文件大小为 " << curr_size << " bytes! Flag:" <<
        int(packet1.head.flags) << " SEQ : " << int(packet1.head.seq) <<
        " ACK : " << int(packet1.head.ack) << " CHECK : " <<
        int(packet1.head.Check) << endl;

        memcpy(message + filesize, Buffer1 + sizeof(packet1), curr_size);
        //cout << "size" << sizeof(message) << endl;
        filesize = filesize + curr_size;

```

```
        //返回ACK
        Packet packet2;
        char* Buffer2 = new char[sizeof(packet2)];

        packet2.head.flags = FLAG_ACK;
        //packet2.head.len = 0;
        packet2.head.seq = seq++;
        packet2.head.ack = ack++;
        packet2.head.Check = 0;
        packet2.head.Check = packetcheck((u_short*)&packet2,
        sizeof(packet2));
        memcpy(Buffer2, &packet2, sizeof(packet2));
        //重发该包的ACK
        sendto(sockServ, Buffer2, sizeof(packet2), 0,
        (sockaddr*)&ClientAddr, ClientAddrLen);

        cout << " 发送ACK，确认数据包发送成功 Flag:" <<
        int(packet2.head.flags) << " SEQ : " << int(packet2.head.seq) <<
        " ACK : " << int(packet2.head.ack) << " CHECK : " <<
        int(packet2.head.Check) << endl;
        if (seq > 255)
        {
            seq = seq - 256;
        }
        if (ack > 255)
        {
            ack = ack - 256;
        }
    }
    else {
        if (packetcheck((u_short*)&packet1, sizeof(packet1)) != 0) {
            cout << "error" << endl;
            system("pause");
        }
        cout << " 错误" << endl;
    }
}
//发送OVER信息
Packet packet3;
char* Buffer3 = new char[sizeof(packet3)];
packet3.head.flags = OVER;
packet3.head.Check = 0;
```

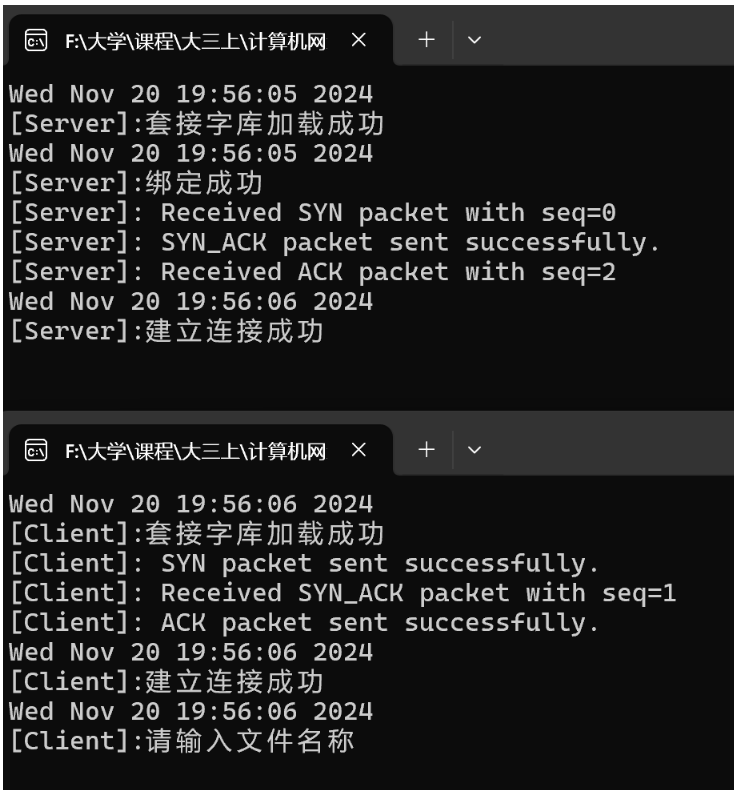
```
u_short temp = packetcheck((u_short*)&packet3, sizeof(packet3));
packet3.head.Check = temp;
memcpy(Buffer3, &packet3, sizeof(packet3));
if (sendto(sockServ, Buffer3, sizeof(packet3), 0, (sockaddr*)&ClientAddr,
ClientAddrLen) == -1)
{
    cout << " 发送结束信息错误" << endl;
    return -1;
}
cout << " 结束信息发送完毕! " << endl;
mode = 0;
ioctlsocket(sockServ, FIONBIO, &mode);  // 阻塞模式

return filesize;
}
```

3 运行结果

3.1 连接建立

经过三次握手连接建立运行结果如下:



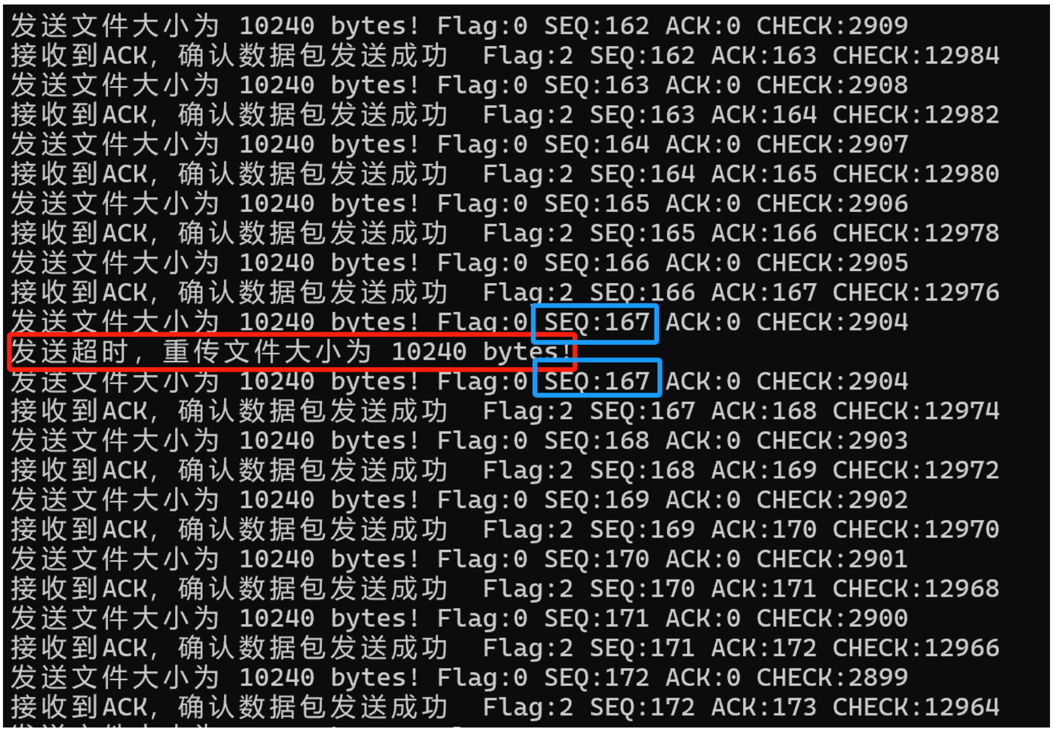
```
Wed Nov 20 19:56:05 2024
[Server]:套接字库加载成功
Wed Nov 20 19:56:05 2024
[Server]:绑定成功
[Server]: Received SYN packet with seq=0
[Server]: SYN_ACK packet sent successfully.
[Server]: Received ACK packet with seq=2
Wed Nov 20 19:56:06 2024
[Server]:建立连接成功
```

```
Wed Nov 20 19:56:06 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.
Wed Nov 20 19:56:06 2024
[Client]:建立连接成功
Wed Nov 20 19:56:06 2024
[Client]:请输入文件名称
```

图 3.5: 连接建立

3.2 数据传输

客户端发送数据结果如下：

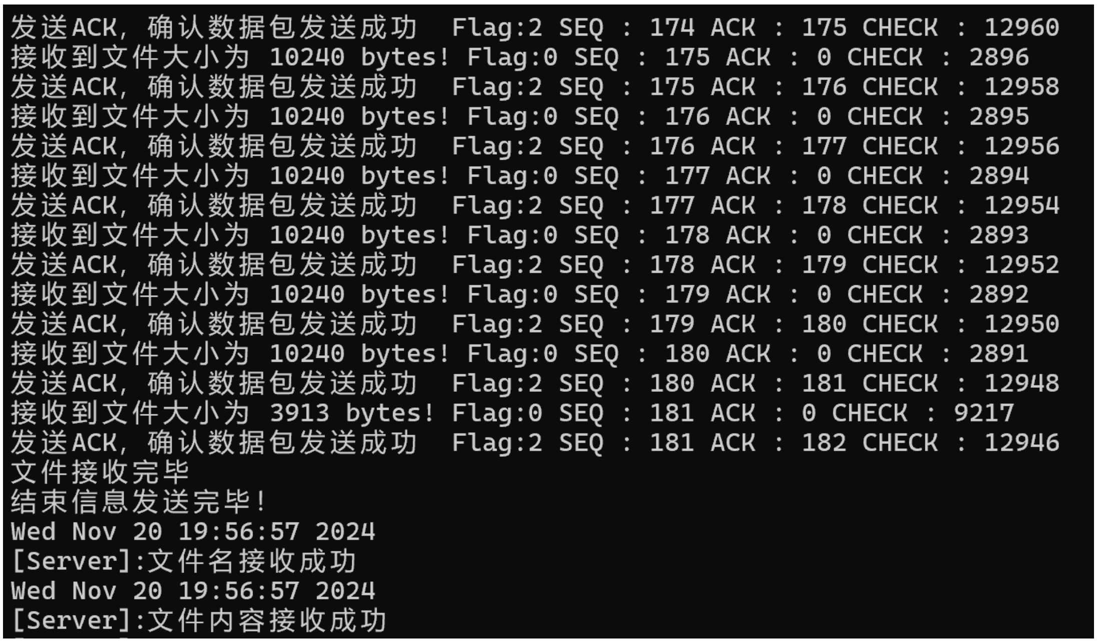


```
发送文件大小为 10240 bytes! Flag:0 SEQ:162 ACK:0 CHECK:2909
接收到ACK, 确认数据包发送成功  Flag:2 SEQ:162 ACK:163 CHECK:12984
发送文件大小为 10240 bytes! Flag:0 SEQ:163 ACK:0 CHECK:2908
接收到ACK, 确认数据包发送成功  Flag:2 SEQ:163 ACK:164 CHECK:12982
发送文件大小为 10240 bytes! Flag:0 SEQ:164 ACK:0 CHECK:2907
接收到ACK, 确认数据包发送成功  Flag:2 SEQ:164 ACK:165 CHECK:12980
发送文件大小为 10240 bytes! Flag:0 SEQ:165 ACK:0 CHECK:2906
接收到ACK, 确认数据包发送成功  Flag:2 SEQ:165 ACK:166 CHECK:12978
发送文件大小为 10240 bytes! Flag:0 SEQ:166 ACK:0 CHECK:2905
接收到ACK, 确认数据包发送成功  Flag:2 SEQ:166 ACK:167 CHECK:12976
发送文件大小为 10240 bytes! Flag:0 SEQ:167 ACK:0 CHECK:2904
发送超时, 重传文件大小为 10240 bytes!
发送文件大小为 10240 bytes! Flag:0 SEQ:167 ACK:0 CHECK:2904
接收到ACK, 确认数据包发送成功  Flag:2 SEQ:167 ACK:168 CHECK:12974
发送文件大小为 10240 bytes! Flag:0 SEQ:168 ACK:0 CHECK:2903
接收到ACK, 确认数据包发送成功  Flag:2 SEQ:168 ACK:169 CHECK:12972
发送文件大小为 10240 bytes! Flag:0 SEQ:169 ACK:0 CHECK:2902
接收到ACK, 确认数据包发送成功  Flag:2 SEQ:169 ACK:170 CHECK:12970
发送文件大小为 10240 bytes! Flag:0 SEQ:170 ACK:0 CHECK:2901
接收到ACK, 确认数据包发送成功  Flag:2 SEQ:170 ACK:171 CHECK:12968
发送文件大小为 10240 bytes! Flag:0 SEQ:171 ACK:0 CHECK:2900
接收到ACK, 确认数据包发送成功  Flag:2 SEQ:171 ACK:172 CHECK:12966
发送文件大小为 10240 bytes! Flag:0 SEQ:172 ACK:0 CHECK:2899
接收到ACK, 确认数据包发送成功  Flag:2 SEQ:172 ACK:173 CHECK:12964
```

图 3.6: 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。发生丢包时，显示发送超时，进行数据包的重传，在运行结果中可以看出，重传的数据包为上一个没有成功传输的数据包，序列号一致，运行结果正确。

服务器端接收数据结果如下：

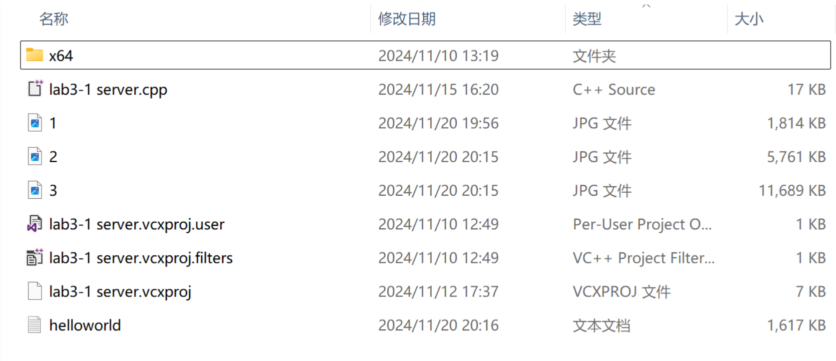


```
发送ACK, 确认数据包发送成功  Flag:2 SEQ : 174 ACK : 175 CHECK : 12960
接收到文件大小为 10240 bytes! Flag:0 SEQ : 175 ACK : 0 CHECK : 2896
发送ACK, 确认数据包发送成功  Flag:2 SEQ : 175 ACK : 176 CHECK : 12958
接收到文件大小为 10240 bytes! Flag:0 SEQ : 176 ACK : 0 CHECK : 2895
发送ACK, 确认数据包发送成功  Flag:2 SEQ : 176 ACK : 177 CHECK : 12956
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
结束信息发送完毕!
Wed Nov 20 19:56:57 2024
[Server]:文件名接收成功
Wed Nov 20 19:56:57 2024
[Server]:文件内容接收成功
```

图 3.7: 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：

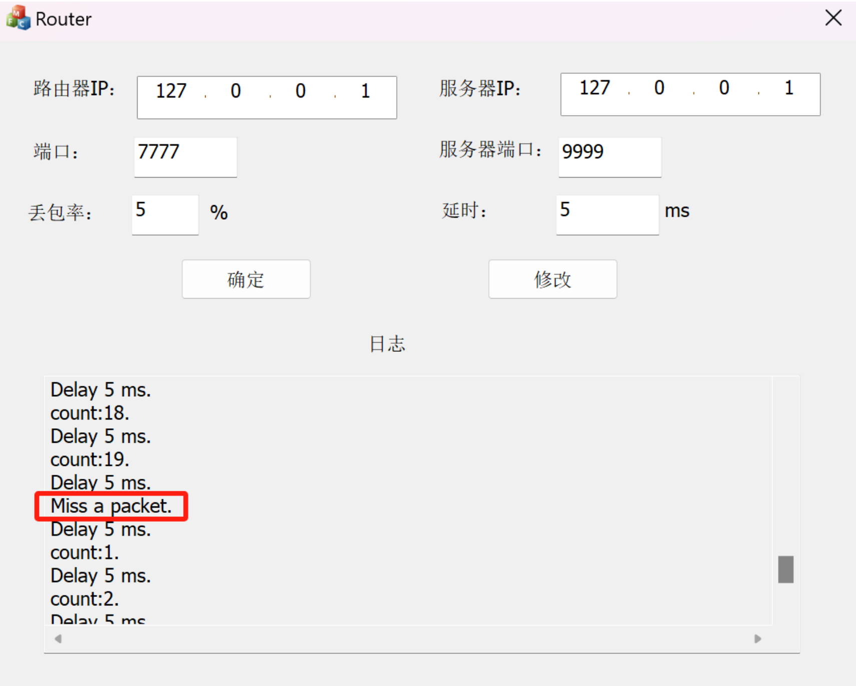


名称	修改日期	类型	大小
x64	2024/11/10 13:19	文件夹	
lab3-1 server.cpp	2024/11/15 16:20	C++ Source	17 KB
1	2024/11/20 19:56	JPG 文件	1,814 KB
2	2024/11/20 20:15	JPG 文件	5,761 KB
3	2024/11/20 20:15	JPG 文件	11,689 KB
lab3-1 server.vcxproj.user	2024/11/10 12:49	Per-User Project O...	1 KB
lab3-1 server.vcxproj.filters	2024/11/10 12:49	VC++ Project Filter...	1 KB
lab3-1 server.vcxproj	2024/11/12 17:37	VCXPROJ 文件	7 KB
helloworld	2024/11/20 20:16	文本文档	1,617 KB

图 3.8: 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。

router 设置：



路由器IP: 127 . 0 . 0 . 1
服务器IP: 127 . 0 . 0 . 1
端口: 7777
服务器端口: 9999
丢包率: 5 %
延时: 5 ms

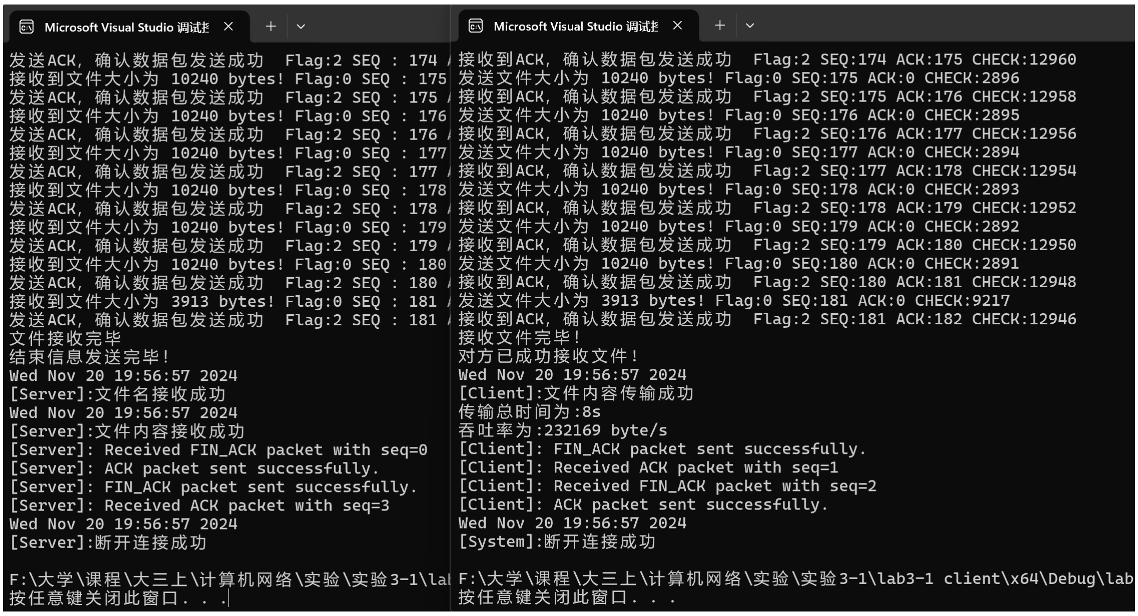
```
Delay 5 ms.
count:18.
Delay 5 ms.
count:19.
Delay 5 ms.
Miss a packet.
Delay 5 ms.
count:1.
Delay 5 ms.
count:2.
Delay 5 ms
```

图 3.9: router 设置

我将丢包率设为 5%, 即每 20 个数据包丢一个，延时率设为 5ms，可以看到 router 的输出为每 20 个数据包 Miss a packet.

3.3 断开连接

经过四次挥手连接断开运行结果如下：



```
发送ACK, 确认数据包发送成功  Flag:2 SEQ : 174
接收到文件大小为 10240 bytes! Flag:0 SEQ : 175
发送ACK, 确认数据包发送成功  Flag:2 SEQ : 175
接收到文件大小为 10240 bytes! Flag:0 SEQ : 176
发送ACK, 确认数据包发送成功  Flag:2 SEQ : 176
接收到文件大小为 10240 bytes! Flag:0 SEQ : 177
发送ACK, 确认数据包发送成功  Flag:2 SEQ : 177
接收到文件大小为 10240 bytes! Flag:0 SEQ : 178
发送ACK, 确认数据包发送成功  Flag:2 SEQ : 178
接收到文件大小为 10240 bytes! Flag:0 SEQ : 179
发送ACK, 确认数据包发送成功  Flag:2 SEQ : 179
接收到文件大小为 10240 bytes! Flag:0 SEQ : 180
发送ACK, 确认数据包发送成功  Flag:2 SEQ : 180
接收到文件大小为 3913 bytes! Flag:0 SEQ : 181
发送ACK, 确认数据包发送成功  Flag:2 SEQ : 181
文件接收完毕
结束信息发送完毕!
Wed Nov 20 19:56:57 2024
[Server]:文件名接收成功
Wed Nov 20 19:56:57 2024
[Server]:文件内容接收成功
[Server]: Received FIN_ACK packet with seq=0
[Server]: ACK packet sent successfully.
[Server]: FIN_ACK packet sent successfully.
[Server]: Received ACK packet with seq=3
Wed Nov 20 19:56:57 2024
[Server]:断开连接成功

F:\大学\课程\大三上\计算机网络\实验\实验3-1\la
按任意键关闭此窗口. . .
```

```
接收到ACK, 确认数据包发送成功  Flag:2 SEQ:174 ACK:175 CHECK:12960
发送文件大小为 10240 bytes! Flag:0 SEQ:175 ACK:0 CHECK:2896
接收到ACK, 确认数据包发送成功  Flag:2 SEQ:175 ACK:176 CHECK:12958
发送文件大小为 10240 bytes! Flag:0 SEQ:176 ACK:0 CHECK:2895
接收到ACK, 确认数据包发送成功  Flag:2 SEQ:176 ACK:177 CHECK:12956
发送文件大小为 10240 bytes! Flag:0 SEQ:177 ACK:0 CHECK:2894
接收到ACK, 确认数据包发送成功  Flag:2 SEQ:177 ACK:178 CHECK:12954
发送文件大小为 10240 bytes! Flag:0 SEQ:178 ACK:0 CHECK:2893
接收到ACK, 确认数据包发送成功  Flag:2 SEQ:178 ACK:179 CHECK:12952
发送文件大小为 10240 bytes! Flag:0 SEQ:179 ACK:0 CHECK:2892
接收到ACK, 确认数据包发送成功  Flag:2 SEQ:179 ACK:180 CHECK:12950
发送文件大小为 10240 bytes! Flag:0 SEQ:180 ACK:0 CHECK:2891
接收到ACK, 确认数据包发送成功  Flag:2 SEQ:180 ACK:181 CHECK:12948
发送文件大小为 3913 bytes! Flag:0 SEQ:181 ACK:0 CHECK:9217
接收到ACK, 确认数据包发送成功  Flag:2 SEQ:181 ACK:182 CHECK:12946
接收文件完毕!
对方已成功接收文件!
Wed Nov 20 19:56:57 2024
[Client]:文件内容传输成功
传输总时间为:8s
吞吐率为:232169 byte/s
[Client]: FIN_ACK packet sent successfully.
[Client]: Received ACK packet with seq=1
[Client]: Received FIN_ACK packet with seq=2
[Client]: ACK packet sent successfully.
Wed Nov 20 19:56:57 2024
[System]:断开连接成功

F:\大学\课程\大三上\计算机网络\实验\实验3-1\lab3-1 client\x64\Debug\lab
按任意键关闭此窗口. . .
```

图 3.10: 断开连接