

计算机网络第三次作业第二部分

张昊星 2113419

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，完成给定测试文件的传输。

实验设计

协议设计

1. 报文格式

```
struct HEADER {
    u_short sum = 0; //校验和 16位
    u_short datasize = 0; //所包含数据长度 16位
    unsigned char flag = 0; //八位，使用后三位，排列是FIN ACK SYN
    unsigned char SEQ = 0; //八位，传输的序列号，0~255，超过后mod
    HEADER() {
        sum = 0;
        datasize = 0;
        flag = 0;
        SEQ = 0;
    }
};
```

报文头长度为48位，前16位为数据长度，用于记录数据区的大小，17-32位为校验和，用于检验传输的正确性，33-40位为标志位，只使用低3位，分别为FIN，ACK，SYN，40-48位为传输的数据包的序号（0~255循环使用）

2. 连接与断开

类似于TCP的握手与挥手功能：

- 三次握手进行连接：

首先，客户端向服务端发送数据包，其中SYN=1，ACK=0，FIN=0

服务端接收到数据包后，向客户端发送SYN=0，ACK=1，FIN=0

客户端再次接收到数据包后，向服务端发送SYN=1，ACK=1，FIN=0

服务端接收到数据包后，连接成功建立，可以进行数据传输

- 四次挥手断开连接：

首先，客户端向服务端发送数据包，其中SYN=0，ACK=0，FIN=1

服务端接收到数据包后，向客户端发送SYN=0，ACK=1，FIN=0

客户端再次接收到数据包后，向服务端发送SYN=0，ACK=1，FIN=1

服务端接收到数据包后，向客户端发送SYN=0，ACK=1，FIN=1

客户端接收到数据包后，连接成功断开

3.数据传输

发送端和接收端的接收机均采用GBN，停等协议虽然编写简单，效果稳定，但是其效率比较低，实际中很难利用其进行大量的数据包发送，所以一般情况下，采用流水线的方式进行数据包的发送和接收。Go-Back-N协议采用累积确认的方式，GBN的关键是发送方能够在收到确认之前发送多个分组，但接收方只能缓存一个分组。发送方为发送出去的分组保留副本，直到来自接收方确认达到。

4.滑动窗口

窗口分为左边界、发送边界和右边界，窗口大小固定。窗口左边界左侧为已经发送并得到确认的数据，左边界到发送边界的数据为已发送但未得到确认的数据，发送边界到右边界为等待发送的数据，右边界右侧为不可发送的数据。

代码实现（仅展示与3-1不同的代码）

传输数据

- 发送单个数据包

```
void send_package(SOCKET& socketClient, SOCKADDR_IN& servAddr, int& servAddrLen, char* message, int len, int& order){
    HEADER header;
    char* buffer = new char[MAXSIZE + sizeof(header)];
    header.datasize = len;
    header.SEQ = unsigned char(order); //序列号
    memcpy(buffer, &header, sizeof(header));
    memcpy(buffer + sizeof(header), message, sizeof(header) + len);
    u_short check = cksum((u_short*)buffer, sizeof(header) + len); //计算校验和
    header.sum = check;
    memcpy(buffer, &header, sizeof(header));
    sendto(socketClient, buffer, len + sizeof(header), 0,
    (sockaddr*)&servAddr, servAddrLen);
    cout << "Send message " << len << " bytes!" << " flag:" <<
    int(header.flag) << " SEQ:" << int(header.SEQ) << " SUM:" << int(header.sum)
    << endl;
}
```

准备发送的数据包：创建一个包含数据的缓冲区 `buffer`，大小为 `MAXSIZE + sizeof(header)`。将数据包的头信息填充到 `buffer` 的开头，并将实际数据复制到 `buffer` 中。

计算校验和：调用 `cksum()` 函数计算 `buffer` 中数据的校验和，并将结果保存到 `header.sum` 中。

发送数据包：使用 `sendto()` 函数将数据包发送到服务端。

- 发送文件

```
void send(SOCKET& socketClient, SOCKADDR_IN& servAddr, int& servAddrLen,
char* message, int len) {
    HEADER header;
    char* Buffer = new char[sizeof(header)];
    int packagenum = len / MAXSIZE + (len % MAXSIZE != 0); // 计算需要发送的数据包数量

    int head = -1; // 缓冲区头部，前方为已经被确认的报文
    int tail = 0; // 缓冲区尾部，指向待发送的报文
    clock_t start; // 记录发送时间
```

```

while (head < packagenum - 1) {
    if (tail - head < WINDOW_SIZE && tail != packagenum) {
        // 如果窗口内有空余位置且还有待发送的数据包，则发送数据包
        send_package(socketClient, servAddr, servAddrLen, message + tail
* MAXSIZE, tail == packagenum - 1 ? len - (packagenum - 1) * MAXSIZE :
MAXSIZE, tail);
        start = clock(); // 记录发送时间
        tail++; // 移动尾部指针
    }
    //sleep(1);
    // 将套接字设置为非阻塞模式，并尝试接收 ACK 报文
    u_long mode = 1;
    ioctlsocket(socketClient, FIONBIO, &mode);
    if (recvfrom(socketClient, Buffer, MAXSIZE, 0, (sockaddr*)&servAddr,
&servAddrLen)>0) {
        // 如果成功接收到数据
        memcpy(&header, Buffer, sizeof(header));
        if (header.flag == ACK) {
            if (header.SEQ >= head) {
                // 如果接收到的 ACK 序列号大于等于当前窗口的头部序列号
                head = header.SEQ; // 移动头部指针
                cout << "Send has been confirmed! flag:" <<
int(header.flag) << " SEQ:" << header.SEQ << endl;
            }
        }
    }
    else {
        if (clock() - start > MAX_TIME) {
            // 如果超时
            tail = head + 1; // 重置尾部指针
            cout << "Time Out! ReSend Message!" << endl;
            start = clock();
        }
    }

    mode = 0;
    ioctlsocket(socketClient, FIONBIO, &mode); // 将套接字设置为阻塞模式
}

// 发送结束信息
header.flag = OVER;
header.sum = 0;
u_short temp = cksum((u_short*)&header, sizeof(header));
header.sum = temp;
memcpy(Buffer, &header, sizeof(header));
sendto(socketClient, Buffer, sizeof(header), 0, (sockaddr*)&servAddr,
servAddrLen);
cout << "Send End!" << endl;
start = clock(); // 记录时间

// 等待接收端的确认结束信息
while (1) {
    u_long mode = 1;
    ioctlsocket(socketClient, FIONBIO, &mode); // 将套接字设置为非阻塞模式
    while (recvfrom(socketClient, Buffer, MAXSIZE, 0,
(sockaddr*)&servAddr, &servAddrLen) <= 0) {
        if (clock() - start > MAX_TIME) {
            // 如果超时

```

```

        char* Buffer = new char[sizeof(header)];
        header.flag = OVER;
        header.sum = 0;
        u_short temp = cksum((u_short*)&header, sizeof(header));
        header.sum = temp;
        memcpy(Buffer, &header, sizeof(header));
        sendto(socketClient, Buffer, sizeof(header), 0,
(sockaddr*)&servAddr, servAddrLen);
        cout << "Time Out! ReSend End!" << endl;
        start = clock();
    }
}

memcpy(&header, Buffer, sizeof(header)); // 读取接收到的数据包头部信息
u_short check = cksum((u_short*)&header, sizeof(header));
if (header.flag == OVER) {
    cout << "对方已成功接收文件!" << endl;
    break;
}
else {
    continue;
}
}

u_long mode = 0;
ioctlsocket(socketClient, FIONBIO, &mode); // 将套接字设置为阻塞模式
}

```

滑动窗口机制：使用两个指针 `head` 和 `tail`。`head` 表示已经被确认的数据包，`tail` 表示窗口中待发送的数据包。

发送数据包：在窗口内有空余位置且仍有待发送的数据包时，发送数据包。函数 `send_package` 负责发送数据包。

接收确认：使用非阻塞模式接收 `ACK` 报文，如果接收到 `ACK`，则根据序列号更新窗口。如果超时，则重置尾部指针 `tail`。

超时重传：在超过最大等待时间 `MAX_TIME` 后，重新发送未确认的数据包。在超时时，重置 `tail` 到窗口的下一个位置。

发送结束信息：发送端发送一个结束标志，表示文件传输结束。

等待确认：发送端等待接收端发送的确认结束信息。如果超时，则重新发送结束标志。

接收数据

```

int RecvMessage(SOCKET& sockServ, SOCKADDR_IN& ClientAddr, int& ClientAddrLen,
char* message) {
    long int file_length = 0; // 文件长度
    HEADER header;
    char* Buffer = new char[MAXSIZE + sizeof(header)];
    int seq = 0;

    while (1) {
        int length = recvfrom(sockServ, Buffer, sizeof(header) + MAXSIZE, 0,
(sockaddr*)&ClientAddr, &ClientAddrLen); // 接收报文长度
        memcpy(&header, Buffer, sizeof(header));
    }
}

```

```

        if (header.flag == unsigned char(0) && cksum((u_short*)Buffer, length -
sizeof(header))) {
            if (seq != header.SEQ) {
                continue;//丢弃该数据包
            }
            //seq = header.SEQ;
            //取出buffer中的内容
            cout << "Recv message " << length - sizeof(header) << " bytes!Flag:"
<< int(header.flag) << " SEQ : " << header.SEQ << " SUM:" << int(header.sum) <<
endl;

            char* temp = new char[length - sizeof(header)];
            memcpy(temp, Buffer + sizeof(header), length - sizeof(header));
            memcpy(message + file_length, temp, length - sizeof(header));
            file_length = file_length + int(header.datasize);
            //返回ACK
            header.flag = ACK;
            header.datasize = 0;
            header.SEQ = seq;
            header.sum = 0;
            u_short temp1 = cksum((u_short*)&header, sizeof(header));
            header.sum = temp1;
            memcpy(Buffer, &header, sizeof(header));
            sendto(sockServ, Buffer, sizeof(header), 0, (sockaddr*)&ClientAddr,
ClientAddrLen);
            cout << "Send to Clinet ACK:" << (int)header.flag << " SEQ:" <<
header.SEQ << endl;
            seq++;
        }
        //判断是否是结束
        if (header.flag == OVER && cksum((u_short*)&header, sizeof(header)) ==
0) {
            cout << "文件接收完毕" << endl;
            break;
        }
    }
    header.flag = OVER;
    header.sum = 0;
    u_short temp = cksum((u_short*)&header, sizeof(header));
    header.sum = temp;
    memcpy(Buffer, &header, sizeof(header));
    if (sendto(sockServ, Buffer, sizeof(header), 0, (sockaddr*)&ClientAddr,
ClientAddrLen) == -1) {
        return -1;
    }
    return file_length;
}

```

循环接收数据报文：使用 `recvfrom()` 从套接字接收数据报文，并将其存储在 `Buffer` 中。提取报文头部信息 `HEADER`，判断接收到的报文是否是结束标志。

确认序列号和校验和：对接收到的数据报文进行序列号和校验和的检查。如果序列号错误则丢弃该数据包。如果接收到的序列号与当前期望的序列号一致，则将数据从报文中提取出来，并存储到 `message` 缓冲区中。

发送确认 (ACK)：发送确认消息给客户端，确认接收到的数据包。

循环接收直到收到结束标志：循环接收直到接收到结束标志的数据包。

发送结束标志的确认：发送结束标志的确认消息给客户端。

实验结果

正常情况

C:\Users\LEGION\Desktop\git

请输入本服务器IP: 127.0.0.1
请输入本服务器端口: 2000
进入监听状态，等待客户端连接....

C:\Users\LEGION\Desktop\git

请输入目标IP: 127.0.0.1
请输入目标端口: 2000
请输入本机IP: 127.0.0.1
请输入本机端口: 2001
服务器连接成功
请输入文件名称:|

C:\Users\LEGION\Desktop\git

Recv message 8192 bytes!Flag:0 SEQ : 214 SUM:42939
Send to c\inet ACK:2 SEQ:214
Recv message 8192 bytes!Flag:0 SEQ : 215 SUM:59486
Send to c\inet ACK:2 SEQ:215
Recv message 8192 bytes!Flag:0 SEQ : 216 SUM:10488
Send to c\inet ACK:2 SEQ:216
Recv message 8192 bytes!Flag:0 SEQ : 217 SUM:22462
Send to c\inet ACK:2 SEQ:217
Recv message 8192 bytes!Flag:0 SEQ : 218 SUM:65496
Send to c\inet ACK:2 SEQ:218
Recv message 8192 bytes!Flag:0 SEQ : 219 SUM:39644
Send to c\inet ACK:2 SEQ:219
Recv message 8192 bytes!Flag:0 SEQ : 220 SUM:57178
Send to c\inet ACK:2 SEQ:220
Recv message 8192 bytes!Flag:0 SEQ : 221 SUM:47823
Send to c\inet ACK:2 SEQ:221
Recv message 8192 bytes!Flag:0 SEQ : 222 SUM:59672
Send to c\inet ACK:2 SEQ:222
Recv message 8192 bytes!Flag:0 SEQ : 223 SUM:46681
Send to c\inet ACK:2 SEQ:223
Recv message 8192 bytes!Flag:0 SEQ : 224 SUM:53500
Send to c\inet ACK:2 SEQ:224
Recv message 8192 bytes!Flag:0 SEQ : 225 SUM:2954
Send to c\inet ACK:2 SEQ:225
Recv message 5961 bytes!Flag:0 SEQ : 226 SUM:48466
Send to c\inet ACK:2 SEQ:226
文件接收完毕
四次握手结束，连接断开！
1.jpg已成功下载到本地
请按任意键继续。...

C:\Users\LEGION\Desktop\git

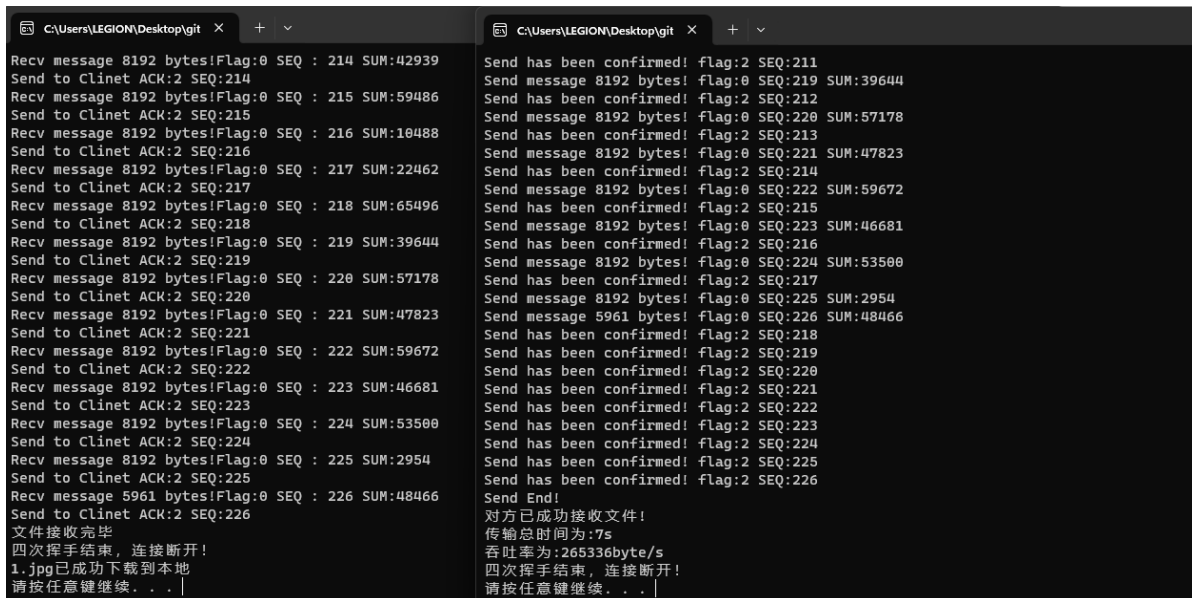
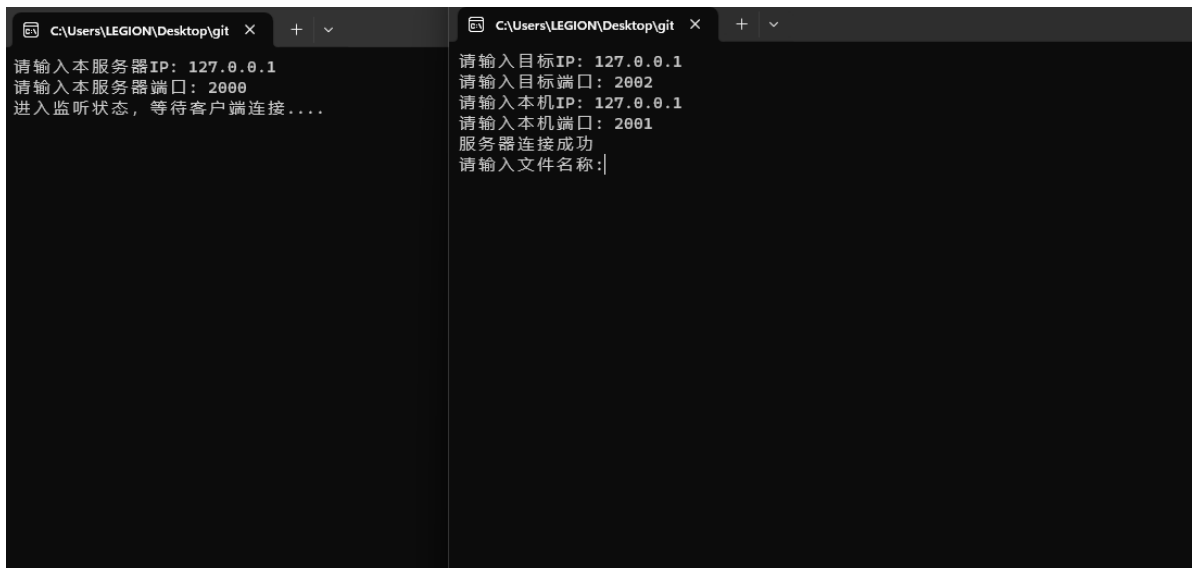
Send has been confirmed! flag:2 SEQ:214
Send message 8192 bytes! flag:0 SEQ:216 SUM:10488
Send has been confirmed! flag:2 SEQ:215
Send message 8192 bytes! flag:0 SEQ:217 SUM:22462
Send has been confirmed! flag:2 SEQ:216
Send message 8192 bytes! flag:0 SEQ:218 SUM:65496
Send has been confirmed! flag:2 SEQ:217
Send message 8192 bytes! flag:0 SEQ:219 SUM:39644
Send has been confirmed! flag:2 SEQ:218
Send message 8192 bytes! flag:0 SEQ:220 SUM:57178
Send has been confirmed! flag:2 SEQ:219
Send message 8192 bytes! flag:0 SEQ:221 SUM:47823
Send has been confirmed! flag:2 SEQ:220
Send message 8192 bytes! flag:0 SEQ:222 SUM:59672
Send has been confirmed! flag:2 SEQ:221
Send message 8192 bytes! flag:0 SEQ:223 SUM:46681
Send has been confirmed! flag:2 SEQ:222
Send message 8192 bytes! flag:0 SEQ:224 SUM:53500
Send has been confirmed! flag:2 SEQ:223
Send message 8192 bytes! flag:0 SEQ:225 SUM:2954
Send has been confirmed! flag:2 SEQ:224
Send message 5961 bytes! flag:0 SEQ:226 SUM:48466
Send has been confirmed! flag:2 SEQ:225
Send has been confirmed! flag:2 SEQ:226
Send End!
对方已成功接收文件！
传输总时间为:0s
吞吐量为:119fbyte/s
四次握手结束，连接断开！
请按任意键继续。...



接收到的图片大小与原图片一致，传输成功。

路由情况

路由器设置如下：



同样成功传输并在客户端运行界面可以看到“TIME OUT”的信号，说明超时重传以及滑动窗口成功。