

南开大学

计算机网络课程实验报告

实验3-2



学院：网络空间安全学院

专业：信息安全

学号：2113997

姓名：齐明杰

班级：信安2班

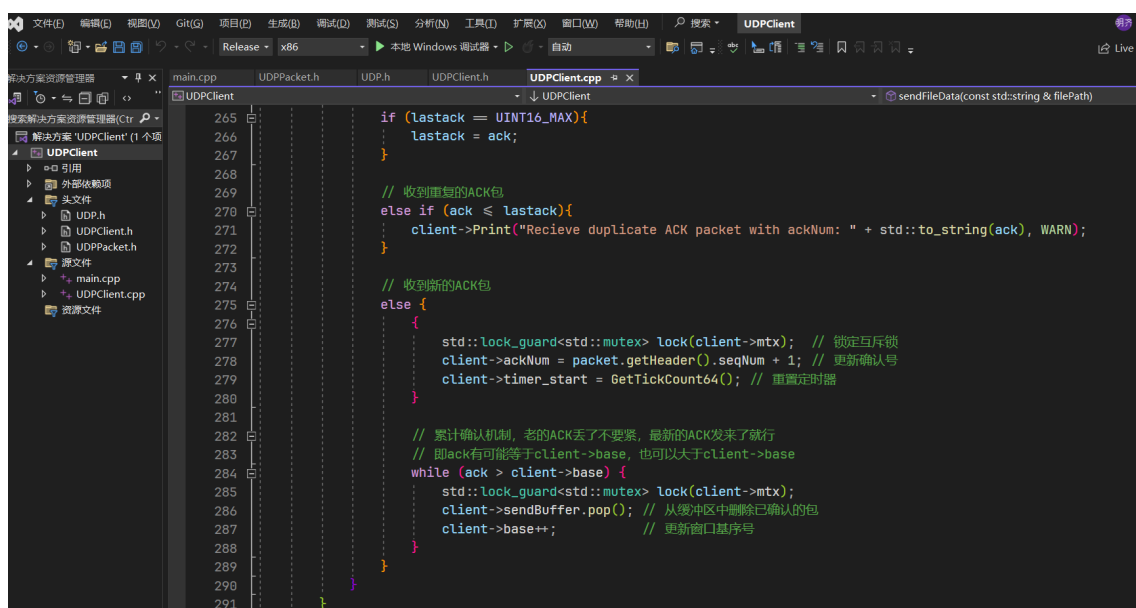
1 实验要求

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**，完成给定测试文件的传输。

- 协议设计：数据包格式，发送端和接收端交互，详细完整
- 流水线协议：多个序列号
- 发送缓冲区、接收缓冲区
- 累积确认：Go Back N
- 日志输出：收到/发送数据包的序号、ACK、校验和等，发送端和接收端的窗口大小等情况，传输时间与吞吐率
- 测试文件：必须使用助教发的测试文件（1.jpg、2.jpg、3.jpg、helloworld.txt）

2 实验环境

Windows 10 + Visual Studio 2022 community



```
265
266
267
268
269
270 // 收到重复的ACK包
271 else if (ack ≤ lastack){
272     client->Print("Recieve duplicate ACK packet with ackNum: " + std::to_string(ack), WARN);
273 }
274
275 // 收到新的ACK包
276 else {
277     {
278         std::lock_guard<std::mutex> lock(client->mtx); // 锁定互斥锁
279         client->ackNum = packet.getHeader().seqNum + 1; // 更新确认号
280         client->timer_start = GetTickCount64(); // 重置定时器
281     }
282
283     // 累积确认机制，老的ACK丢了不要紧，最新的ACK发来了就行
284     // 即ack有可能等于client->base，也可以大于client->base
285     while (ack > client->base) {
286         std::lock_guard<std::mutex> lock(client->mtx);
287         client->sendBuffer.pop(); // 从缓冲区中删除已确认的包
288         client->base++; // 更新窗口基序号
289     }
290
291 }
```

3 实验原理

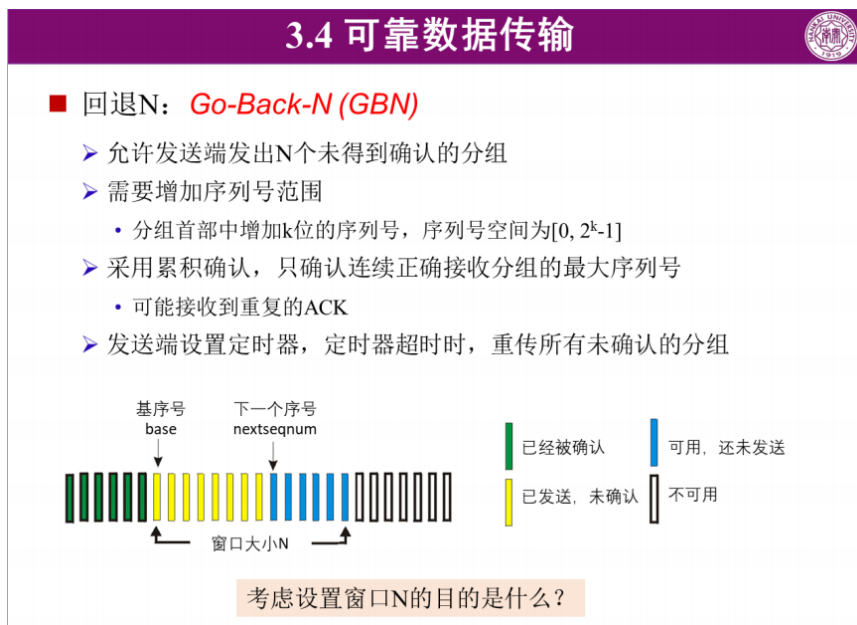
3.1 滑动窗口机制

滑动窗口是一种流控制机制，用于在不可靠的网络环境中实现可靠的数据传输。它通过维护发送方和接收方各自的窗口来控制发送和接收的数据量。发送方的窗口定义了它可以在等待确认之前发送的最大数据包数量。接收方的窗口则指定了它可以接受的未确认数据包的数量。窗口的大小可以根据网络状况动态调整，以实现高效且可靠的数据传输。这种机制还支持流水线传输，

允许同时发送多个数据包，从而提高整体传输效率。

3.2 Go-Back-N (GBN) 累积确认协议

Go-Back-N 是一种**流水线协议**，它允许发送方在停止并等待确认之前发送多个数据包。在这种协议中，如果一个数据包丢失或损坏，接收方会丢弃后续所有包并等待重传。这意味着发送方需要重传从丢失的包开始的所有包。GBN 使用累积确认，这意味着接收方只需确认最近接收的连续数据包序列号。如果接收方收到的数据包序列号大于期望的序列号，则该包及其后续包会被暂存，直到缺失的包到达并被确认。



3.3 握手挥手

- **握手**：在建立连接之前，客户端和服务端之间执行三次握手。这个过程涉及到客户端首先发送一个同步（SYN）包，服务器响应一个同步确认（SYN-ACK）包，然后客户端再次发送一个确认（ACK）包，以此建立连接。
- **挥手**：在传输数据结束后，进行四次挥手来断开连接。首先由发送方发送终止（FIN）包，接收方回复确认（ACK），然后接收方发送自己的终止包，发送方对此进行确认。这确保了双方都完成了数据的发送和接收。

3.4 超时重传

在Go-Back-N协议中，超时重传是一个关键机制。如果发送方在预定的超时时间内没有收到某个数据包的确认，它会重新发送从该数据包开始的所有数据包。超时机制确保了数据的可靠传输，即使在网络状况不佳的情况下。为了实现超时重传，发送方需要维护一个定时器，当发送数据包后启动，如果在定时器到期前未收到确认，则触发重传。

3.5 对比停等机制

与停等机制相比，滑动窗口和GBN提供了更高的效率和吞吐量。在停等机制中，每发送一个数据包，发送方必须等待其确认后才能发送下一个包。这会导致带宽的低效利用和延迟的增加，特别是在延迟大的网络中。相反，滑动窗口和GBN允许发送多个数据包而不必等待每个包的确认，从而有效利用带宽，减少了总体传输时间，增加了网络吞吐量。

3.6 累计确认机制

累计确认机制是Go-Back-N协议的核心特性，它显著提高了数据传输的效率和可靠性。在这种机制中，接收方不需要为每个接收到的数据包发送单独的确认。相反，接收方通过发送一个确认来表明它已成功接收到包括该确认号及其之前所有的数据包。例如，如果接收方发送了确认号5的确认，这意味着序列号为1到5的所有数据包都已被正确接收。这种方法减少了网络上的确认消息数量，降低了网络负载，从而提高了整体通信效率，特别是在高延迟或高带宽的网络环境中。

然而，累计确认机制也有其局限性。最主要的是，如果一个数据包丢失，发送方可能需要重传所有未被确认的数据包，即使其中一些包已经被接收方正确接收。这可能导致网络上的数据包重复，从而浪费带宽。尽管存在这样的局限性，累计确认在许多实时通信系统和数据传输协议中因其高效性和简便性而被广泛采用。

4 报文设计

我此次实验报文和3-1一样，没有做修改，报文头如下：

```
1  #define DATA_SIZE 10000
2
3  struct Flag {
4      static constexpr uint16_t START = 0x1;
5      static constexpr uint16_t END = (0x1 << 1);
6      static constexpr uint16_t DATA = (0x1 << 2);
7      static constexpr uint16_t ACK = (0x1 << 3);
8      static constexpr uint16_t SYN = (0x1 << 4);
9      static constexpr uint16_t FIN = (0x1 << 5);
10 };
11
12 struct Header {
13     uint32_t seqNum;    // 序列号
14     uint32_t ackNum;    // 确认号
15     uint16_t length;    // 数据长度
16     uint16_t checksum;  // 校验和
17     uint16_t flags;     // 标志位
18 };
19
20 class UDPPacket {
21 private:
22     Header header;
```

```

23     char data[DATA_SIZE];
24     .....
25 };

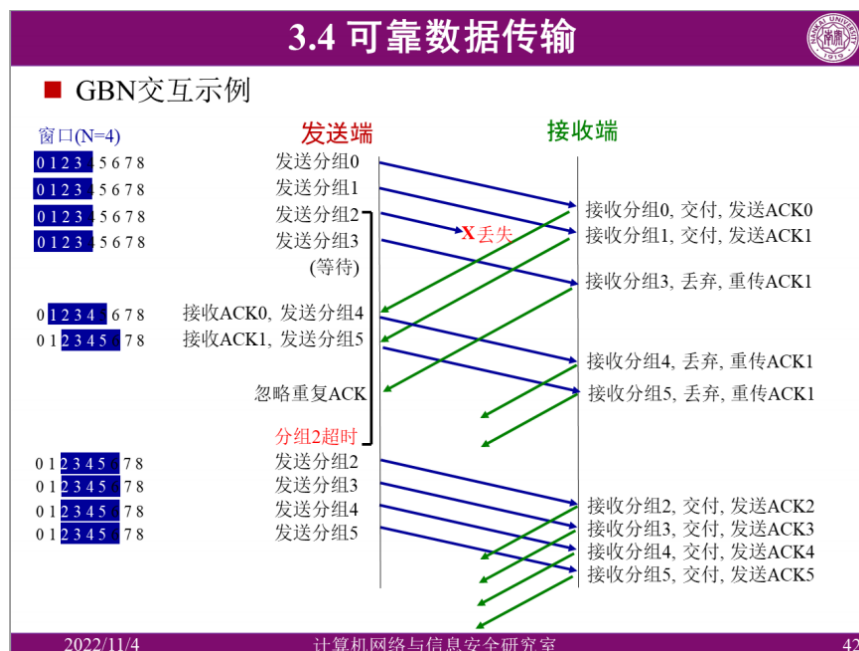
```

- **握手和挥手**使用FIN和SYN，ACK标志位，我使用了三次握手，四次挥手的机制，仿照TCP协议。
- **发送数据**：在握手成功后，发送端**首先发送一个置START位的包**，表明开始发送数据，然后开始发送数据，**发送数据使用DATA标志位，待数据都传输完毕后，发送一个置END位的包**，表明传输结束，然后开始挥手。

5 程序逻辑

我将从发送端和接收端两个方面对我程序的逻辑进行介绍。在我的代码中，**服务端即接收端，客户端即发送端**。

总体来说，我发送端和接收端都遵循下图的交互方式：



5.1 公共代码

服务端和客户端共用一个报文类，作为传输文件的协议。

我定义了**报文类**，在发送包，接收包的时候都可以使用，其中定义了反序列化和序列化函数，以及计算Checksum的函数：

```

1 class UDPPacket {
2 private:
3     Header header;
4     char data[DATA_SIZE];
5
6 public:
7

```

```

8     UDPPacket() {
9         std::memset(&header, 0, sizeof(header));
10        std::memset(data, 0, sizeof(data));
11    }
12
13    .....
14    .....
15
16    // 序列化
17    std::string serialize() const {
18        Header netHeader = header;
19        // 序列化之前, 转换为网络字节顺序
20        netHeader.seqNum = htonl(netHeader.seqNum);
21        netHeader.ackNum = htonl(netHeader.ackNum);
22        netHeader.length = htons(netHeader.length);
23
24        std::string serialized;
25        serialized.append(reinterpret_cast<const char*>(&netHeader),
sizeof(netHeader));
26        serialized.append(data, ntohs(netHeader.length));
27        return serialized;
28    }
29
30    // 反序列化
31    void deserialize(const std::string& serialized) {
32        std::memcpy(&header, serialized.data(), sizeof(header));
33        header.seqNum = ntohl(header.seqNum);
34        header.ackNum = ntohl(header.ackNum);
35        header.length = ntohs(header.length);
36
37        if (header.length <= DATA_SIZE) {
38            std::memcpy(data, serialized.data() + sizeof(header),
header.length);
39        }
40    }
41
42
43    // 计算检验和
44    uint16_t calChecksum() const {
45        uint32_t sum = 0;
46        UDPPacket tempPacket = *this;
47        tempPacket.header.checksum = 0; // 将checksum字段设置为0
48
49        const uint8_t* bytes = reinterpret_cast<const uint8_t*>
(&tempPacket.header);
50

```

```

51 // 确保转换为网络字节序
52 Header netHeader = tempPacket.header;
53 netHeader.seqNum = htonl(netHeader.seqNum);
54 netHeader.ackNum = htonl(netHeader.ackNum);
55 netHeader.length = htons(netHeader.length);
56 netHeader.checksum = htons(netHeader.checksum); // 这个字段已经是
0, 转换不影响
57 netHeader.flags = htons(netHeader.flags);
58
59 // 计算头部的校验和
60 for (size_t i = 0; i < sizeof(Header); i += 2) {
61     uint16_t word = bytes[i] << 8;
62     if (i + 1 < sizeof(Header)) {
63         word += bytes[i + 1];
64     }
65     sum += word;
66     if (sum >> 16) {
67         sum = (sum & 0xFFFF) + (sum >> 16);
68     }
69 }
70
71 // 计算数据部分的校验和
72 bytes = reinterpret_cast<const uint8_t*>(tempPacket.data);
73 for (size_t i = 0; i < ntohs(netHeader.length); i += 2) {
74     uint16_t word = bytes[i] << 8;
75     if (i + 1 < ntohs(netHeader.length)) {
76         word += bytes[i + 1];
77     }
78     sum += word;
79     if (sum >> 16) {
80         sum = (sum & 0xFFFF) + (sum >> 16);
81     }
82 }
83
84 return ~sum;
85 }
86 };

```

- **序列化**：其实也可以直接使用char数组来存所有的数据，包括报文头，报文数据体，但是这样的话无论是DEBUG还是代码可读性都会变得很差，因此我代码尽量使用类体来作为包体，那么**类作为一种数据结构，不能直接在网络上传输**，我们需要将其转化为“字节流”的形式，才能传输，因此引入了序列化的函数，来将报文头，数据合并成一个可以传输的数据。
- **反序列化**：与上面同理，服务端接收到报文后，这是一串字节流的数据，我们需要将其各个内容赋值到我的结构体里面，比如哪几个字节是序列号，哪几个字节是ACK？通过反序列

化，我们可以将收到的数据整合到类内，这样可以调用类函数来方便地对包进行查看，修改等各种操作，代码可读性也会得到很大的提升。

- **校验和**：通过计算报文头，数据的校验和，我们可以保证数据传输是正确的，因为我们会在接收端检验这个校验和是否正确。在其中，我们将**报文头转换为网络字节序**，这是因为不同的机器可能运行时内部字节序不同，我们将其转化为统一的字节序，可以保证数据传输的正确性。

为了提升传输显示，日志输出的美观性，我将一些打印函数封装到了发送端和接收端的**基类**：

```
1  enum Level { INFO, WARN, ERR, RECV, SEND, NOP };
2
3  class UDP {
4  public:
5      virtual void handshake() = 0;
6      virtual void waveHand() = 0;
7
8      std::string GetCurrTime() const {
9          std::string strTime = "";
10         time_t now;
11         time(&now);  // 获取当前时间
12         tm tmNow;
13         localtime_s(&tmNow, &now);
14         strTime += std::to_string(tmNow.tm_year + 1900) + "-";
15         strTime += std::to_string(tmNow.tm_mon + 1) + "-";
16         strTime += std::to_string(tmNow.tm_mday) + " ";
17         strTime += std::to_string(tmNow.tm_hour) + ":";
18         strTime += std::to_string(tmNow.tm_min) + ":";
19         strTime += std::to_string(tmNow.tm_sec);
20         return strTime;
21     }
22
23     // 打印包信息
24     void PrintPacketInfo(const UDPPacket& packet, Level lv) const {
25         const Header& hdr = packet.getHeader();
26         std::string flagStr = packet.flagsToString();
27         std::string info = "Packet - SeqNum: " +
28             std::to_string(hdr.seqNum) +
29             ", AckNum: " + std::to_string(hdr.ackNum) +
30             ", Checksum: " + std::to_string(hdr.checksum) +
31             ", Flags: " + flagStr;
32
33         if (packet.isFlagSet(Flag::DATA)) {
34             // 如果是数据包，添加数据长度信息
35             info += ", Data Length: " + std::to_string(hdr.length);
36         }
37         Print(info, lv);
38     }
```



```

37     }
38
39     // 打印信息
40     void Print(const std::string& info, Level lv = NOP) const {
41         HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
42         WORD saved_attributes;
43
44         // 保存当前的颜色设置
45         CONSOLE_SCREEN_BUFFER_INFO consoleInfo;
46         GetConsoleScreenBufferInfo(hConsole, &consoleInfo);
47         saved_attributes = consoleInfo.wAttributes;
48
49         std::cout << GetCurrTime() + " ";
50
51         // 设置新的颜色属性
52         switch (lv) {
53             case Level::INFO:
54                 SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
55                 std::cout << "[INFO] ";
56                 break;
57             case Level::WARN:
58                 SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN |
59 FOREGROUND_RED);
60                 std::cout << "[WARN] ";
61                 break;
62             case Level::ERR:
63                 SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
64                 std::cout << "[ERROR] ";
65                 break;
66             case Level::RECV:
67                 SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN |
68 FOREGROUND_BLUE);
69                 std::cout << "[RECV] ";
70                 break;
71             case Level::SEND:
72                 SetConsoleTextAttribute(hConsole, FOREGROUND_BLUE |
73 FOREGROUND_INTENSITY);
74                 std::cout << "[SEND] ";
75                 break;
76             default:
77                 SetConsoleTextAttribute(hConsole, saved_attributes); // 使用默认颜色
78                 break;
79         }
80         // 恢复原来的颜色设置
81         SetConsoleTextAttribute(hConsole, saved_attributes);

```

```

79         // 输出信息
80         std::cout << info << std::endl;
81     }
82 };

```

这些函数调用了系统函数来改变命令行的输出颜色，便于我们区分发送和接收，以及报错。在后文我们可以看到，发送端和接收端都继承了这个类。

5.2 发送端

发送端定义了 `UDPClient` 类：

```

1  #define BUFFER_SIZE (DATA_SIZE + sizeof(Header))
2
3  class UDPClient : public UDP {
4  public:
5      UDPClient(const std::string& serverIP, UINT serverPort, uint32_t
window_size);
6      ~UDPClient();
7
8      void SendFile(const std::string& filePath);
9
10 private:
11     std::string serverIP;
12     UINT serverPort;
13     SOCKET clientSocket;
14     sockaddr_in serverAddr;
15     uint32_t window_size;
16     bool isconnected = false;
17     uint32_t nextseq = 0;           // 下一个要发送的DATA包的序列号
18     uint32_t base = 0;             // 窗口基序列号
19     uint32_t ackNum = 0;           // 如果发送ACK，使用的ack值
20     uint64_t totalBytesSent = 0;    // 发送的总字节数
21     bool running = false;
22     ULONGLONG timer_start;
23     std::queue<UDPPacket> sendBuffer; // 存储已发送但尚未确认的数据包
24     std::mutex mtx;                 // 用于保护共享资源的互斥锁
25     static const ULONGLONG timeoutMs = 1000; // 超时时间
26
27     void handshake();
28     void waveHand();
29     void sendPacket(uint32_t flags, uint32_t seq, uint32_t ack = UINT_MAX,
const char* data = nullptr, uint16_t length = 0, bool resend = false);
30     void sendFileData(const std::string& filePath);
31     bool waitForPacket(uint32_t expectedFlag);
32     static DWORD WINAPI receiveAck(LPVOID pParam);
33     void Print(const std::string& info, Level lv = NOP);

```

```

34     void PrintPacketInfo(const UDPPacket& packet, Level lv);
35     void PrintsendBuffer();
36 };

```

首先定义了一些变量，如 `nextseq`，`base`，这些变量在GBN中用来操作滑动窗口，如 `base++` 则窗口向右滑一格，`nextseq` 则表明了下一个即将发送的未发送过的数据包。

• 发送缓冲区

值得注意的是我定义了一个队列：

```

1  std::queue<UDPPacket> sendBuffer;

```

这个队列充当发送缓冲区的作用，可以看到队列的内容是一个个包，包括包头，包体，我们如果需要重发，直接利用这个队列，从头到尾遍历一遍，将其中所有包发送即可。

关于发送缓存区的维护，在下文各个函数将会进行解释。

• 发送包(通用函数)

```

1  // 发送Packet
2  void UDPClient::sendPacket(uint32_t flags, uint32_t seq, uint32_t ack,
3  const char* data, uint16_t length, bool resend)
4  {
5      UDPPacket packet;
6      packet.setSeq(seq);      // 使用当前序列号
7      packet.setFlag(flags);
8      if (packet.isFlagSet(Flag::DATA)) {
9          packet.setData(data, length);
10         // 如果不是重发包则添加到发送缓冲区
11         if (!resend){
12             std::lock_guard<std::mutex> lock(mtx);
13             sendBuffer.push(packet);
14             totalBytesSent += length;
15         }
16     }
17     if (packet.isFlagSet(Flag::ACK)) {
18         packet.setAck(ack);
19     }
20     packet.setChecksum(packet.calChecksum()); // 计算校验和
21     // 打印发送的数据包信息
22     PrintPacketInfo(packet, SEND);
23
24     // 如果发送数据包，打印sendBuffer的信息
25     if (packet.isFlagSet(Flag::DATA)) PrintsendBuffer();
26

```

```

27 // 序列化发送数据包
28 std::string serialized = packet.serialize();
29 sendto(clientSocket, serialized.c_str(), serialized.size(), 0,
30        (struct sockaddr*)&serverAddr, sizeof(serverAddr));
31 }

```

可以看到，使用类而不是char数组作为包进行使用，使得代码变得更为清晰：

1. 首先根据参数设置序列号，设置标志位。
2. 如果是数据包，即DATA标志位置位，并且这个包不是重发包，则将其加入到发送缓存区中。这是因为，重发包的时候也需要调用这个函数，如果重发包也加入发送缓存区，那么它将出现两次，三次，造成缓存区超出窗口大小，越来越大，造成程序的死循环。
3. 在打印信息后，利用前面提到的序列化，将其转化为string类型，再转为char*，就可以直接发送了。

• 文件发送 & 超时重传

```

1 // 发送文件数据
2 void UDPClient::sendFileData(const std::string& filePath) {
3     // 发送 START 包
4     sendPacket(Flag::START, nextseq++);
5
6     totalBytesSent = 0; // 重置发送的总字节数
7     ULONGLONG startTime = GetTickCount64(); // 记录开始时间(区别于
timer_start)
8
9     std::ifstream file(filePath, std::ios::binary | std::ios::ate);
10    file.seekg(0, std::ios::beg);
11    char* const buffer = new char[DATA_SIZE];
12    bool eof = false;
13    running = true;
14    base = nextseq; //让base指向第一个发送的数据包
15    HANDLE hrecv = CreateThread(NULL, 0, receiveAck, this, 0, NULL); //
创建接收ACK线程
16
17    while (true) {
18        // 读取文件并发送包
19        while (nextseq < base + window_size && !eof) {
20            file.read(buffer, DATA_SIZE);
21            std::streamsize bytesRead = file.gcount();
22            eof = (bytesRead < DATA_SIZE);
23
24            if (base == nextseq) {
25                // 如果这是窗口中的第一个包，则重置定时器
26                std::lock_guard<std::mutex> lock(mtx);
27                timer_start = GetTickCount64();
28            }

```

```

29         sendPacket(Flag::DATA, nextseq++, 0, buffer,
static_cast<uint16_t>(bytesRead));
30     }
31
32     // 当发送缓冲区为空时结束
33     if (sendBuffer.empty()) break;
34
35     // 超时重传逻辑
36     if (GetTickCount64() - timer_start > timeoutMs) {
37         // 打印出base和sendbuffer内容
38         Print("Timeout, resend Packet in the Window! Current base: " +
std::to_string(base), WARN);
39         PrintsendBuffer();
40
41         // 从缓冲区中获取所有包(均为未确认)并重传, 实际上就是窗口内的包
42         std::queue<UDPPacket> tmp = sendBuffer;
43         while (!tmp.empty()){
44             UDPPacket packet = tmp.front();
45             Header pHead = packet.getHeader();
46             sendPacket(pHead.flags, pHead.seqNum, 0, packet.getData(),
pHead.length, true);
47             tmp.pop();
48         }
49     }
50 }
51
52 if (hrecv) CloseHandle(hrecv); // 关闭接收线程
53
54 {
55     std::lock_guard<std::mutex> lock(mtx);
56     running = false;
57 }
58
59 // 发送 END 包
60 sendPacket(Flag::END, nextseq++);
61 file.close();
62 delete[] buffer;
63
64 // 打印发送的总字节数和时间
65 ULONGLONG endTime = GetTickCount64();
66 double elapsed = static_cast<double>(endTime - startTime) / 1000.0;
67 Print("File: " + filePath, INFO);
68 Print("Bytes Sent: " + std::to_string(totalBytesSent) + " bytes",
INFO);
69 Print("Time Taken: " + std::to_string(elapsed) + " seconds", INFO);

```

```

70     Print("Average Speed: " + std::to_string(totalBytesSent / elapsed) + "
    bytes/s", INFO);
71 }

```

在 `sendFileData` 函数中，我们首先发送一个START包，以通知接收端文件传输即将开始。然后，我们进入一个循环，不断从文件中读取数据并发送。对于每个数据包，**我们都会检查是否达到了滑动窗口的上限**（通过比较 `nextseq` 和 `base + window_size`）。如果没有达到上限并且文件未读完，我们读取数据，创建一个数据包，并发送它。

简单地说，就是一次把窗口内的包发完，当然下文会提到我同时也开了接收ACK的线程，**如果窗口很大，发送的过程可能就已经收到ACK了，这时候我的多线程就发挥了作用。**

关键的一点是超时重传机制。我们设置了一个定时器（`timer_start`），每当窗口中的第一个包被发送时，它就会重置。如果在预定的超时时间（`timeoutMs`）内没有收到对该包的确认，我们会重传窗口中的所有包，而不仅仅是丢掉的包，这是通过复制发送缓冲区（`sendBuffer`）并遍历它来完成的。每个包都会被重新发送，直到收到确认或者再次超时。

此外，发送缓冲区的管理也很重要。每次发送一个新的数据包时，它就会被加入到发送缓冲区中。当收到一个包的确认时，该包和它之前的所有包都会从缓冲区中移除，因为我们采用了累计确认机制，这在下面会提到。

• 专门用来接收ACK的线程 & 累计确认

这是我用来接收ACK的线程函数，累计确认机制可以在这个函数里得以体现：

```

1  // 接收ACK线程
2  DWORD WINAPI UDPClient::receiveAck(LPVOID pParam){
3      UDPClient* client = (UDPClient*)pParam;
4      char* const buffer = new char[BUFFER_SIZE];
5      int addrLen = sizeof(client->serverAddr);
6      static uint16_t lastack = UINT16_MAX;    // 记录上一次收到的ACK，用于甄别
        重复ACK
7
8      while (client->running){
9          int recvLen = recvfrom(client->clientSocket, buffer, BUFFER_SIZE,
        0, (struct sockaddr*)&client->serverAddr, &addrLen);
10         if (recvLen > 0){
11             UDPPacket packet;
12             packet.deserialize(std::string(buffer, recvLen));
13             client->PrintPacketInfo(packet, RECV);
14
15             // 检查校验和
16             if (!packet.validChecksum()) {
17                 client->Print("Checksum failed for packet with seq: " +
        std::to_string(packet.getHeader().seqNum), WARN);
18                 continue;

```

```

19         }
20
21         // 如果是ACK包
22         if (packet.isFlagSet(Flag::ACK)){
23             uint32_t ack = packet.getHeader().ackNum;
24
25             // 第一次收到ACK包
26             if (lastack == UINT16_MAX){
27                 lastack = ack;
28             }
29
30             // 收到重复的ACK包
31             else if (ack <= lastack){
32                 client->Print("Recieve duplicate ACK packet with
ackNum: " + std::to_string(ack), WARN);
33             }
34
35             // 收到新的ACK包
36             else {
37                 {
38                     std::lock_guard<std::mutex> lock(client->mtx); // 锁定互斥锁
39                     client->ackNum = packet.getHeader().seqNum + 1; // 更新确认号
40                     client->timer_start = GetTickCount64(); // 重置定时器
41                 }
42
43                 // 累计确认机制, 老的ACK丢了不要紧, 最新的ACK发来了就行
44                 // 即ack有可能等于client->base, 也可以大于client->base
45                 while (ack > client->base) {
46                     std::lock_guard<std::mutex> lock(client->mtx);
47                     client->sendBuffer.pop(); // 从缓冲区中删除已确认的包
48                     client->base++;           // 更新窗口基序号
49                 }
50             }
51         }
52     }
53     Sleep(10);
54 }
55 client->Print("File is not sending, RecvThread close.", INFO);
56 delete[] buffer;
57 return 0;
58 }


```

为了不阻塞主线程，我们创建了一个独立的线程来专门处理ACK的接收。在 `receiveAck` 函数中，线程不断检查是否有新的数据包到达。一旦收到一个包，我们首先检查它的校验和，以确保数据的完整性。

如果是一个ACK包，我们会检查它的确认号（`ackNum`）。如果这个确认号是新的（即大于我们之前收到的最大确认号 `lastack`），我们就更新 `ackNum` 和 `timer_start`。然后，我们使用累计确认机制来更新窗口的基序列号 `base`。如果接收到的确认号小于或等于 `lastack`，**我们认为这是一个重复的ACK，并不进行任何操作。**

这个线程在整个文件传输过程中持续运行，直到主线程设置 `running` 标志为 `false`，表示文件已完全发送。最后，线程退出前清理资源并关闭。

通过这种方式，客户端能够不断发送数据包，同时独立地处理接收到的ACK，从而有效地实现滑动窗口协议和Go-Back-N重传机制，提高了数据传输的效率和可靠性。

 **对发送缓冲区的维护：**如果确认号是新的（即大于之前收到的任何确认号），线程将更新 `ackNum` 并重置超时计时器（`timer_start`）。然后，它将移除 `sendBuffer` 中所有已经被确认的包。具体来说，**队列中序列号小于或等于收到的确认号的所有包都被移除，因为累计确认意味着所有这些包都被正确接收**。这便是累计确认机制的实现。

5.3 接收端

△首先注意到一点，**接收窗口大小为1**，即我们可以在收到一个包后，在同一段代码后面对其进行处理，而不用另外开一个线程来进行异步处理，因为其在发送ACK的过程中不可能收到数据包，其收到数据包一定是在前一个ACK发送完毕之后，即“收发收发……”，而对比发送端，发送端需要在发送数据的同时兼顾接收ACK，所以做不到这点。

我定义了UDPServer类：

```
1  #define BUFFER_SIZE (DATA_SIZE + sizeof(Header))
2
3  class UDPServer : public UDP {
4  public:
5      UDPServer(UINT port);
6      ~UDPServer();
7
8      void Start();
9      void Stop();
10
11 private:
12     UINT port;
13     SOCKET serverSocket;
14     sockaddr_in serverAddr;
15     sockaddr_in clientAddr;
16     std::ofstream outFile;
17     bool isconnect = false;
18     uint32_t ackNum = 0;
19     uint32_t lastack = 0;
20     uint32_t currSeq = 0;
```

```

21     uint32_t exptSeq = 0;
22     uint64_t totalBytesRecv = 0; // 接收的总字节数
23
24     void receiveData();
25     void writeData(const char* data, uint16_t length);
26     void sendPacket(uint32_t flags, uint32_t seq, uint32_t ack = UINT_MAX,
const char* data = nullptr, uint16_t length = 0);
27     void openFile(const std::string& filename);
28     void closeFile();
29     void handshake();
30     void waveHand();
31     bool waitForPacket(uint32_t expectedFlag);
32     bool receivePacket(UDPPacket& packet);
33 };

```

在我们的Go-Back-N协议实现中，接收端是由 `UDPServer` 类承担的。与发送端（`UDPClient`）不同，接收端不需要处理并行的发送和接收操作，因为它主要负责接收数据包并发送ACK。这种设计简化了接收端的逻辑，使其不需要额外的线程来异步处理数据。

• 关键属性

`UDPServer` 类具备以下关键属性：

1. `serverSocket`：用于UDP通信的socket。
2. `serverAddr`和`clientAddr`：分别用于存储服务器和客户端的地址信息。
3. `outFile`：用于写入接收到的文件数据。
4. `isconnect`：标记当前是否处于连接状态。
5. `ackNum`和`lastack`：分别用于存储当前的确认号和上一个确认号。
6. `currSeq`和`exptSeq`：分别用于存储当前的序列号和期望接收的下一个数据包的序列号。
7. `totalBytesRecv`：记录接收到的总字节数。

其中，两个重点成员是 `exptSeq` 和 `lastack`。

1. `exptSeq`

`exptSeq` 用来记录接收端期望收到的下一个数据包的序列号。这是滑动窗口协议中的关键部分，因为它决定了接收端能够接收哪些数据包。当接收端收到一个序列号匹配 `exptSeq` 的数据包时，它会处理该数据包并将 `exptSeq` 递增，这样就可以接收下一个期望的数据包。

2. `lastack`

`lastack` 用于记录最后一个发送的确认号（ACK）。在Go-Back-N协议中，如果接收端接收到一个序列号大于 `exptSeq` 的数据包，它意味着有数据包丢失。在这种情况下，接收端将重传上一个确认号 `lastack`，提示发送端重新发送丢失的数据包及其之后的所有数据包。

- 通用函数：接收数据包并模拟丢包和延时

```
1  bool UDPServer::receivePacket(UDPPacket& packet) {
2      // 模拟丢包和延时处理参数
3      const int fixedDelay = 0;
4      const int drop_every = 33;
5      static std::default_random_engine generator_drop, generator_delay;
6      static std::uniform_int_distribution<int> delayDistribution(0,
drop_every); // 每drop_every个包中随机选择一个进行延时
7      static std::uniform_int_distribution<int> lossDistribution(0,
drop_every); // 每drop_every个包中随机选择一个进行丢包
8      static bool drop = false;
9      static bool delay = false;
10
11     char* const buffer = new char[BUFFER_SIZE];
12     int addrLen = sizeof(clientAddr);
13     int recvLen = recvfrom(serverSocket, buffer, BUFFER_SIZE, 0, (struct
sockaddr*)&clientAddr, &addrLen);
14
15     if (recvLen > 0) {
16         packet.deserialize(std::string(buffer, recvLen));
17         Header pktHeader = packet.getHeader();
18
19         // 检查数据包的检验和
20         if (!packet.validChecksum()) {
21             Print("Checksum failed for packet with seq: " +
std::to_string(pktHeader.seqNum), WARN);
22             return false;
23         }
24
25         // 对数据包模拟丢包和延时
26         if (packet.isFlagSet(Flag::DATA))
27         {
28             drop = (lossDistribution(generator_drop) == 0);
29             delay = (delayDistribution(generator_delay) == 0);
30             // 随机选择Data包进行丢包
31             if (drop) {
32                 Print("Simulating packet loss for packet seq: " +
std::to_string(pktHeader.seqNum), WARN);
33                 return false; // 不处理该包, 模拟丢包
34             }
35             // 随机选择Data包进行延时
36             if (delay) {
37                 Print("Delaying ACK for packet seq: " +
std::to_string(pktHeader.seqNum), WARN);
38                 Sleep(fixedDelay);

```

```

39         }
40     }
41     lastack = ackNum;                // 保存上一次的ACK
42     ackNum = pktHeader.seqNum + 1;   // 更新ACK
43     // 打印接收到的数据包信息
44     PrintPacketInfo(packet, RECV);
45     return true;
46 }
47 else if (recvLen == 0 || WSAGetLastError() != WSAEWOULDBLOCK) {
48     Print("recvfrom() failed with error code: " +
std::to_string(WSAGetLastError()), ERR);
49 }
50 delete[] buffer;
51 return false;
52 }

```

该函数是服务器端接收数据包的前端部分，采用阻塞方法等待数据包的到来。这个方法确保了在数据包到达之前，服务器不会进行其他操作，从而能够专注于处理每一个接收到的数据包。

1. **阻塞式接收数据**：使用 `recvfrom` 函数以阻塞方式接收数据包。这意味着，如果没有数据包到达，函数会停在那里等待，直到有数据包到来。这种方式简化了逻辑，因为不需要考虑异步处理或轮询等复杂的数据接收逻辑。
2. **反序列化和校验**：一旦接收到数据包，函数首先进行反序列化操作，将网络字节流转换为 `UDPPacket` 对象。接着，进行校验和验证以确保数据包的完整性和准确性。
3. **模拟网络条件**：为模拟真实网络环境中可能出现的数据丢失和延迟，函数内部实现了数据包丢失和延时的逻辑。通过随机数生成器，它决定是否暂时忽略（延迟）或完全丢弃（丢包）当前处理的数据包。
4. **ACK更新和错误处理**：在校验和验证通过后，函数会更新 `ackNum` 为接收到的数据包序号加一，准备发送确认信息。如果 `recvfrom` 出现错误或接收到的数据包校验和不匹配，函数会进行相应的错误处理并记录日志。

● 核心函数：处理数据包

```

1 // 监听并接收数据
2 void UDPServer::receiveData() {
3     bool receivingFile = false;
4     ULONGLONG startTime = 0;
5     ULONGLONG endTime = 0;
6
7     while (true) {
8         UDPPacket packet;
9         if (receivePacket(packet)) {
10             const Header& pktHeader = packet.getHeader();
11
12             // 检查是否是文件传输的开始
13             if (packet.isFlagSet(Flag::START)) {
14                 openFile("received_file.bin");

```

```

15         receivingFile = true;
16         totalBytesRecv = 0; // 重置接收的总字节数
17         exptSeq = pktHeader.seqNum + 1; // 重置期望的序列号
18         Print("Start receiving file.", INFO);
19         startTime = GetTickCount64(); // 记录开始时间
20         continue;
21     }
22
23     // 如果是数据包, 并且已经开始接收文件
24     if (packet.isFlagSet(Flag::DATA) && receivingFile) {
25         // 检查是否是期望seq
26         uint32_t recvSeq = pktHeader.seqNum;
27         if (recvSeq > exptSeq) { // 接收到的包序列号大于期望的序
列号, 说明有包丢失, 重传ACK
28             Print("Received out of order packet with seq: " +
std::to_string(recvSeq), WARN);
29             ackNum = lastack; // 退回ACK的值
30             sendPacket(Flag::ACK, currSeq++, ackNum); // 重传ACK
31             continue;
32         }
33         else if (recvSeq < exptSeq) { // 重复接收到的包, 回复老的
ACK即可
34             Print("Received duplicate packet with seq: " +
std::to_string(recvSeq), WARN);
35             sendPacket(Flag::ACK, currSeq++, recvSeq + 1); // 回
复老的ACK
36             continue;
37         }
38         // 更新期望的序列号
39         exptSeq = recvSeq + 1;
40         // 写入数据
41         writeData(packet.getData(), pktHeader.length);
42         totalBytesRecv += pktHeader.length;
43         // 发送ACK
44         sendPacket(Flag::ACK, currSeq++, ackNum);
45     }
46
47     // 检查是否是文件传输的结束
48     if (packet.isFlagSet(Flag::END)) {
49         endTime = GetTickCount64(); // 记录结束时间
50         closeFile();
51         receivingFile = false;
52         double elapsed = static_cast<double>(endTime - startTime)
/ 1000.0;
53         Print("Bytes Recv: " + std::to_string(totalBytesRecv) + "
bytes", INFO);

```

```

54         Print("Time Taken: " + std::to_string(elapsed) + "
seconds", INFO);
55         Print("Average Speed: " + std::to_string(totalBytesRecv /
elapsed) + " bytes/s", INFO);
56         continue;
57     }
58
59     // 处理握手请求
60     if (packet.isFlagSet(Flag::SYN)) {
61         handshake();
62         continue;
63     }
64
65     // 检查是否是挥手请求 (FIN)
66     if (packet.isFlagSet(Flag::FIN)) {
67         waveHand();
68         continue;
69     }
70 }
71 }
72 }

```

在 `UDPServer::receiveData()` 函数中，服务器端不断监听并处理接收到的数据包。这个函数是UDP服务器端逻辑的核心，涵盖了对不同类型数据包的处理流程。重点关注的是对 `DATA` 类型数据包的处理，其中包含了几个关键的判断和操作：

1. 文件传输开始的识别：

- 通过检查 `Flag::START` 标志，函数识别文件传输开始的信号。
- 当收到起始信号，函数会打开文件写入流，准备接收数据，并将相关状态变量（如接收的字节总数、期望的序列号）重置。

2. 对 `DATA` 数据包的处理：

- 当收到 `DATA` 类型的数据包时，首先判断其序列号（`seqNum`）与期望序列号（`exptSeq`）的关系。这里有三种情况：
 - **序列号大于期望序列号**：表示接收到了非顺序的数据包，可能是由于中间有数据包丢失。在这种情况下，服务器将回退到上一个确认的序列号（`lastack`）并发送ACK，这是Go-Back-N协议的特性，即重传ACK以提示发送端重传丢失的包。
 - **序列号小于期望序列号**：表示收到了重复的数据包。这可能是因为之前的ACK丢失或延迟到达，导致发送端重传了已经接收的包。在这种情况下，服务器将回复老的ACK。
 - **序列号等于期望序列号**：这是最理想的情况，意味着数据包按顺序到达。服务器将数据写入文件，并更新接收的总字节数。同时，期望序列号 `exptSeq` 更新为下一个序列号，并发送相应的ACK。

3. 文件传输结束的处理:

- 通过检查 `Flag::END` 标志，函数判断文件传输是否结束。一旦收到结束信号，关闭文件写入流，并打印接收文件的相关信息，如接收的总字节数、总用时和平均速率。

4. 握手和挥手:

- 对于握手 (SYN) 和挥手 (FIN) 请求，函数调用相应的 `handshake()` 和 `waveHand()` 方法来处理。

此函数的关键在于如何处理 `DATA` 类型的数据包，特别是如何根据序列号判断数据包的状态 (是否丢失、重复或按顺序接收)，并采取相应的措施。这种处理机制确保了数据传输的可靠性和顺序性，是UDP协议在不可靠传输层上实现可靠数据传输的关键。通过精确地处理每个数据包，服务器能够有效地接收文件，同时通过发送ACK通知客户端数据包的接收状态，从而支持Go-Back-N协议的实现。

6 运行结果

我使用代码内部对丢包和延时的模拟，并未使用路由器。以下将对四个文件分别运行，首先设置一些参数:

参数	值
丢包率	3%
延时	50ms
每个数据包的数据大小 10000字节	
窗口大小	16

这里延时是对包的随机延时，并不是每个包都有进行延时，是概率性操作。

- 1.jpg


```

The server port: 12720
Enter the Window Size: 16
Enter the path of the file to send: 1.jpg
2023-12-1 23:33:54 [INFO] File opened successfully: 1.jpg
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 0, AckNum: 0, Checksum: 61439, Flags: SYN
2023-12-1 23:33:54 [RECV] Packet - SeqNum: 0, AckNum: 1, Checksum: 59135, Flags: ACK SYN
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 1, AckNum: 1, Checksum: 62975, Flags: ACK
2023-12-1 23:33:54 [INFO] Handshake successful.
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 2, AckNum: 0, Checksum: 64767, Flags: START
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 3, AckNum: 0, Checksum: 7657, Flags: DATA, Data Length: 10000
2023-12-1 23:33:54 [INFO] SendBuffer: 3
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 4, AckNum: 0, Checksum: 42644, Flags: DATA, Data Length: 10000
2023-12-1 23:33:54 [INFO] SendBuffer: 3 4
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 5, AckNum: 0, Checksum: 36778, Flags: DATA, Data Length: 10000
2023-12-1 23:33:54 [INFO] SendBuffer: 3 4 5
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 6, AckNum: 0, Checksum: 11908, Flags: DATA, Data Length: 10000
2023-12-1 23:33:54 [INFO] SendBuffer: 3 4 5 6
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 7, AckNum: 0, Checksum: 30777, Flags: DATA, Data Length: 10000
2023-12-1 23:33:54 [INFO] SendBuffer: 3 4 5 6 7
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 8, AckNum: 0, Checksum: 16067, Flags: DATA, Data Length: 10000
2023-12-1 23:33:54 [INFO] SendBuffer: 3 4 5 6 7 8
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 9, AckNum: 0, Checksum: 13421, Flags: DATA, Data Length: 10000
2023-12-1 23:33:54 [INFO] SendBuffer: 3 4 5 6 7 8 9
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 10, AckNum: 0, Checksum: 17717, Flags: DATA, Data Length: 10000
2023-12-1 23:33:54 [INFO] SendBuffer: 3 4 5 6 7 8 9 10
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 11, AckNum: 0, Checksum: 47122, Flags: DATA, Data Length: 10000
2023-12-1 23:33:54 [INFO] SendBuffer: 3 4 5 6 7 8 9 10 11
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 12, AckNum: 0, Checksum: 30594, Flags: DATA, Data Length: 10000
2023-12-1 23:33:54 [INFO] SendBuffer: 3 4 5 6 7 8 9 10 11 12
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 13, AckNum: 0, Checksum: 8434, Flags: DATA, Data Length: 10000

```

```

2023-12-1 23:33:54 [SEND] Packet - SeqNum: 24, AckNum: 27, Checksum: 50431, Flags: ACK
2023-12-1 23:33:54 [RECV] Packet - SeqNum: 27, AckNum: 0, Checksum: 52449, Flags: DATA, Data Length: 10000
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 25, AckNum: 28, Checksum: 49919, Flags: ACK
2023-12-1 23:33:54 [RECV] Packet - SeqNum: 28, AckNum: 0, Checksum: 38804, Flags: DATA, Data Length: 10000
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 26, AckNum: 29, Checksum: 49407, Flags: ACK
2023-12-1 23:33:54 [RECV] Packet - SeqNum: 29, AckNum: 0, Checksum: 43850, Flags: DATA, Data Length: 10000
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 27, AckNum: 30, Checksum: 48895, Flags: ACK
2023-12-1 23:33:54 [RECV] Packet - SeqNum: 30, AckNum: 0, Checksum: 46250, Flags: DATA, Data Length: 10000
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 28, AckNum: 31, Checksum: 48383, Flags: ACK
2023-12-1 23:33:54 [RECV] Packet - SeqNum: 31, AckNum: 0, Checksum: 56950, Flags: DATA, Data Length: 10000
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 29, AckNum: 32, Checksum: 47871, Flags: ACK
2023-12-1 23:33:54 [RECV] Packet - SeqNum: 32, AckNum: 0, Checksum: 12951, Flags: DATA, Data Length: 10000
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 30, AckNum: 33, Checksum: 47359, Flags: ACK
2023-12-1 23:33:54 [RECV] Packet - SeqNum: 33, AckNum: 0, Checksum: 33753, Flags: DATA, Data Length: 10000
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 31, AckNum: 34, Checksum: 46847, Flags: ACK
2023-12-1 23:33:54 [WARN] Simulating packet loss for packet seq: 34
2023-12-1 23:33:54 [RECV] Packet - SeqNum: 35, AckNum: 0, Checksum: 54838, Flags: DATA, Data Length: 10000
2023-12-1 23:33:54 [WARN] Received out of order packet with seq: 35
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 32, AckNum: 34, Checksum: 46591, Flags: ACK
2023-12-1 23:33:54 [RECV] Packet - SeqNum: 36, AckNum: 0, Checksum: 30634, Flags: DATA, Data Length: 10000
2023-12-1 23:33:54 [WARN] Received out of order packet with seq: 36
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 33, AckNum: 34, Checksum: 46335, Flags: ACK
2023-12-1 23:33:54 [RECV] Packet - SeqNum: 37, AckNum: 0, Checksum: 65172, Flags: DATA, Data Length: 10000
2023-12-1 23:33:54 [WARN] Received out of order packet with seq: 37
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 34, AckNum: 34, Checksum: 46079, Flags: ACK
2023-12-1 23:33:54 [RECV] Packet - SeqNum: 38, AckNum: 0, Checksum: 44656, Flags: DATA, Data Length: 10000
2023-12-1 23:33:54 [WARN] Received out of order packet with seq: 38
2023-12-1 23:33:54 [SEND] Packet - SeqNum: 35, AckNum: 34, Checksum: 45823, Flags: ACK
2023-12-1 23:33:54 [RECV] Packet - SeqNum: 39, AckNum: 0, Checksum: 50168, Flags: DATA, Data Length: 10000
2023-12-1 23:33:54 [WARN] Received out of order packet with seq: 39

```

```

2023-12-1 23:34:6 [SEND] Packet - SeqNum: 299, AckNum: 182, Checksum: 5886, Flags: ACK
2023-12-1 23:34:6 [RECV] Packet - SeqNum: 182, AckNum: 0, Checksum: 44060, Flags: DATA, Data Length: 10000
2023-12-1 23:34:6 [SEND] Packet - SeqNum: 300, AckNum: 183, Checksum: 5374, Flags: ACK
2023-12-1 23:34:6 [RECV] Packet - SeqNum: 183, AckNum: 0, Checksum: 30067, Flags: DATA, Data Length: 10000
2023-12-1 23:34:6 [SEND] Packet - SeqNum: 301, AckNum: 184, Checksum: 4862, Flags: ACK
2023-12-1 23:34:6 [RECV] Packet - SeqNum: 184, AckNum: 0, Checksum: 8667, Flags: DATA, Data Length: 10000
2023-12-1 23:34:6 [SEND] Packet - SeqNum: 302, AckNum: 185, Checksum: 4350, Flags: ACK
2023-12-1 23:34:6 [RECV] Packet - SeqNum: 185, AckNum: 0, Checksum: 61253, Flags: DATA, Data Length: 10000
2023-12-1 23:34:6 [SEND] Packet - SeqNum: 303, AckNum: 186, Checksum: 3838, Flags: ACK
2023-12-1 23:34:6 [RECV] Packet - SeqNum: 186, AckNum: 0, Checksum: 42198, Flags: DATA, Data Length: 10000
2023-12-1 23:34:6 [SEND] Packet - SeqNum: 304, AckNum: 187, Checksum: 3326, Flags: ACK
2023-12-1 23:34:6 [RECV] Packet - SeqNum: 187, AckNum: 0, Checksum: 7757, Flags: DATA, Data Length: 10000
2023-12-1 23:34:6 [SEND] Packet - SeqNum: 305, AckNum: 188, Checksum: 2814, Flags: ACK
2023-12-1 23:34:6 [RECV] Packet - SeqNum: 188, AckNum: 0, Checksum: 4143, Flags: DATA, Data Length: 7353
2023-12-1 23:34:6 [SEND] Packet - SeqNum: 306, AckNum: 189, Checksum: 2302, Flags: ACK
2023-12-1 23:34:7 [RECV] Packet - SeqNum: 189, AckNum: 0, Checksum: 16639, Flags: END
2023-12-1 23:34:7 [INFO] File received and saved.
2023-12-1 23:34:7 [INFO] Bytes Recv: 1857353 bytes
2023-12-1 23:34:7 [INFO] Time Taken: 12.984000 seconds
2023-12-1 23:34:7 [INFO] Average Speed: 143049.368453 bytes/s
2023-12-1 23:34:7 [RECV] Packet - SeqNum: 190, AckNum: 0, Checksum: 8703, Flags: FIN
2023-12-1 23:34:7 [SEND] Packet - SeqNum: 307, AckNum: 191, Checksum: 1534, Flags: ACK
2023-12-1 23:34:7 [SEND] Packet - SeqNum: 308, AckNum: 191, Checksum: 58621, Flags: ACK FIN
2023-12-1 23:34:7 [RECV] Packet - SeqNum: 191, AckNum: 309, Checksum: 1022, Flags: ACK
2023-12-1 23:34:7 [INFO] Wavehand successful.

```

```

2023-12-1 23:34:5 [RECV] Packet - SeqNum: 275, AckNum: 169, Checksum: 15358, Flags: ACK
2023-12-1 23:34:5 [RECV] Packet - SeqNum: 276, AckNum: 169, Checksum: 15102, Flags: ACK
2023-12-1 23:34:5 [RECV] Packet - SeqNum: 277, AckNum: 169, Checksum: 14846, Flags: ACK
2023-12-1 23:34:5 [RECV] Packet - SeqNum: 278, AckNum: 169, Checksum: 14590, Flags: ACK
2023-12-1 23:34:5 [RECV] Packet - SeqNum: 279, AckNum: 169, Checksum: 14334, Flags: ACK
2023-12-1 23:34:5 [RECV] Packet - SeqNum: 280, AckNum: 169, Checksum: 14078, Flags: ACK
2023-12-1 23:34:5 [RECV] Packet - SeqNum: 281, AckNum: 169, Checksum: 13822, Flags: ACK
2023-12-1 23:34:5 [RECV] Packet - SeqNum: 282, AckNum: 169, Checksum: 13566, Flags: ACK
2023-12-1 23:34:5 [RECV] Packet - SeqNum: 283, AckNum: 169, Checksum: 13310, Flags: ACK
2023-12-1 23:34:5 [RECV] Packet - SeqNum: 284, AckNum: 169, Checksum: 13054, Flags: ACK
2023-12-1 23:34:5 [RECV] Packet - SeqNum: 285, AckNum: 169, Checksum: 12798, Flags: ACK
2023-12-1 23:34:5 [RECV] Packet - SeqNum: 286, AckNum: 169, Checksum: 12542, Flags: ACK
2023-12-1 23:34:6 [WARN] Timeout, resend Packet in the Window! Current base: 169
2023-12-1 23:34:6 [INFO] SendBuffer: 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184
2023-12-1 23:34:6 [SEND] Packet - SeqNum: 169, AckNum: 0, Checksum: 31863, Flags: DATA, Data Length: 10000
2023-12-1 23:34:6 [INFO] SendBuffer: 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184
2023-12-1 23:34:6 [SEND] Packet - SeqNum: 170, AckNum: 0, Checksum: 10502, Flags: DATA, Data Length: 10000
2023-12-1 23:34:6 [INFO] SendBuffer: 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184
2023-12-1 23:34:6 [SEND] Packet - SeqNum: 171, AckNum: 0, Checksum: 46549, Flags: DATA, Data Length: 10000
2023-12-1 23:34:6 [INFO] SendBuffer: 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184
2023-12-1 23:34:6 [SEND] Packet - SeqNum: 172, AckNum: 0, Checksum: 10178, Flags: DATA, Data Length: 10000
2023-12-1 23:34:6 [INFO] SendBuffer: 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184
2023-12-1 23:34:6 [SEND] Packet - SeqNum: 173, AckNum: 0, Checksum: 56596, Flags: DATA, Data Length: 10000
2023-12-1 23:34:6 [INFO] SendBuffer: 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184
2023-12-1 23:34:6 [SEND] Packet - SeqNum: 174, AckNum: 0, Checksum: 42605, Flags: DATA, Data Length: 10000
2023-12-1 23:34:6 [INFO] SendBuffer: 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184
2023-12-1 23:34:6 [SEND] Packet - SeqNum: 175, AckNum: 0, Checksum: 37104, Flags: DATA, Data Length: 10000
2023-12-1 23:34:6 [INFO] SendBuffer: 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184
2023-12-1 23:34:6 [SEND] Packet - SeqNum: 176, AckNum: 0, Checksum: 32946, Flags: DATA, Data Length: 10000
2023-12-1 23:34:6 [INFO] SendBuffer: 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184

```

上述黄色的WARN表明丢包启动，触发了超时重传机制，可以看到发送端重新传输了整个发送缓冲区的内容，而接收端因为序列号对不上也“无视”了丢的数据包之后的数据包，直到重传。

可以看到最后传输成功，文件Byte数量正确。

• 2.jpg

在接下来三个文件，由于第一个文件已经展示了超时重传和丢包，下面截图最终结果即可：

```

2023-12-1 23:39:30 [SEND] Packet - SeqNum: 834, AckNum: 584, Checksum: 28154, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 584, AckNum: 0, Checksum: 39514, Flags: DATA, Data Length: 10000
2023-12-1 23:39:30 [SEND] Packet - SeqNum: 835, AckNum: 585, Checksum: 27642, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 585, AckNum: 0, Checksum: 22779, Flags: DATA, Data Length: 10000
2023-12-1 23:39:30 [SEND] Packet - SeqNum: 836, AckNum: 586, Checksum: 27130, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 586, AckNum: 0, Checksum: 57472, Flags: DATA, Data Length: 10000
2023-12-1 23:39:30 [SEND] Packet - SeqNum: 837, AckNum: 587, Checksum: 26618, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 587, AckNum: 0, Checksum: 33381, Flags: DATA, Data Length: 10000
2023-12-1 23:39:30 [SEND] Packet - SeqNum: 838, AckNum: 588, Checksum: 26106, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 588, AckNum: 0, Checksum: 38822, Flags: DATA, Data Length: 10000
2023-12-1 23:39:30 [SEND] Packet - SeqNum: 839, AckNum: 589, Checksum: 25594, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 589, AckNum: 0, Checksum: 24601, Flags: DATA, Data Length: 10000
2023-12-1 23:39:30 [SEND] Packet - SeqNum: 840, AckNum: 590, Checksum: 25082, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 590, AckNum: 0, Checksum: 31334, Flags: DATA, Data Length: 10000
2023-12-1 23:39:30 [SEND] Packet - SeqNum: 841, AckNum: 591, Checksum: 24570, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 591, AckNum: 0, Checksum: 14877, Flags: DATA, Data Length: 10000
2023-12-1 23:39:30 [SEND] Packet - SeqNum: 842, AckNum: 592, Checksum: 24058, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 592, AckNum: 0, Checksum: 49011, Flags: DATA, Data Length: 8505
2023-12-1 23:39:30 [SEND] Packet - SeqNum: 843, AckNum: 593, Checksum: 23546, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 593, AckNum: 0, Checksum: 44285, Flags: END
2023-12-1 23:39:30 [INFO] File received and saved.
2023-12-1 23:39:30 [INFO] Bytes Recv: 5898505 bytes
2023-12-1 23:39:30 [INFO] Time Taken: 30.282000 seconds
2023-12-1 23:39:30 [INFO] Average Speed: 194785.846377 bytes/s
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 594, AckNum: 0, Checksum: 36349, Flags: FIN
2023-12-1 23:39:30 [SEND] Packet - SeqNum: 844, AckNum: 595, Checksum: 22778, Flags: ACK
2023-12-1 23:39:31 [SEND] Packet - SeqNum: 845, AckNum: 595, Checksum: 14330, Flags: ACK FIN
2023-12-1 23:39:31 [RCV] Packet - SeqNum: 595, AckNum: 846, Checksum: 22266, Flags: ACK
2023-12-1 23:39:31 [INFO] Wavehand successful.

```

```

2023-12-1 23:39:30 [SEND] Packet - SeqNum: 592, AckNum: 0, Checksum: 49011, Flags: DATA, Data Length: 8505
2023-12-1 23:39:30 [INFO] SendBuffer: 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 828, AckNum: 578, Checksum: 31226, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 829, AckNum: 579, Checksum: 30714, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 830, AckNum: 580, Checksum: 30202, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 831, AckNum: 581, Checksum: 29690, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 832, AckNum: 582, Checksum: 29178, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 833, AckNum: 583, Checksum: 28666, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 834, AckNum: 584, Checksum: 28154, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 835, AckNum: 585, Checksum: 27642, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 836, AckNum: 586, Checksum: 27130, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 837, AckNum: 587, Checksum: 26618, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 838, AckNum: 588, Checksum: 26106, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 839, AckNum: 589, Checksum: 25594, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 840, AckNum: 590, Checksum: 25082, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 841, AckNum: 591, Checksum: 24570, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 842, AckNum: 592, Checksum: 24058, Flags: ACK
2023-12-1 23:39:30 [RCV] Packet - SeqNum: 843, AckNum: 593, Checksum: 23546, Flags: ACK
2023-12-1 23:39:30 [SEND] Packet - SeqNum: 593, AckNum: 0, Checksum: 44285, Flags: END
2023-12-1 23:39:30 [INFO] File: 2.jpg
2023-12-1 23:39:30 [INFO] Bytes Sent: 5898505 bytes
2023-12-1 23:39:30 [INFO] Time Taken: 30.282000 seconds
2023-12-1 23:39:30 [INFO] Average Speed: 194785.846377 bytes/s
2023-12-1 23:39:30 [SEND] Packet - SeqNum: 594, AckNum: 0, Checksum: 36349, Flags: FIN
2023-12-1 23:39:31 [INFO] File is not sending, RecvThread close.
2023-12-1 23:39:31 [RCV] Packet - SeqNum: 844, AckNum: 595, Checksum: 22778, Flags: ACK
2023-12-1 23:39:31 [RCV] Packet - SeqNum: 845, AckNum: 595, Checksum: 14330, Flags: ACK FIN
2023-12-1 23:39:31 [SEND] Packet - SeqNum: 595, AckNum: 846, Checksum: 22266, Flags: ACK
2023-12-1 23:39:31 [INFO] Wavehand successful.
Enter the path of the file to send:

```

- 3.jpg


```

2023-12-1 23:41:16 [SEND] Packet - SeqNum: 1688, AckNum: 1191, Checksum: 47348, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1191, AckNum: 0, Checksum: 60726, Flags: DATA, Data Length: 10000
2023-12-1 23:41:16 [SEND] Packet - SeqNum: 1689, AckNum: 1192, Checksum: 46836, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1192, AckNum: 0, Checksum: 52349, Flags: DATA, Data Length: 10000
2023-12-1 23:41:16 [SEND] Packet - SeqNum: 1690, AckNum: 1193, Checksum: 46324, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1193, AckNum: 0, Checksum: 39201, Flags: DATA, Data Length: 10000
2023-12-1 23:41:16 [SEND] Packet - SeqNum: 1691, AckNum: 1194, Checksum: 45812, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1194, AckNum: 0, Checksum: 59787, Flags: DATA, Data Length: 10000
2023-12-1 23:41:16 [SEND] Packet - SeqNum: 1692, AckNum: 1195, Checksum: 45300, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1195, AckNum: 0, Checksum: 15756, Flags: DATA, Data Length: 10000
2023-12-1 23:41:16 [SEND] Packet - SeqNum: 1693, AckNum: 1196, Checksum: 44788, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1196, AckNum: 0, Checksum: 39496, Flags: DATA, Data Length: 10000
2023-12-1 23:41:16 [SEND] Packet - SeqNum: 1694, AckNum: 1197, Checksum: 44276, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1197, AckNum: 0, Checksum: 33073, Flags: DATA, Data Length: 10000
2023-12-1 23:41:16 [SEND] Packet - SeqNum: 1695, AckNum: 1198, Checksum: 43764, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1198, AckNum: 0, Checksum: 50986, Flags: DATA, Data Length: 10000
2023-12-1 23:41:16 [SEND] Packet - SeqNum: 1696, AckNum: 1199, Checksum: 43252, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1199, AckNum: 0, Checksum: 49538, Flags: DATA, Data Length: 8994
2023-12-1 23:41:16 [SEND] Packet - SeqNum: 1697, AckNum: 1200, Checksum: 42740, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1200, AckNum: 0, Checksum: 19963, Flags: END
2023-12-1 23:41:16 [INFO] File received and saved.
2023-12-1 23:41:16 [INFO] Bytes Recv: 11968994 bytes
2023-12-1 23:41:16 [INFO] Time Taken: 62.422000 seconds
2023-12-1 23:41:16 [INFO] Average Speed: 191743.199513 bytes/s
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1201, AckNum: 0, Checksum: 12027, Flags: FIN
2023-12-1 23:41:16 [SEND] Packet - SeqNum: 1698, AckNum: 1202, Checksum: 41972, Flags: ACK
2023-12-1 23:41:16 [SEND] Packet - SeqNum: 1699, AckNum: 1202, Checksum: 33524, Flags: ACK FIN
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1202, AckNum: 1700, Checksum: 41460, Flags: ACK
2023-12-1 23:41:16 [INFO] Wavehand successful.

```

```

2023-12-1 23:41:16 [SEND] Packet - SeqNum: 1199, AckNum: 0, Checksum: 49538, Flags: DATA, Data Length: 8994
2023-12-1 23:41:16 [INFO] SendBuffer: 1184 1185 1186 1187 1188 1189 1190 1191 1192 1193 1194 1195 1196 1197
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1682, AckNum: 1185, Checksum: 50420, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1683, AckNum: 1186, Checksum: 49908, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1684, AckNum: 1187, Checksum: 49396, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1685, AckNum: 1188, Checksum: 48884, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1686, AckNum: 1189, Checksum: 48372, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1687, AckNum: 1190, Checksum: 47860, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1688, AckNum: 1191, Checksum: 47348, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1689, AckNum: 1192, Checksum: 46836, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1690, AckNum: 1193, Checksum: 46324, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1691, AckNum: 1194, Checksum: 45812, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1692, AckNum: 1195, Checksum: 45300, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1693, AckNum: 1196, Checksum: 44788, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1694, AckNum: 1197, Checksum: 44276, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1695, AckNum: 1198, Checksum: 43764, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1696, AckNum: 1199, Checksum: 43252, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1697, AckNum: 1200, Checksum: 42740, Flags: ACK
2023-12-1 23:41:16 [SEND] Packet - SeqNum: 1200, AckNum: 0, Checksum: 19963, Flags: END
2023-12-1 23:41:16 [INFO] File: 3.jpg
2023-12-1 23:41:16 [INFO] Bytes Sent: 11968994 bytes
2023-12-1 23:41:16 [INFO] Time Taken: 62.422000 seconds
2023-12-1 23:41:16 [INFO] Average Speed: 191743.199513 bytes/s
2023-12-1 23:41:16 [SEND] Packet - SeqNum: 1201, AckNum: 0, Checksum: 12027, Flags: FIN
2023-12-1 23:41:16 [INFO] File is not sending, RecvThread close.
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1698, AckNum: 1202, Checksum: 41972, Flags: ACK
2023-12-1 23:41:16 [RECV] Packet - SeqNum: 1699, AckNum: 1202, Checksum: 33524, Flags: ACK FIN
2023-12-1 23:41:16 [SEND] Packet - SeqNum: 1202, AckNum: 1700, Checksum: 41460, Flags: ACK
2023-12-1 23:41:16 [INFO] Wavehand successful.

```

- helloworld.txt

```

2023-12-1 23:42:6 [SEND] Packet - SeqNum: 244, AckNum: 169, Checksum: 23294, Flags: ACK
2023-12-1 23:42:6 [RECV] Packet - SeqNum: 167, AckNum: 0, Checksum: 43519, Flags: DATA, Data Length: 10000
2023-12-1 23:42:6 [WARN] Received duplicate packet with seq: 167
2023-12-1 23:42:6 [SEND] Packet - SeqNum: 245, AckNum: 168, Checksum: 23294, Flags: ACK
2023-12-1 23:42:6 [RECV] Packet - SeqNum: 168, AckNum: 0, Checksum: 7293, Flags: DATA, Data Length: 5808
2023-12-1 23:42:6 [WARN] Received duplicate packet with seq: 168
2023-12-1 23:42:6 [SEND] Packet - SeqNum: 246, AckNum: 169, Checksum: 22782, Flags: ACK
2023-12-1 23:42:6 [RECV] Packet - SeqNum: 167, AckNum: 0, Checksum: 43519, Flags: DATA, Data Length: 10000
2023-12-1 23:42:6 [WARN] Received duplicate packet with seq: 167
2023-12-1 23:42:6 [SEND] Packet - SeqNum: 247, AckNum: 168, Checksum: 22782, Flags: ACK
2023-12-1 23:42:6 [RECV] Packet - SeqNum: 168, AckNum: 0, Checksum: 7293, Flags: DATA, Data Length: 5808
2023-12-1 23:42:6 [WARN] Received duplicate packet with seq: 168
2023-12-1 23:42:6 [SEND] Packet - SeqNum: 248, AckNum: 169, Checksum: 22270, Flags: ACK
2023-12-1 23:42:6 [RECV] Packet - SeqNum: 167, AckNum: 0, Checksum: 43519, Flags: DATA, Data Length: 10000
2023-12-1 23:42:6 [WARN] Received duplicate packet with seq: 167
2023-12-1 23:42:6 [SEND] Packet - SeqNum: 249, AckNum: 168, Checksum: 22270, Flags: ACK
2023-12-1 23:42:6 [RECV] Packet - SeqNum: 168, AckNum: 0, Checksum: 7293, Flags: DATA, Data Length: 5808
2023-12-1 23:42:6 [WARN] Received duplicate packet with seq: 168
2023-12-1 23:42:6 [SEND] Packet - SeqNum: 250, AckNum: 169, Checksum: 21758, Flags: ACK
2023-12-1 23:42:6 [RECV] Packet - SeqNum: 169, AckNum: 0, Checksum: 21759, Flags: END
2023-12-1 23:42:6 [INFO] File received and saved.
2023-12-1 23:42:6 [INFO] Bytes Recv: 1655808 bytes
2023-12-1 23:42:6 [INFO] Time Taken: 9.937000 seconds
2023-12-1 23:42:6 [INFO] Average Speed: 166630.572607 bytes/s
2023-12-1 23:42:6 [RECV] Packet - SeqNum: 170, AckNum: 0, Checksum: 13823, Flags: FIN
2023-12-1 23:42:6 [SEND] Packet - SeqNum: 251, AckNum: 171, Checksum: 20990, Flags: ACK
2023-12-1 23:42:6 [SEND] Packet - SeqNum: 252, AckNum: 171, Checksum: 12542, Flags: ACK FIN
2023-12-1 23:42:6 [RECV] Packet - SeqNum: 171, AckNum: 253, Checksum: 20478, Flags: ACK
2023-12-1 23:42:6 [INFO] Wavehand successful.

```

```

2023-12-1 23:42:6 [SEND] Packet - SeqNum: 168, AckNum: 0, Checksum: 7293, Flags: DATA, Data Length: 5808
2023-12-1 23:42:6 [INFO] SendBuffer: 167 168
2023-12-1 23:42:6 [WARN] Timeout, resend Packet in the Window! Current base: 167
2023-12-1 23:42:6 [INFO] SendBuffer: 167 168
2023-12-1 23:42:6 [SEND] Packet - SeqNum: 167, AckNum: 0, Checksum: 43519, Flags: DATA, Data Length: 10000
2023-12-1 23:42:6 [INFO] SendBuffer: 167 168
2023-12-1 23:42:6 [RECV] Packet - SeqNum: 241, AckNum: 168, Checksum: 24318, Flags: ACK
2023-12-1 23:42:6 [SEND] Packet - SeqNum: 168, AckNum: 0, Checksum: 7293, Flags: DATA, Data Length: 5808
2023-12-1 23:42:6 [INFO] SendBuffer: 168
2023-12-1 23:42:6 [RECV] Packet - SeqNum: 242, AckNum: 169, Checksum: 23806, Flags: ACK
2023-12-1 23:42:6 [SEND] Packet - SeqNum: 169, AckNum: 0, Checksum: 21759, Flags: END
2023-12-1 23:42:6 [INFO] File: helloworld.txt
2023-12-1 23:42:6 [INFO] Bytes Sent: 1655808 bytes
2023-12-1 23:42:6 [INFO] Time Taken: 9.937000 seconds
2023-12-1 23:42:6 [INFO] Average Speed: 166630.572607 bytes/s
2023-12-1 23:42:6 [SEND] Packet - SeqNum: 170, AckNum: 0, Checksum: 13823, Flags: FIN
2023-12-1 23:42:6 [RECV] Packet - SeqNum: 243, AckNum: 168, Checksum: 23806, Flags: ACK
2023-12-1 23:42:6 [RECV] Packet - SeqNum: 244, AckNum: 169, Checksum: 23294, Flags: ACK
2023-12-1 23:42:6 [INFO] File is not sending, RecvThread close.
2023-12-1 23:42:6 [RECV] Packet - SeqNum: 245, AckNum: 168, Checksum: 23294, Flags: ACK
2023-12-1 23:42:6 [RECV] Packet - SeqNum: 246, AckNum: 169, Checksum: 22782, Flags: ACK
2023-12-1 23:42:6 [RECV] Packet - SeqNum: 247, AckNum: 168, Checksum: 22782, Flags: ACK
2023-12-1 23:42:6 [RECV] Packet - SeqNum: 248, AckNum: 169, Checksum: 22270, Flags: ACK
2023-12-1 23:42:6 [RECV] Packet - SeqNum: 249, AckNum: 168, Checksum: 22270, Flags: ACK
2023-12-1 23:42:6 [RECV] Packet - SeqNum: 250, AckNum: 169, Checksum: 21758, Flags: ACK
2023-12-1 23:42:6 [RECV] Packet - SeqNum: 251, AckNum: 171, Checksum: 20990, Flags: ACK
2023-12-1 23:42:6 [RECV] Packet - SeqNum: 252, AckNum: 171, Checksum: 12542, Flags: ACK FIN
2023-12-1 23:42:6 [SEND] Packet - SeqNum: 171, AckNum: 253, Checksum: 20478, Flags: ACK
2023-12-1 23:42:6 [INFO] Wavehand successful.
Enter the path of the file to send:

```

7 总结与感想

在完成这个UDP文件传输项目的过程中，我深刻体会到了计算机网络原理在实际应用中的重要性。通过这次实验，我不仅加深了对UDP协议、滑动窗口、Go-Back-N重传机制等概念的理解，而且还学会了如何将这些理论知识应用到实际编程中。

实现过程中的挑战与收获：

1. **理论与实践的结合：**理解UDP协议的不可靠性和如何通过编程技巧实现可靠性，是我在这次实验中最大的挑战。通过实现滑动窗口和Go-Back-N机制，我更加深入地理解了这些概念的工作原理及其在实际通信中的重要性。

2. **编程能力的提升：**这次实验不仅要求理论知识的应用，还考验了我的编程能力。在实现过程中，我学习了更多关于多线程、互斥锁、队列等高级编程技巧。特别是在处理并发和数据一致性方面，我感到了明显的成长。