

南开大学

计算机网络课程实验报告

实验3-3



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

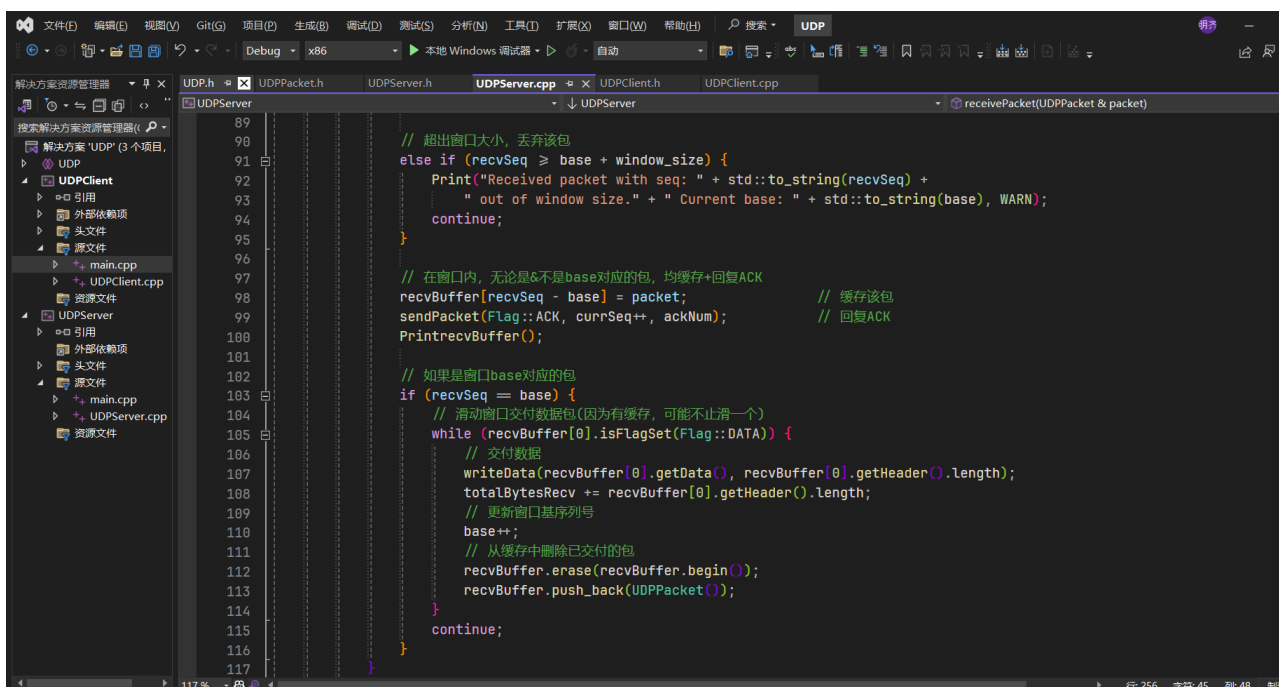
1 实验要求

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**选择确认**，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
- 选择确认：SR(Selective Repeat)
- 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



```
89
90
91 // 超出窗口大小，丢弃该包
92 else if (recvSeq >= base + window_size) {
93     Print("Received packet with seq: " + std::to_string(recvSeq) +
94         " out of window size." + " Current base: " + std::to_string(base), WARN);
95     continue;
96 }
97
98 // 在窗口内，无论是&不是base对应的包，均缓存+回复ACK
99 recvBuffer[recvSeq - base] = packet; // 缓存该包
100 sendPacket(Flag::ACK, currSeq++, ackNum); // 回复ACK
101 PrintrecvBuffer();
102
103 // 如果是窗口base对应的包
104 if (recvSeq == base) {
105     // 滑动窗口交付数据包(因为有缓存，可能不止滑一个)
106     while (recvBuffer[0].isFlagSet(Flag::DATA)) {
107         // 交付数据
108         writeData(recvBuffer[0].getData(), recvBuffer[0].getHeader().length);
109         totalBytesRecv += recvBuffer[0].getHeader().length;
110         // 更新窗口基序列号
111         base++;
112         // 从缓存中删除已交付的包
113         recvBuffer.erase(recvBuffer.begin());
114         recvBuffer.push_back(UDPpacket());
115     }
116     continue;
117 }
```

3 实验原理

3.1 滑动窗口机制

滑动窗口机制是一种流量控制策略，用于控制发送方在等待确认之前可以发送的数据量。它定义了发送方和接收方各自维护的一个窗口大小，这个窗口大小决定了可以发送或接收的数据包的数量。窗口随着数据包的确认逐渐“滑动”，即随着数据的确认，窗口会向前移动，允许发送或接收更多的数据包。这种机制确保了网络不会因为过多的未确认数据而过载，同时也允许在高延迟网络中实现更高的吞吐量。

3.2 Selective Repeat(SR) 选择确认协议

Selective Repeat (SR) 是一种自动重传请求 (ARQ) 协议，用于错误控制。在SR协议中，接收方独立地确认每个正确接收的数据包。如果某个数据包丢失或出错，只需重传那个特定的数据包，而不是重传所有后续的数据包。这与累计确认的协议相对，后者只确认到连续收到的最后一个数据包。SR协议通过允许发送方仅重传未被确认的数据包，提高了网络的效率。

3.3 握手挥手

在可靠的传输协议中，建立和终止连接通常涉及“握手”和“挥手”机制。三次握手（用于建立连接）包括发送方发送连接请求（SYN），接收方回应（SYN-ACK），然后发送方确认（ACK）。这确保双方都准备好进行数据交换。连接终止时使用的四次挥手包括发送方的终止请求（FIN），接收方的确认（ACK），接收方的终止请求（FIN），和发送方的最终确认（ACK）。这个过程确保了连接双方都同意关闭连接。

3.4 超时重传

超时重传是在数据传输过程中处理数据包丢失的机制。发送方在发送数据包后启动一个计时器。如果在设定的超时时间内未收到对应的确认（ACK），则假定数据包丢失，并重发该数据包。这种机制保证了即使在不可靠的网络条件下，数据也能最终被传输成功。

3.5 选择确认 vs 累计确认

选择确认（Selective Acknowledgment, SACK）和累计确认是两种不同的确认策略。在选择确认策略中，接收方可以独立确认每个接收到的数据包，而不是仅确认连续收到的最后一个数据包。这允许发送方只重传那些未被确认的数据包，而不是所有后续的数据包，从而提高了效率。相比之下，累计确认策略仅确认连续收到的最后一个数据包，使得发送方需要重传此后所有未被确认的数据包。虽然累计确认在简单性上有优势，但在高丢包环境下效率较低。

4 报文设计

我此次实验报文和3-2一样，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 };

```

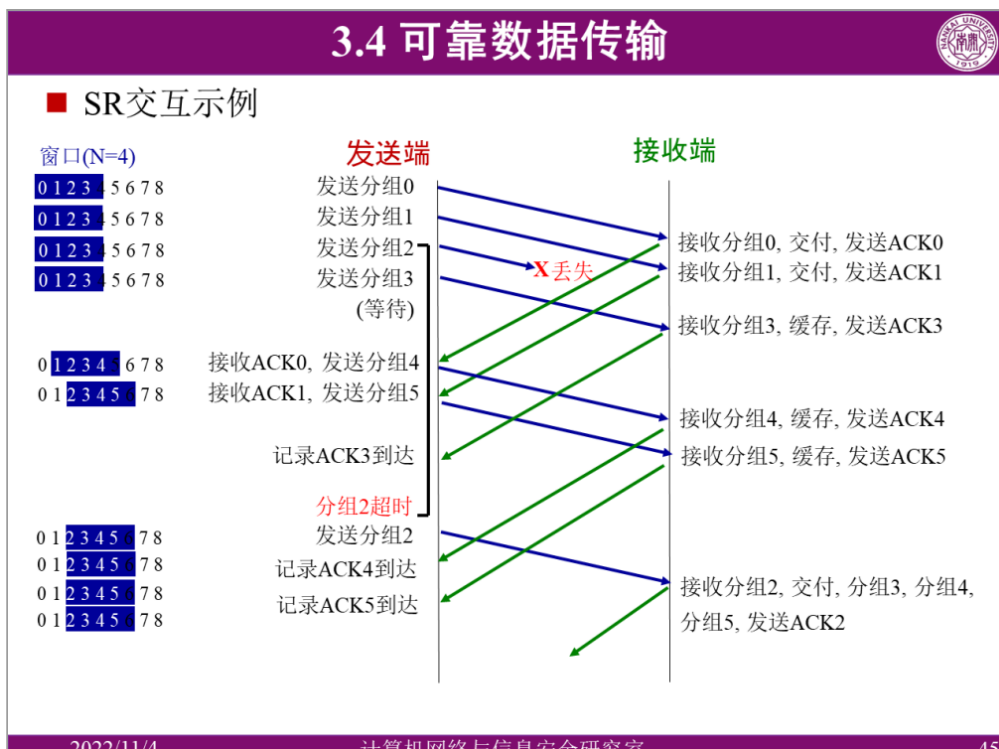
- **握手和挥手**使用FIN和SYN，ACK标志位，我使用了三次握手，四次挥手的机制，仿照TCP协议。
- **发送数据**：在握手成功后，发送端**首先发送一个置START位的包**，表明开始发送数据，然后开始发送数据，**发送数据使用DATA标志位，待数据都传输完毕后，发送一个置END位的包**，表明传输结束，然后开始挥手。

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**：

3.4 可靠数据传输



代码分为三部分：公共代码，客户端，服务端。

5.1 公共代码

公共代码部分基本同3-2，这里简单再次介绍一下。

服务端和客户端共用一个报文类，作为传输文件的协议。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 };

```

值得再次强调的是序列化和反序列化：

- **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。

另外为了提升传输显示，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      // 获取当前时间戳
9      std::string GetCurrTime() const {
10          .....
11      }
12
13      // 打印包信息
14      void PrintPacketInfo(const UDPPacket& packet, Level lv) const {
15          .....
16      }
17
18      // 打印信息
19      void Print(const std::string& info, Level lv = NOP) const {
20          .....
21      }
22 };

```

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  #pragma once
2
3  #include <fstream>
4  #include <random>
5  #include <utility>
6  #include <vector>
7  #include <cassert>
8  #include <mutex>
9  #include "UDP.h"
10
11 #define BUFFER_SIZE (DATA_SIZE + sizeof(Header))
12
13 struct BufferPacket {
14     UDPPacket packet;
15     bool ack;
16 };
17
18 class UDPClient : public UDP {
19 public:
20     UDPClient(const std::string& serverIP, UINT serverPort, uint32_t
window_size);
21     ~UDPClient();
22
23     void SendFile(const std::string& filePath);

```



```

24
25 private:
26     std::string serverIP;
27     UINT serverPort;
28     SOCKET clientSocket;
29     sockaddr_in serverAddr;
30     uint32_t window_size;
31     bool isconnected = false;
32     uint32_t nextseq = 0;
33     uint32_t base = 0;
34     uint32_t ackNum = 0;
35     uint64_t totalBytesSent = 0;           // 发送的总字节数
36     bool running = false;
37     ULONGLONG timer_start;
38     std::vector<BufferPacket> sendBuffer; // 存储已发送但尚未确认的数据包
39     std::mutex mtx;                       // 用于保护共享资源的互斥锁
40     static const ULONGLONG timeoutMs = 1000; // 超时时间
41
42     void handshake();
43     void waveHand();
44     void sendPacket(uint32_t flags, uint32_t seq, uint32_t ack = UINT_MAX,
45 const char* data = nullptr, uint16_t length = 0, bool resend = false);
46     void sendFileData(const std::string& filePath);
47     bool waitForPacket(uint32_t expectedFlag);
48     static DWORD WINAPI receiveAck(LPVOID pParam);
49     void Print(const std::string& info, Level lv = NOP);
50     void PrintPacketInfo(const UDPPacket& packet, Level lv);
51     void PrintsendBuffer();
52 };

```

首先定义了一些变量，如 `nextseq`，`base`，这些变量在SR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待发送的数据包。

• 发送缓冲区

在GBN的实验中，我采用了**队列**来作为缓冲区装载包：

```

1 std::queue<UDPPacket> sendBuffer;

```

但是，在这次SR的实验有所不同，我使用**向量**，并且使用了结构体：

```

1 struct BufferPacket {
2     UDPPacket packet;
3     bool ack;
4 };
5
6 std::vector<BufferPacket> sendBuffer;

```

为什么不使用队列? 因为在SR协议中，我们在发送端，不仅需要使用缓冲区来存储已发送的数据包，并且需要记录收到的ACK。言外之意，我们需要同时记录缓冲区的每个包是否收到了ACK，这是必须的，因为并不是累计确认而是选择确认，即便收到了ACK，如果出现了丢包情况，收到的ACK是后来的ACK，这时就不能滑动窗口，而是需要记录收到该ACK。因此我才定义一个结构体，使用bool变量同时记录了数据包是否收到了对应的ACK，同时使用向量来方便下标查找，而队列不方便进行下标查找。

我封装了一个通用的发送任何类型的包的函数：

```
1 // 发送Packet
2 void UDPPClient::sendPacket(uint32_t flags, uint32_t seq, uint32_t ack,
   const char* data, uint16_t length, bool resend)
3 {
4     UDPPacket packet;
5     packet.setSeq(seq); // 使用当前序列号
6     packet.setFlag(flags);
7     if (packet.isFlagSet(Flag::DATA)) {
8         packet.setData(data, length);
9         // 如果不是重发包则添加到发送缓冲区
10        if (!resend){
11            std::lock_guard<std::mutex> lock(mtx);
12            sendBuffer.push(packet);
13            totalBytesSent += length;
14        }
15    }
16    if (packet.isFlagSet(Flag::ACK)) {
17        packet.setAck(ack);
18    }
19    packet.setChecksum(packet.calChecksum()); // 计算校验和
20
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。

在函数中，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。这也是写通用函数代码需要注意的地方，需要考虑各个调用点的情况。

- **文件发送 & 超时重传**

使用如下函数发送文件数据，同时处理超时重传的逻辑：

```
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();
8      // 记录开始时间(区别于timer_start)
9
10     std::ifstream file(filePath, std::ios::binary | std::ios::ate);
11     file.seekg(0, std::ios::beg);
12     char* const buffer = new char[DATA_SIZE];
13     bool eof = false;
14     running = true;
15     base = nextseq; //让base指向第一个发送的数据包
16     HANDLE hrecv = CreateThread(NULL, 0, receiveAck, this, 0, NULL);
17     // 创建接收ACK线程
18     while (true) {
19         // 读取文件并发送包
20         while (nextseq < base + window_size && !eof) {
21             file.read(buffer, DATA_SIZE);
22             std::streamsize bytesRead = file.gcount();
23             eof = (bytesRead < DATA_SIZE);
24
25             if (base == nextseq) {
26                 // 如果这是窗口中的第一个包，则重置定时器
27                 std::lock_guard<std::mutex> lock(mtx);
28                 timer_start = GetTickCount64();
29             }
30             sendPacket(Flag::DATA, nextseq++, 0, buffer,
static_cast<uint16_t>(bytesRead));
31             if (eof) {
32                 Print("EOF reached.", INFO);
33             }
34         }
35     }
```

```

36         // 当发送缓冲区为空时结束
37         if (sendBuffer.empty()) break;
38
39         // 超时重传逻辑
40         if (GetTickCount64() - timer_start > timeoutMs &&
!sendBuffer.empty()) {
41             // 打印出base和sendbuffer内容
42             Print("Timeout, resend the First Packet in the Window! Current
base: " + std::to_string(base), WARN);
43             // 从缓冲区中获取当前base的包并重传(选择重传)
44             UDPPacket packet = sendBuffer[0].packet;
45             Header pHead = packet.getHeader();
46             sendPacket(pHead.flags, pHead.seqNum, 0, packet.getData(),
pHead.length, true);
47             timer_start = GetTickCount64(); // 重置定时器
48         }
49     }
50
51     // 关闭接收线程
52     {
53         std::lock_guard<std::mutex> lock(mtx);
54         running = false;
55     }
56     if (hrecv) CloseHandle(hrecv);
57
58     // 发送 END 包
59     sendPacket(Flag::END, nextseq++);
60     file.close();
61     delete[] buffer;
62
63     // 打印发送的总字节数和时间
64     ULONGLONG endTime = GetTickCount64();
65     double elapsed = static_cast<double>(endTime - startTime) / 1000.0;
66     Print("File: " + filePath, INFO);
67     Print("Bytes Sent: " + std::to_string(totalBytesSent) + " bytes",
INFO);
68     Print("Time Taken: " + std::to_string(elapsed) + " seconds", INFO);
69     Print("Average Speed: " + std::to_string(totalBytesSent / elapsed) + "
bytes/s", INFO);
70 }

```

根据我的协议，发送数据的大致三步流程：

1. 握手，然后发送START标志包
2. 发送DATA标志包
3. 发送END包，然后挥手

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，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，我们只会重传超时的包。

• 一个思考

根据我的理解，我在超时重传的逻辑上做了一个简化。具体来说，我进行了一个思考：如果同时有多个数据包需要超时重传，比如窗口是4，发送端发送了1 2 3 4，2和4均需要重传，那么1确实是收到ACK了，滑动窗口，这时候缓冲区第一个包是2，2重传后会收到ACK，然后继续滑动窗口，滑到4之后同理继续重传，这样是不是不需要为每个包单独开计时器了？于是可以看到我的代码里，重传只使用缓冲区的第一个包：

```
1 // 从缓冲区中获取当前base的包并重传(选择重传)
2 UDPPacket packet = sendBuffer[0].packet;
```

因为正如我上述描述的，任何需要重传的包，无论是一个还是若干个，最终都会滑到缓冲区的第一个位置，因此我只需要“瞄准”这个位置，处理这个位置的超时重传，即可简化逻辑。

以上是我对SR超时重传的思考，但是否存在问题呢？答案是肯定的。助教学长指出了我的问题：这样做逻辑上确实是SR的逻辑，即我们只重传需要重传的包，而不是整个窗口，但是因为我只处理缓冲区开头的包，势必造成时间的浪费，即发送端发送了1 2 3 4，2和4均需要重传，如果开一个线程来单独处理每个包的重传，那么完全不需要等到处理完2的重传后，再等滑动窗口，再处理4的重传。

因此可以总结出，我这样处理有利有弊，即简化了代码逻辑，减少了线程的使用，减小代码开销，但增大了等待时间，如果在丢包率特别大的环境下，我的弊端就得以显现，幸运的是一般丢包率不会很大。

• 专门用来接收ACK的线程

这是我用来接收ACK的线程函数：

```
1 // 接收ACK线程
2 DWORD WINAPI UDPCliet::receiveAck(LPVOID pParam) {
3     UDPCliet* client = (UDPCliet*)pParam;
4     char* const buffer = new char[BUFFER_SIZE];
5     int addrLen = sizeof(client->serverAddr);
6 }
```

```

7     while (client->running) {
8         int recvLen = recvfrom(client->clientSocket, buffer, BUFFER_SIZE,
14 0, (struct sockaddr*)&client->serverAddr, &addrLen);
9         if (recvLen > 0) {
10             UDPPacket packet;
11             packet.deserialize(std::string(buffer, recvLen));
12             client->PrintPacketInfo(packet, RECV);
13
14             // 检查校验和
15             if (!packet.validChecksum()) {
16                 client->Print("Checksum failed for packet with seq: " +
17 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                 // 如果是重复的ACK, 不做处理
26                 if (ack <= client->base) {
27                     client->Print("Received duplicate ACK with ack: " +
28 std::to_string(ack)
29 + " Current base: " + std::to_string(client-
30 >base), WARN);
31                     continue;
32                 }
33
34                 {
35                     std::lock_guard<std::mutex> lock(client->mtx);
36                     // 锁定互斥锁
37                     client->ackNum = packet.getHeader().seqNum + 1;
38                     // 更新确认号
39                     client->timer_start = GetTickCount64();
40                     // 重置定时器
41
42                     // 选择确认(SR)机制
43                     // 记录对应的包收到ACK
44                     client->sendBuffer[ack - client->base - 1].ack = true;
45                 }
46
47                 // 如果是base对应的ACK, 滑动窗口
48                 if (ack == client->base + 1) {
49                     std::lock_guard<std::mutex> lock(client->mtx);
50                     // 因为记录了ACK, 所以可能不止滑动一个

```

```

48         while (!client->sendBuffer.empty() && client-
>sendBuffer[0].ack) {
49             client->sendBuffer.erase(client-
>sendBuffer.begin()); // 从缓冲区中删除已确认的包
50             client->base++; // 更新窗口基序号
51         }
52     }
53     else client->PrintsendBuffer();
54 }
55 }
56 Sleep(10);
57 }
58 client->Print("File is not sending, RecvThread close.", INFO);
59 delete[] buffer;
60 return 0;
61 }

```

- **线程的创建与运行**：我在 `sendFileData` 函数中创建了这个线程，使用了 `CreateThread` 函数。线程启动后，它会不断地监听来自服务端的数据包，直到 `running` 变量被设置为 `false`，这个标志在文件传输完成或中断时被设置。
- **接收并处理ACK**：线程中，我使用 `recvfrom` 函数接收数据包。每当接收到一个数据包，我首先验证其校验和，以确保数据的完整性。对于有效的ACK包，我采取进一步的操作。
- **重复ACK的处理**：如果接收到的ACK是重复的（即 `ack <= client->base`），我不做任何处理。这种情况通常发生在ACK包丢失或网络延迟的情况下。
- **更新确认状态**：对于新的ACK，我更新发送缓冲区中对应数据包的确认状态。在SR协议中，每个数据包独立被确认，所以我在缓冲区中对每个数据包维护一个 `ack` 标志。
- **滑动窗口的实现**：与Go-Back-N (GBN)协议的累计确认机制不同，SR协议允许单独确认每个数据包，这意味着在我的发送缓冲区中，可能会出现一种情况：连续多个数据包都已经收到了确认（ACK），窗口中的第一个包才收到确认。当基序号（`base`）对应的数据包收到ACK时，滑动窗口，从发送缓冲区中移除已确认的数据包，并更新 `base` 值。通过这种方式，窗口向前移动，为新的数据包的发送腾出空间。如果连续的几个包都已经被确认，我就一次性将它们从缓冲区中移除，并相应地更新 `base` 值，这样窗口就一次性向前滑动了多个位置。
- **线程的终止**：当文件发送完毕后，我在 `sendFileData` 函数中修改 `running` 变量为 `false`，这会导致线程结束运行。线程完成其任务后，我还会输出一条日志消息，表示接收线程已关闭。

5.3 接收端

接收端本次和GBN的大不相同，因为GBN可以理解为接受端的窗口大小为1，但SR要求接收窗口大于1。

我定义了UDPServer类：

```

1 class UDPServer : public UDP {
2 public:

```



```

3     UDPServer(UINT port, uint32_t window_size);
4     ~UDPServer();
5
6     void Start();
7     void Stop();
8
9 private:
10    UINT port;
11    SOCKET serverSocket;
12    sockaddr_in serverAddr;
13    sockaddr_in clientAddr;
14    std::ofstream outFile;
15    bool isconnect = false;
16    uint32_t window_size;
17    uint32_t base = 0;           // 接收窗口基序号
18    uint32_t ackNum = 0;        // 确认号
19    uint32_t currSeq = 0;
20    uint64_t totalBytesRecv = 0; // 接收的总字节数
21    std::vector<UDPPacket> recvBuffer; // 存储已接收但尚未交付的数据包
22
23    void receiveData();
24    void writeData(const char* data, uint16_t length);
25    void sendPacket(uint32_t flags, uint32_t seq, uint32_t ack = UINT_MAX,
const char* data = nullptr, uint16_t length = 0);
26    void openFile(const std::string& filename);
27    void closeFile();
28    void handshake();
29    void waveHand();
30    bool waitForPacket(uint32_t expectedFlag);
31    bool receivePacket(UDPPacket& packet);
32    void PrintrecvBuffer();
33 };

```

- 接收窗口与缓冲区

```

1 std::vector<UDPPacket> recvBuffer;

```

我在 `UDPServer` 类中增加了一个接收缓冲区 `recvBuffer`，这个缓冲区的大小等于窗口大小 `window_size`。这个缓冲区用于存储已经接收但尚未按顺序处理的数据包。在SR协议中，数据包可以被乱序接收和确认，因此需要一个能够临时存储这些乱序数据包的结构，直到它们可以被按顺序处理。

- 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 = 50; // 延时50ms

```



```

4     const int drop_every = 33; // 丢包率为3%
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             drop = (lossDistribution(generator_drop) == 0);
28             delay = (delayDistribution(generator_delay) == 0);
29             // 随机选择Data包进行丢包
30             if (drop) {
31                 Print("Simulating packet loss for packet seq: " +
std::to_string(pktHeader.seqNum), WARN);
32                 return false; // 不处理该包, 模拟丢包
33             }
34             // 随机选择Data包进行延时
35             if (delay) {
36                 Print("Delaying ACK for packet seq: " +
std::to_string(pktHeader.seqNum), WARN);
37                 Sleep(fixedDelay);
38             }
39         }
40         ackNum = pktHeader.seqNum + 1; // 更新ACK
41         // 打印接收到的数据包信息
42         PrintPacketInfo(packet, RECV);
43         return true;

```

```

44     }
45     else if (recvLen == 0 || WSAGetLastError() != WSAEWOULDBLOCK) {
46         Print("recvfrom() failed with error code: " +
std::to_string(WSAGetLastError()), ERR);
47     }
48     delete[] buffer;
49     return false;
50 }

```

1. 数据包校验

为了确保数据完整性，我对每个接收到的数据包执行了校验和计算。如果校验和不匹配，这意味着数据包在传输过程中被破坏，我将打印一条警告信息，并丢弃该数据包。

2. 模拟丢包

为了模拟真实网络环境中的丢包情况，我引入了一个概率机制。通过随机数生成器 `generator_drop` 和分布 `lossDistribution`，我设定了一个固定的丢包概率（这里是3%）。如果随机数生成器产生的数字符合丢包条件，函数将打印一条模拟丢包的警告信息并返回，模拟该数据包在传输过程中丢失的情况。

3. 模拟延时

与丢包类似，我也为数据包引入了延时机制。使用另一个随机数生成器 `generator_delay` 和分布 `delayDistribution`，我随机决定某些数据包是否应该被延时处理。一旦一个数据包被选中进行延时，函数将使用 `Sleep` 函数暂停一段时间（这里设置为50毫秒），模拟网络延迟。

• 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;
17             }
18             // 重置接收的总字节数
19         }
20     }
21 }

```

```

17         base = pktHeader.seqNum + 1;           // 重置窗口基序列号
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seq范围
26         uint32_t recvSeq = pktHeader.seqNum;
27         // 重复接收到的包
28         if (recvSeq < base) {
29             Print("Received duplicate packet with seq: " +
30                 std::to_string(recvSeq) + " Current base: " +
std::to_string(base), WARN);
31             sendPacket(Flag::ACK, currSeq++, recvSeq + 1); // 回
复ACK
32             continue;
33         }
34
35         // 超出窗口大小, 丢弃该包
36         else if (recvSeq >= base + window_size) {
37             Print("Received packet with seq: " +
std::to_string(recvSeq) +
38                 " out of window size." + " Current base: " +
std::to_string(base), WARN);
39             continue;
40         }
41
42         // 在窗口内, 无论是&不是base对应的包, 均缓存+回复ACK
43         recvBuffer[recvSeq - base] = packet; // 缓
存该包
44         sendPacket(Flag::ACK, currSeq++, ackNum); // 回
复ACK
45         PrintrecvBuffer();
46
47         // 如果是窗口base对应的包
48         if (recvSeq == base) {
49             // 滑动窗口交付数据包(因为有缓存, 可能不止滑一个)
50             while (recvBuffer[0].isFlagSet(Flag::DATA)) {
51                 // 交付数据
52                 writeData(recvBuffer[0].getData(),
recvBuffer[0].getHeader().length);
53                 totalBytesRecv +=
recvBuffer[0].getHeader().length;
54                 // 更新窗口基序列号

```

```

55         base++;
56         // 从缓存中删除已交付的包
57         recvBuffer.erase(recvBuffer.begin());
58         recvBuffer.push_back(UDPPacket());
59     }
60     continue;
61 }
62 }
63
64 // 检查是否是文件传输的结束
65 if (packet.isFlagSet(Flag::END)) {
66     endTime = GetTickCount64(); // 记录结束时间
67     closeFile();
68     receivingFile = false;
69     double elapsed = static_cast<double>(endTime - startTime)
/ 1000.0;
70     Print("Bytes Recv: " + std::to_string(totalBytesRecv) + "
bytes", INFO);
71     Print("Time Taken: " + std::to_string(elapsed) + "
seconds", INFO);
72     Print("Average Speed: " + std::to_string(totalBytesRecv /
elapsed) + " bytes/s", INFO);
73     continue;
74 }
75
76 // 处理握手请求
77 if (packet.isFlagSet(Flag::SYN)) {
78     handshake();
79     continue;
80 }
81
82 // 检查是否是挥手请求 (FIN)
83 if (packet.isFlagSet(Flag::FIN)) {
84     waveHand();
85     continue;
86 }
87 }
88 }
89 }

```

在我的 `UDPServer` 类中，`receiveData` 函数是接收端逻辑的核心。这个函数负责监听和处理来自客户端的所有数据包。让我详细地解释一下这个函数的主要逻辑：

1. 文件传输的开始与结束

- 当接收到带有 `Flag::START` 标志的包时，意味着文件传输即将开始。我首先通过 `openFile` 函数打开文件准备写入数据，并重置了接收到的总字节数 `totalBytesRecv`，同时记录文件传输的开始时间 `startTime`。
- 类似地，当接收到带有 `Flag::END` 标志的包时，意味着文件传输已经结束。我关闭文件，并计算文件传输的总时间和平均速率，记录这些信息以便后续的分析。

2. 处理数据包

- 对于每个接收到的数据包，首先检查其序列号 `seqNum`。如果序列号小于当前窗口的基序号 `base`，说明这是一个已经接收并确认过的重复包。在这种情况下，我仅发送相应的ACK，但不再处理数据。
- 如果序列号超出了接收窗口的范围（即 `seqNum >= base + window_size`），这表明包的序列号太高，无法处理，因此我选择丢弃该包，并打印一条警告信息。
- 最重要的部分是当包的序列号在当前窗口内时（即 `base <= seqNum < base + window_size`）。这时，我将包存储在接收缓冲区 `recvBuffer` 中，并回复ACK。不论包的序列号是否与窗口基序号相符，我都执行此操作。

3. 滑动窗口与数据交付

- 在滑动窗口机制中，当收到基序号对应的包时，我开始检查接收缓冲区。如果基序号对应的包存在，我就交付该数据包的内容到文件中，并从缓冲区中删除它。此后，我更新基序号 `base`，使其指向下一个期望接收的序列号。
- 重要的是，我不仅检查和处理基序号对应的包，还会检查接收缓冲区中是否还有连续的后续包也已经到达，这也是接收缓冲区的作用。如果是，我也将它们交付到文件中，相应地更新 `base`。这种方法是选择重传（SR）协议的特点，允许接收端接收并缓存非顺序到达的包，并在适当的时候一次性处理它们。

4. 握手与挥手

- 此外，我还处理了握手和挥手请求。当收到握手（SYN）或挥手（FIN）请求时，我调用相应的 `handshake` 或 `waveHand` 函数来处理这些控制消息。

6 运行结果

同3-2，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	25

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

```

Enter the Window Size: 25
2023-12-15 23:22:32 [INFO] Server start at port: 12720
2023-12-15 23:22:47 [RECV] Packet - SeqNum: 0, AckNum: 0, Checksum: 61439, Flags: SYN
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 0, AckNum: 1, Checksum: 59135, Flags: ACK SYN
2023-12-15 23:22:47 [RECV] Packet - SeqNum: 1, AckNum: 1, Checksum: 62975, Flags: ACK
2023-12-15 23:22:47 [INFO] Handshake successful.
2023-12-15 23:22:47 [RECV] Packet - SeqNum: 2, AckNum: 0, Checksum: 64767, Flags: START
2023-12-15 23:22:47 [INFO] Start receiving file.
2023-12-15 23:22:47 [RECV] Packet - SeqNum: 3, AckNum: 0, Checksum: 7657, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 1, AckNum: 4, Checksum: 62207, Flags: ACK
2023-12-15 23:22:47 [INFO] RecvBuffer: 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2023-12-15 23:22:47 [RECV] Packet - SeqNum: 4, AckNum: 0, Checksum: 42644, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 2, AckNum: 5, Checksum: 61695, Flags: ACK
2023-12-15 23:22:47 [INFO] RecvBuffer: 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2023-12-15 23:22:47 [RECV] Packet - SeqNum: 5, AckNum: 0, Checksum: 36778, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 3, AckNum: 6, Checksum: 61183, Flags: ACK
2023-12-15 23:22:47 [INFO] RecvBuffer: 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2023-12-15 23:22:47 [RECV] Packet - SeqNum: 6, AckNum: 0, Checksum: 11908, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 4, AckNum: 7, Checksum: 60671, Flags: ACK
2023-12-15 23:22:47 [INFO] RecvBuffer: 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2023-12-15 23:22:47 [RECV] Packet - SeqNum: 7, AckNum: 0, Checksum: 30777, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 5, AckNum: 8, Checksum: 60159, Flags: ACK
2023-12-15 23:22:47 [INFO] RecvBuffer: 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2023-12-15 23:22:47 [RECV] Packet - SeqNum: 8, AckNum: 0, Checksum: 16067, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 6, AckNum: 9, Checksum: 59647, Flags: ACK
2023-12-15 23:22:47 [INFO] RecvBuffer: 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2023-12-15 23:22:47 [RECV] Packet - SeqNum: 9, AckNum: 0, Checksum: 13421, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 7, AckNum: 10, Checksum: 59135, Flags: ACK
2023-12-15 23:22:47 [INFO] RecvBuffer: 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2023-12-15 23:22:47 [RECV] Packet - SeqNum: 10, AckNum: 0, Checksum: 17717, Flags: DATA, Data Length: 10000

```

```

The server port: 12720
Enter the Window Size: 25
Enter the path of the file to send: 1.jpg
2023-12-15 23:22:47 [INFO] File opened successfully: 1.jpg
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 0, AckNum: 0, Checksum: 61439, Flags: SYN
2023-12-15 23:22:47 [RECV] Packet - SeqNum: 0, AckNum: 1, Checksum: 59135, Flags: ACK SYN
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 1, AckNum: 1, Checksum: 62975, Flags: ACK
2023-12-15 23:22:47 [INFO] Handshake successful.
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 2, AckNum: 0, Checksum: 64767, Flags: START
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 3, AckNum: 0, Checksum: 7657, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [INFO] SendBuffer: 3
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 4, AckNum: 0, Checksum: 42644, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [INFO] SendBuffer: 3 4
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 5, AckNum: 0, Checksum: 36778, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [INFO] SendBuffer: 3 4 5
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 6, AckNum: 0, Checksum: 11908, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [INFO] SendBuffer: 3 4 5 6
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 7, AckNum: 0, Checksum: 30777, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [INFO] SendBuffer: 3 4 5 6 7
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 8, AckNum: 0, Checksum: 16067, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [INFO] SendBuffer: 3 4 5 6 7 8
2023-12-15 23:22:47 [RECV] Packet - SeqNum: 1, AckNum: 4, Checksum: 62207, Flags: ACK
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 9, AckNum: 0, Checksum: 13421, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [INFO] SendBuffer: 4 5 6 7 8 9
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 10, AckNum: 0, Checksum: 17717, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [INFO] SendBuffer: 4 5 6 7 8 9 10
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 11, AckNum: 0, Checksum: 47122, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [INFO] SendBuffer: 4 5 6 7 8 9 10 11
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 12, AckNum: 0, Checksum: 30594, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [INFO] SendBuffer: 4 5 6 7 8 9 10 11 12

```

可以看到，在未触发丢包机制时，接收缓冲区始终只有一个包，并且下一次这个包也被滑走，因为包一直是按序到达的，接下来是触发丢包的效果：


```

D:\study\大三\计算机网络\实验\3-3\2113997_齐明杰_编程作业3-3\UDPServer\UDPServer.exe
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 30, AckNum: 34, Checksum: 47103, Flags: ACK
2023-12-15 23:22:47 [INFO] RecvBuffer: 0 31 32 33 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2023-12-15 23:22:47 [RECV] Packet - SeqNum: 34, AckNum: 0, Checksum: 35913, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 31, AckNum: 35, Checksum: 46591, Flags: ACK
2023-12-15 23:22:47 [INFO] RecvBuffer: 0 31 32 33 34 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2023-12-15 23:22:47 [WARN] Simulating packet loss for packet seq: 35
2023-12-15 23:22:47 [RECV] Packet - SeqNum: 36, AckNum: 0, Checksum: 30634, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 32, AckNum: 37, Checksum: 45823, Flags: ACK
2023-12-15 23:22:47 [INFO] RecvBuffer: 0 31 32 33 34 0 36 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2023-12-15 23:22:47 [RECV] Packet - SeqNum: 37, AckNum: 0, Checksum: 65172, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 33, AckNum: 38, Checksum: 45311, Flags: ACK
2023-12-15 23:22:47 [INFO] RecvBuffer: 0 31 32 33 34 0 36 37 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2023-12-15 23:22:47 [RECV] Packet - SeqNum: 38, AckNum: 0, Checksum: 44656, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 34, AckNum: 39, Checksum: 44799, Flags: ACK
2023-12-15 23:22:47 [INFO] RecvBuffer: 0 31 32 33 34 0 36 37 38 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2023-12-15 23:22:47 [RECV] Packet - SeqNum: 39, AckNum: 0, Checksum: 50168, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 35, AckNum: 40, Checksum: 44287, Flags: ACK
2023-12-15 23:22:47 [INFO] RecvBuffer: 0 31 32 33 34 0 36 37 38 39 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2023-12-15 23:22:47 [RECV] Packet - SeqNum: 40, AckNum: 0, Checksum: 41841, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 36, AckNum: 41, Checksum: 43775, Flags: ACK
2023-12-15 23:22:47 [INFO] RecvBuffer: 0 31 32 33 34 0 36 37 38 39 40 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2023-12-15 23:22:47 [RECV] Packet - SeqNum: 41, AckNum: 0, Checksum: 51346, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 37, AckNum: 42, Checksum: 43263, Flags: ACK
2023-12-15 23:22:47 [INFO] RecvBuffer: 0 31 32 33 34 0 36 37 38 39 40 41 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2023-12-15 23:22:47 [RECV] Packet - SeqNum: 42, AckNum: 0, Checksum: 47900, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 38, AckNum: 43, Checksum: 42751, Flags: ACK
2023-12-15 23:22:47 [INFO] RecvBuffer: 0 31 32 33 34 0 36 37 38 39 40 41 42 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2023-12-15 23:22:47 [RECV] Packet - SeqNum: 43, AckNum: 0, Checksum: 21420, Flags: DATA, Data Length: 10000
2023-12-15 23:22:47 [SEND] Packet - SeqNum: 39, AckNum: 44, Checksum: 42239, Flags: ACK
2023-12-15 23:22:47 [INFO] RecvBuffer: 0 31 32 33 34 0 36 37 38 39 40 41 42 43 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

```

D:\study\大三\计算机网络\实验\3-3\2113997_齐明杰_编程作业3-3\UDPClient\UDPClient.exe
A) 45(A) 46(A) 47(A) 48(A) 49(A) 50(A) 51 52 53 54
2023-12-15 23:22:48 [RECV] Packet - SeqNum: 47, AckNum: 52, Checksum: 38143, Flags: ACK
2023-12-15 23:22:48 [INFO] SendBuffer: 30 31(A) 32(A) 33(A) 34(A) 35 36(A) 37(A) 38(A) 39(A) 40(A) 41(A) 42(A) 43(A) 44(A)
A) 45(A) 46(A) 47(A) 48(A) 49(A) 50(A) 51(A) 52 53 54
2023-12-15 23:22:48 [RECV] Packet - SeqNum: 48, AckNum: 53, Checksum: 37631, Flags: ACK
2023-12-15 23:22:48 [INFO] SendBuffer: 30 31(A) 32(A) 33(A) 34(A) 35 36(A) 37(A) 38(A) 39(A) 40(A) 41(A) 42(A) 43(A) 44(A)
A) 45(A) 46(A) 47(A) 48(A) 49(A) 50(A) 51(A) 52(A) 53 54
2023-12-15 23:22:48 [RECV] Packet - SeqNum: 49, AckNum: 54, Checksum: 37119, Flags: ACK
2023-12-15 23:22:48 [INFO] SendBuffer: 30 31(A) 32(A) 33(A) 34(A) 35 36(A) 37(A) 38(A) 39(A) 40(A) 41(A) 42(A) 43(A) 44(A)
A) 45(A) 46(A) 47(A) 48(A) 49(A) 50(A) 51(A) 52(A) 53(A) 54
2023-12-15 23:22:48 [RECV] Packet - SeqNum: 50, AckNum: 55, Checksum: 36607, Flags: ACK
2023-12-15 23:22:48 [INFO] SendBuffer: 30 31(A) 32(A) 33(A) 34(A) 35 36(A) 37(A) 38(A) 39(A) 40(A) 41(A) 42(A) 43(A) 44(A)
A) 45(A) 46(A) 47(A) 48(A) 49(A) 50(A) 51(A) 52(A) 53(A) 54(A)
2023-12-15 23:22:49 [WARN] Timeout, resend the First Packet in the Window! Current base: 30
2023-12-15 23:22:49 [SEND] Packet - SeqNum: 30, AckNum: 0, Checksum: 46250, Flags: DATA, Data Length: 10000
2023-12-15 23:22:49 [INFO] SendBuffer: 30 31(A) 32(A) 33(A) 34(A) 35 36(A) 37(A) 38(A) 39(A) 40(A) 41(A) 42(A) 43(A) 44(A)
A) 45(A) 46(A) 47(A) 48(A) 49(A) 50(A) 51(A) 52(A) 53(A) 54(A)
2023-12-15 23:22:49 [RECV] Packet - SeqNum: 51, AckNum: 31, Checksum: 42495, Flags: ACK
2023-12-15 23:22:49 [SEND] Packet - SeqNum: 55, AckNum: 0, Checksum: 63818, Flags: DATA, Data Length: 10000
2023-12-15 23:22:49 [INFO] SendBuffer: 35 36(A) 37(A) 38(A) 39(A) 40(A) 41(A) 42(A) 43(A) 44(A) 45(A) 46(A) 47(A) 48(A)
49(A) 50(A) 51(A) 52(A) 53(A) 54(A) 55
2023-12-15 23:22:49 [SEND] Packet - SeqNum: 56, AckNum: 0, Checksum: 38262, Flags: DATA, Data Length: 10000
2023-12-15 23:22:49 [INFO] SendBuffer: 35 36(A) 37(A) 38(A) 39(A) 40(A) 41(A) 42(A) 43(A) 44(A) 45(A) 46(A) 47(A) 48(A)
49(A) 50(A) 51(A) 52(A) 53(A) 54(A) 55 56
2023-12-15 23:22:49 [SEND] Packet - SeqNum: 57, AckNum: 0, Checksum: 12529, Flags: DATA, Data Length: 10000
2023-12-15 23:22:49 [INFO] SendBuffer: 35 36(A) 37(A) 38(A) 39(A) 40(A) 41(A) 42(A) 43(A) 44(A) 45(A) 46(A) 47(A) 48(A)
49(A) 50(A) 51(A) 52(A) 53(A) 54(A) 55 56 57
2023-12-15 23:22:49 [SEND] Packet - SeqNum: 58, AckNum: 0, Checksum: 17200, Flags: DATA, Data Length: 10000
2023-12-15 23:22:49 [INFO] SendBuffer: 35 36(A) 37(A) 38(A) 39(A) 40(A) 41(A) 42(A) 43(A) 44(A) 45(A) 46(A) 47(A) 48(A)
49(A) 50(A) 51(A) 52(A) 53(A) 54(A) 55 56 57 58

```

由于丢包，序列号为30的包无法收到ACK，触发了超时重传，同时这时候发送端对31，32.....等ACK进行记录，如图中的(A)标记，可以看到收到30包的ACK后，发送缓存直接滑到了35的包，因为35的包也被模拟丢包了，之后同理。

而对于接收端，由于丢包的存在，接收缓存保存了未按需到达的数据包，在之后30这个包到达后将一次性滑动窗口进行交付数据。

最终结果：

- 2.jpg


```

D:\study\大三\计算机网络_实验\3-3\2113997_齐明杰_编程作业3-3\UDPServer\UDPServer.exe
2023-12-15 23:33:54 [SEND] Packet - SeqNum: 1193, AckNum: 1198, Checksum: 41206, Flags: ACK
2023-12-15 23:33:54 [INFO] RecvBuffer: 0 1176 0 1178 1179 1180 1181 1182 1183 1184 1185 1186 1187 1188 1189 1190 1191 11
92 1193 1194 1195 1196 1197 0 0
2023-12-15 23:33:54 [RECV] Packet - SeqNum: 1198, AckNum: 0, Checksum: 50986, Flags: DATA, Data Length: 10000
2023-12-15 23:33:54 [SEND] Packet - SeqNum: 1194, AckNum: 1199, Checksum: 40694, Flags: ACK
2023-12-15 23:33:54 [INFO] RecvBuffer: 0 1176 0 1178 1179 1180 1181 1182 1183 1184 1185 1186 1187 1188 1189 1190 1191 11
92 1193 1194 1195 1196 1197 1198 0
2023-12-15 23:33:54 [RECV] Packet - SeqNum: 1199, AckNum: 0, Checksum: 49538, Flags: DATA, Data Length: 8994
2023-12-15 23:33:54 [SEND] Packet - SeqNum: 1195, AckNum: 1200, Checksum: 40182, Flags: ACK
2023-12-15 23:33:54 [INFO] RecvBuffer: 0 1176 0 1178 1179 1180 1181 1182 1183 1184 1185 1186 1187 1188 1189 1190 1191 11
92 1193 1194 1195 1196 1197 1198 1199
2023-12-15 23:33:55 [RECV] Packet - SeqNum: 1175, AckNum: 0, Checksum: 11081, Flags: DATA, Data Length: 10000
2023-12-15 23:33:55 [SEND] Packet - SeqNum: 1196, AckNum: 1176, Checksum: 46070, Flags: ACK
2023-12-15 23:33:55 [INFO] RecvBuffer: 1175 1176 0 1178 1179 1180 1181 1182 1183 1184 1185 1186 1187 1188 1189 1190 1191
1192 1193 1194 1195 1196 1197 1198 1199
2023-12-15 23:33:56 [RECV] Packet - SeqNum: 1177, AckNum: 0, Checksum: 14929, Flags: DATA, Data Length: 10000
2023-12-15 23:33:56 [SEND] Packet - SeqNum: 1197, AckNum: 1178, Checksum: 45302, Flags: ACK
2023-12-15 23:33:56 [INFO] RecvBuffer: 1177 1178 1179 1180 1181 1182 1183 1184 1185 1186 1187 1188 1189 1190 1191 1192 1
193 1194 1195 1196 1197 1198 1199 0 0
2023-12-15 23:33:56 [RECV] Packet - SeqNum: 1200, AckNum: 0, Checksum: 19963, Flags: END
2023-12-15 23:33:56 [INFO] File received and saved.
2023-12-15 23:33:56 [INFO] Bytes Recv: 11968994 bytes
2023-12-15 23:33:56 [INFO] Time Taken: 77.922000 seconds
2023-12-15 23:33:56 [INFO] Average Speed: 153602.243269 bytes/s
2023-12-15 23:33:56 [RECV] Packet - SeqNum: 1201, AckNum: 0, Checksum: 12027, Flags: FIN
2023-12-15 23:33:56 [SEND] Packet - SeqNum: 1198, AckNum: 1202, Checksum: 38902, Flags: ACK
2023-12-15 23:33:56 [SEND] Packet - SeqNum: 1199, AckNum: 1202, Checksum: 30454, Flags: ACK FIN
2023-12-15 23:33:56 [RECV] Packet - SeqNum: 1202, AckNum: 1200, Checksum: 38390, Flags: ACK
2023-12-15 23:33:56 [INFO] Wavehand successful.

```

```

D:\study\大三\计算机网络_实验\3-3\2113997_齐明杰_编程作业3-3\UDPClient\UDPClient.exe
2023-12-15 23:33:54 [INFO] SendBuffer: 1175 1176(A) 1177 1178(A) 1179(A) 1180(A) 1181(A) 1182(A) 1183(A) 1184(A) 1185(A)
1186(A) 1187(A) 1188(A) 1189(A) 1190(A) 1191(A) 1192(A) 1193(A) 1194(A) 1195(A) 1196(A) 1197(A) 1198 1199
2023-12-15 23:33:54 [RECV] Packet - SeqNum: 1194, AckNum: 1199, Checksum: 40694, Flags: ACK
2023-12-15 23:33:54 [INFO] SendBuffer: 1175 1176(A) 1177 1178(A) 1179(A) 1180(A) 1181(A) 1182(A) 1183(A) 1184(A) 1185(A)
1186(A) 1187(A) 1188(A) 1189(A) 1190(A) 1191(A) 1192(A) 1193(A) 1194(A) 1195(A) 1196(A) 1197(A) 1198(A) 1199
2023-12-15 23:33:54 [RECV] Packet - SeqNum: 1195, AckNum: 1200, Checksum: 40182, Flags: ACK
2023-12-15 23:33:54 [INFO] SendBuffer: 1175 1176(A) 1177 1178(A) 1179(A) 1180(A) 1181(A) 1182(A) 1183(A) 1184(A) 1185(A)
1186(A) 1187(A) 1188(A) 1189(A) 1190(A) 1191(A) 1192(A) 1193(A) 1194(A) 1195(A) 1196(A) 1197(A) 1198(A) 1199(A)
2023-12-15 23:33:55 [WARN] Timeout, resend the First Packet in the Window! Current base: 1175
2023-12-15 23:33:55 [SEND] Packet - SeqNum: 1175, AckNum: 0, Checksum: 11081, Flags: DATA, Data Length: 10000
2023-12-15 23:33:55 [INFO] SendBuffer: 1175 1176(A) 1177 1178(A) 1179(A) 1180(A) 1181(A) 1182(A) 1183(A) 1184(A) 1185(A)
1186(A) 1187(A) 1188(A) 1189(A) 1190(A) 1191(A) 1192(A) 1193(A) 1194(A) 1195(A) 1196(A) 1197(A) 1198(A) 1199(A)
2023-12-15 23:33:55 [RECV] Packet - SeqNum: 1196, AckNum: 1176, Checksum: 46070, Flags: ACK
2023-12-15 23:33:56 [WARN] Timeout, resend the First Packet in the Window! Current base: 1177
2023-12-15 23:33:56 [SEND] Packet - SeqNum: 1177, AckNum: 0, Checksum: 14929, Flags: DATA, Data Length: 10000
2023-12-15 23:33:56 [INFO] SendBuffer: 1177 1178(A) 1179(A) 1180(A) 1181(A) 1182(A) 1183(A) 1184(A) 1185(A) 1186(A) 1187(A)
1188(A) 1189(A) 1190(A) 1191(A) 1192(A) 1193(A) 1194(A) 1195(A) 1196(A) 1197(A) 1198(A) 1199(A)
2023-12-15 23:33:56 [RECV] Packet - SeqNum: 1197, AckNum: 1178, Checksum: 45302, Flags: ACK
2023-12-15 23:33:56 [SEND] Packet - SeqNum: 1200, AckNum: 0, Checksum: 19963, Flags: END
2023-12-15 23:33:56 [INFO] File: 3.jpg
2023-12-15 23:33:56 [INFO] Bytes Sent: 11968994 bytes
2023-12-15 23:33:56 [INFO] Time Taken: 77.922000 seconds
2023-12-15 23:33:56 [INFO] Average Speed: 153602.243269 bytes/s
2023-12-15 23:33:56 [SEND] Packet - SeqNum: 1201, AckNum: 0, Checksum: 12027, Flags: FIN
2023-12-15 23:33:56 [RECV] Packet - SeqNum: 1198, AckNum: 1202, Checksum: 38902, Flags: ACK
2023-12-15 23:33:56 [INFO] File is not sending, RecvThread close.
2023-12-15 23:33:56 [RECV] Packet - SeqNum: 1199, AckNum: 1202, Checksum: 30454, Flags: ACK FIN
2023-12-15 23:33:56 [SEND] Packet - SeqNum: 1202, AckNum: 1200, Checksum: 38390, Flags: ACK
2023-12-15 23:33:56 [INFO] Wavehand successful.

```

- helloworld.txt

```

D:\study\大三\计算机网络\实验\3-3\2113997_齐明杰_编程作业3-3\UDPServer\UDPServer.exe
0 0
2023-12-15 23:35:25 [RECV] Packet - SeqNum: 147, AckNum: 0, Checksum: 48639, Flags: DATA, Data Length: 10000
2023-12-15 23:35:25 [SEND] Packet - SeqNum: 162, AckNum: 148, Checksum: 49662, Flags: ACK
2023-12-15 23:35:25 [INFO] RecvBuffer: 147 148 0 0 151 152 153 154 155 156 157 158 159 160 161 162 0 0 165 166 167 168 0
0 0
2023-12-15 23:35:26 [RECV] Packet - SeqNum: 149, AckNum: 0, Checksum: 48127, Flags: DATA, Data Length: 10000
2023-12-15 23:35:26 [SEND] Packet - SeqNum: 163, AckNum: 150, Checksum: 48894, Flags: ACK
2023-12-15 23:35:26 [INFO] RecvBuffer: 149 0 151 152 153 154 155 156 157 158 159 160 161 162 0 0 165 166 167 168 0 0 0 0
0
2023-12-15 23:35:27 [RECV] Packet - SeqNum: 150, AckNum: 0, Checksum: 8066, Flags: DATA, Data Length: 10000
2023-12-15 23:35:27 [SEND] Packet - SeqNum: 164, AckNum: 151, Checksum: 48382, Flags: ACK
2023-12-15 23:35:27 [INFO] RecvBuffer: 150 151 152 153 154 155 156 157 158 159 160 161 162 0 0 165 166 167 168 0 0 0 0 0
0
2023-12-15 23:35:28 [RECV] Packet - SeqNum: 163, AckNum: 0, Checksum: 44543, Flags: DATA, Data Length: 10000
2023-12-15 23:35:28 [SEND] Packet - SeqNum: 165, AckNum: 164, Checksum: 44798, Flags: ACK
2023-12-15 23:35:28 [INFO] RecvBuffer: 163 0 165 166 167 168 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2023-12-15 23:35:29 [RECV] Packet - SeqNum: 164, AckNum: 0, Checksum: 4482, Flags: DATA, Data Length: 10000
2023-12-15 23:35:29 [SEND] Packet - SeqNum: 166, AckNum: 165, Checksum: 44286, Flags: ACK
2023-12-15 23:35:29 [INFO] RecvBuffer: 164 165 166 167 168 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2023-12-15 23:35:29 [RECV] Packet - SeqNum: 169, AckNum: 0, Checksum: 21759, Flags: END
2023-12-15 23:35:29 [INFO] File received and saved.
2023-12-15 23:35:29 [INFO] Bytes Recv: 1655808 bytes
2023-12-15 23:35:29 [INFO] Time Taken: 37.859000 seconds
2023-12-15 23:35:29 [INFO] Average Speed: 43736.178980 bytes/s
2023-12-15 23:35:29 [RECV] Packet - SeqNum: 170, AckNum: 0, Checksum: 13823, Flags: FIN
2023-12-15 23:35:29 [SEND] Packet - SeqNum: 167, AckNum: 171, Checksum: 42494, Flags: ACK
2023-12-15 23:35:29 [SEND] Packet - SeqNum: 168, AckNum: 171, Checksum: 34046, Flags: ACK FIN
2023-12-15 23:35:29 [RECV] Packet - SeqNum: 171, AckNum: 169, Checksum: 41982, Flags: ACK
2023-12-15 23:35:29 [INFO] Wavehand successful.

```

```

D:\study\大三\计算机网络\实验\3-3\2113997_齐明杰_编程作业3-3\UDPCliet\UDPCliet.exe
2023-12-15 23:35:26 [WARN] Timeout, resend the First Packet in the Window! Current base: 149
2023-12-15 23:35:26 [SEND] Packet - SeqNum: 149, AckNum: 0, Checksum: 48127, Flags: DATA, Data Length: 10000
2023-12-15 23:35:26 [INFO] SendBuffer: 149 150 151(A) 152(A) 153(A) 154(A) 155(A) 156(A) 157(A) 158(A) 159(A) 160(A) 161(A) 162(A) 163 164 165(A) 166(A) 167(A) 168(A)
2023-12-15 23:35:26 [RECV] Packet - SeqNum: 163, AckNum: 150, Checksum: 48894, Flags: ACK
2023-12-15 23:35:27 [WARN] Timeout, resend the First Packet in the Window! Current base: 150
2023-12-15 23:35:27 [SEND] Packet - SeqNum: 150, AckNum: 0, Checksum: 8066, Flags: DATA, Data Length: 10000
2023-12-15 23:35:27 [INFO] SendBuffer: 150 151(A) 152(A) 153(A) 154(A) 155(A) 156(A) 157(A) 158(A) 159(A) 160(A) 161(A) 162(A) 163 164 165(A) 166(A) 167(A) 168(A)
2023-12-15 23:35:27 [RECV] Packet - SeqNum: 164, AckNum: 151, Checksum: 48382, Flags: ACK
2023-12-15 23:35:28 [WARN] Timeout, resend the First Packet in the Window! Current base: 163
2023-12-15 23:35:28 [SEND] Packet - SeqNum: 163, AckNum: 0, Checksum: 44543, Flags: DATA, Data Length: 10000
2023-12-15 23:35:28 [INFO] SendBuffer: 163 164 165(A) 166(A) 167(A) 168(A)
2023-12-15 23:35:28 [RECV] Packet - SeqNum: 165, AckNum: 164, Checksum: 44798, Flags: ACK
2023-12-15 23:35:29 [WARN] Timeout, resend the First Packet in the Window! Current base: 164
2023-12-15 23:35:29 [SEND] Packet - SeqNum: 164, AckNum: 0, Checksum: 4482, Flags: DATA, Data Length: 10000
2023-12-15 23:35:29 [INFO] SendBuffer: 164 165(A) 166(A) 167(A) 168(A)
2023-12-15 23:35:29 [RECV] Packet - SeqNum: 166, AckNum: 165, Checksum: 44286, Flags: ACK
2023-12-15 23:35:29 [SEND] Packet - SeqNum: 169, AckNum: 0, Checksum: 21759, Flags: END
2023-12-15 23:35:29 [INFO] File: helloworld.txt
2023-12-15 23:35:29 [INFO] Bytes Sent: 1655808 bytes
2023-12-15 23:35:29 [INFO] Time Taken: 37.859000 seconds
2023-12-15 23:35:29 [INFO] Average Speed: 43736.178980 bytes/s
2023-12-15 23:35:29 [SEND] Packet - SeqNum: 170, AckNum: 0, Checksum: 13823, Flags: FIN
2023-12-15 23:35:29 [INFO] File is not sending, RecvThread close.
2023-12-15 23:35:29 [RECV] Packet - SeqNum: 167, AckNum: 171, Checksum: 42494, Flags: ACK
2023-12-15 23:35:29 [RECV] Packet - SeqNum: 168, AckNum: 171, Checksum: 34046, Flags: ACK FIN
2023-12-15 23:35:29 [SEND] Packet - SeqNum: 171, AckNum: 169, Checksum: 41982, Flags: ACK
2023-12-15 23:35:29 [INFO] Wavehand successful.

```

7 总结与感想

在完成这次的实验中，我深入了解并实践了滑动窗口机制和选择重传（SR）协议，这对于我的网络编程技能和理解是一次极大的提升。

- 理论与实践的结合：**这次实验不仅仅是编写代码，更多的是将复杂的网络协议理论转化为实际的程序逻辑。这让我意识到理论知识的重要性，并且体会到了将理论应用到实践中的满足感。
- 问题解决能力的提升：**在实验过程中，我遇到了许多挑战，例如如何处理重复包，如何优化超时重传机制等。解决这些问题的过程锻炼了我的问题解决能力和调试技巧。
- 代码优化的重要性：**我意识到了代码优化的重要性。在实验中，我尝试使用不同的数据结构和算法来提高程序的效率和稳定性，这对我未来的编程工作是一个宝贵的经验。

