

计算机网络实验报告

Lab3-3 基于 UDP 服务设计可靠传输协议

网络安全空间学院 信息安全专业

2112492 刘修铭 1063

https://github.com/lxmliu2002/Computer_Networking

一、实验内容

利用数据报套接字在用户空间实现面向连接的可靠数据传输，功能包括：建立连接、差错检测、接收确认、超时重传等。采用基于滑动窗口的流量控制机制，发送窗口和接收窗口采取相同大小，支持选择确认，完成给定测试文件的传输。

二、协议设计

(一) 报文格式

在本次实验中，仿照 TCP 协议的报文格式进行了数据报设计，其中包括源端口号、目的端口号、序列号、确认号、消息数据长度、标志位、检测值以及数据包，其中标志位包括 FIN、CFH（是否为文件头部信息）、ACK、SYN 四位。

报文头部共 24Byte，数据段共 8168Byte，整个数据报大小为 8192Byte。

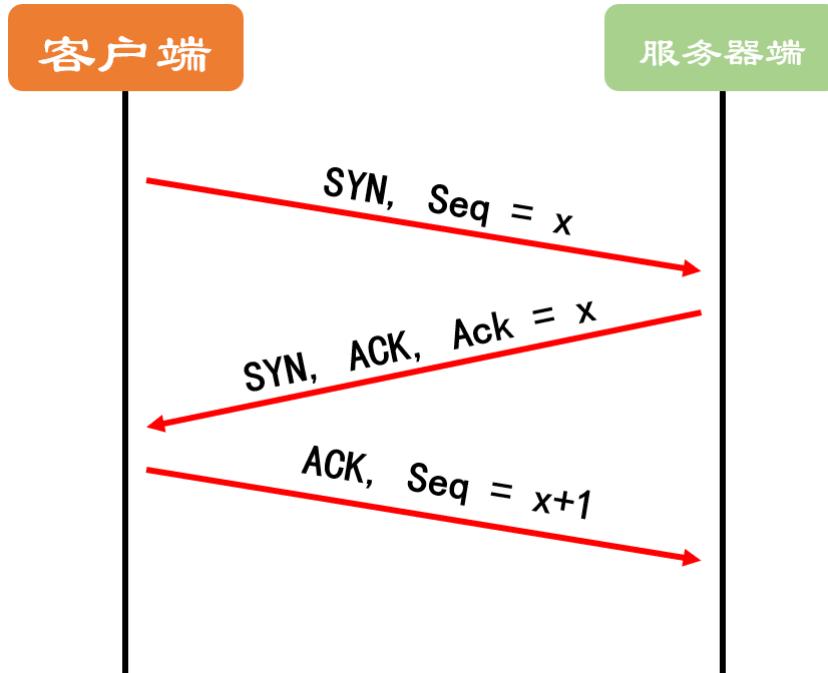
1	0	15 16	31	
2				
3		SrcPort		
4				
5		DstPort		
6				
7		Seq		
8				
9		Ack		
10				
11		win		
12				
13		Length		
14				
15		Flag		Check
16				
17		Data		
18				
19				
20		Flag		
21	0 1 2 3 4		15	

(二) 消息传输机制

本次实验对传统机制进行修改，确认消息的 Ack 为发送消息的 Seq。

1. 建立连接——三次握手

仿照 TCP 协议设计了连接的建立机制——三次握手，依旧使用停等机制，示意图如下：

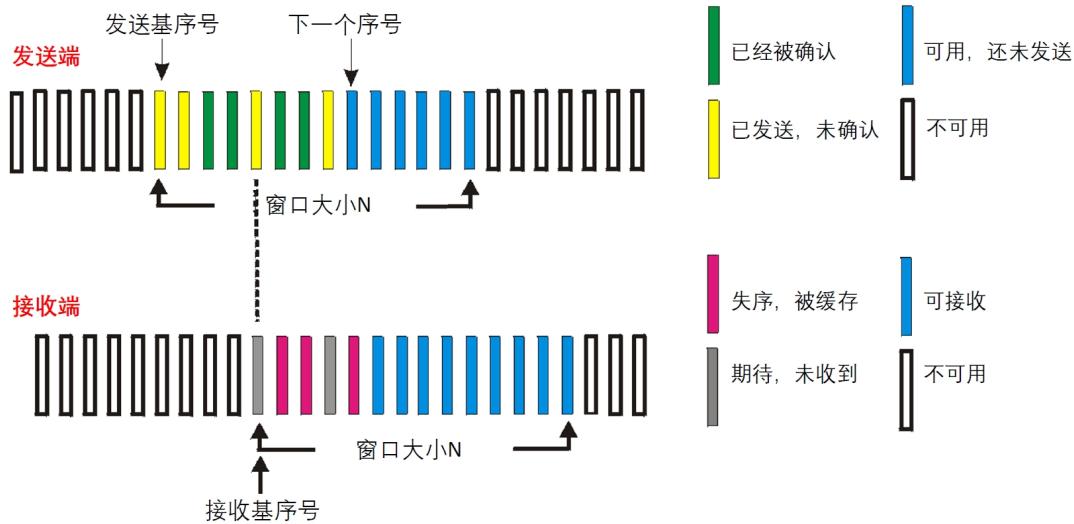


2. 差错检测

为了保证数据传输的可靠性，本次实验仿照 UDP 的校验和机制设计了差错检测机制。对消息头部和数据的所有 16 位字求和，然后对结果取反。算法原理同教材，不再赘述。

3. 滑动窗口与选择确认

按照实验要求，在本次实验中，基于 GBN 流水线协议，使用固定窗口大小，实现了流量控制机制。



在 Defs.h 中定义了窗口大小 Windows_Size。在 Client_SR.cpp 中，为了避免多线程导致的冲突，定义了基于原子操作的 Base_Seq 与 Next_Seq，用于标定窗口的位置；在 Server_SR.cpp 中定义了 Base_Seq，用来标定接收序列号。

发送端

- 发送端维护了一个 bool 类型的 Rec_Ack 数组，用于记录是否接受到对应的报文的 ACK。
- 发送端的发送缓冲区大小和窗口大小一致，即发送端最多一次性发送 Windows_Size 个报文。
- 通过两个指针来控制当前已经发送的报文序号和还未确认的报文序号。Base_Seq 指针表示的是当前已经确认的最大序列号，Next_Seq 指针表示的是当前已经发送的最大序列号。
- 每当发送端接收到回应的 ACK 的时候，将对应的 Rec_Ack 置为 true，表示已经收到对应的 ACK，同时 Is_Finish 自增。
 - 如果接收到的 Ack 与 Base_Seq 相等，则向后移动窗口至 Rec_Ack 为 false 的位置；
 - 如果连续接收到相同 Ack 的次数达到三次，将激发重传；
 - 此处仅重传窗口中所有 Rec_Ack 为 false 的数据；
 - 当 Is_Finish 的值与 Msg_Num 相等时，则将 Finish 写为 true，表示文件传输结束。

接收端

- 接收端维护了一个 bool 类型的 Rec_Ack 数组，用于记录是否接受到对应的报文；
- 接收端对接收到的报文进行判定，并记录接收到的时间
 - 如果接收到的数据包校验和检验未通过，将直接丢弃；
 - 如果收到的序号小于期望的序号，则丢弃该报文，继续接收其他报文；
 - 如果接收到的序号等于期望的序号，则将对应的 Rec_Ack 置为 true，并向后移动窗口至 Rec_Ack 为 false 的地方；
 - 如果接收到的序号大于期望的序号但小于窗口大小，则缓存该数据，并将其对应的 Rec_Ack 置为 true；
 - 如果接收到的序号超过窗口所能缓存的最大数量，则丢弃该数据；
 - 当接收时间超时时，将当前期望的 Ack 发送三次，刺激发送端进入重传状态；

- 特别的，如果接收到零数据包，则将当前期望的 Ack 发送三次，刺激发送端进入重传状态。

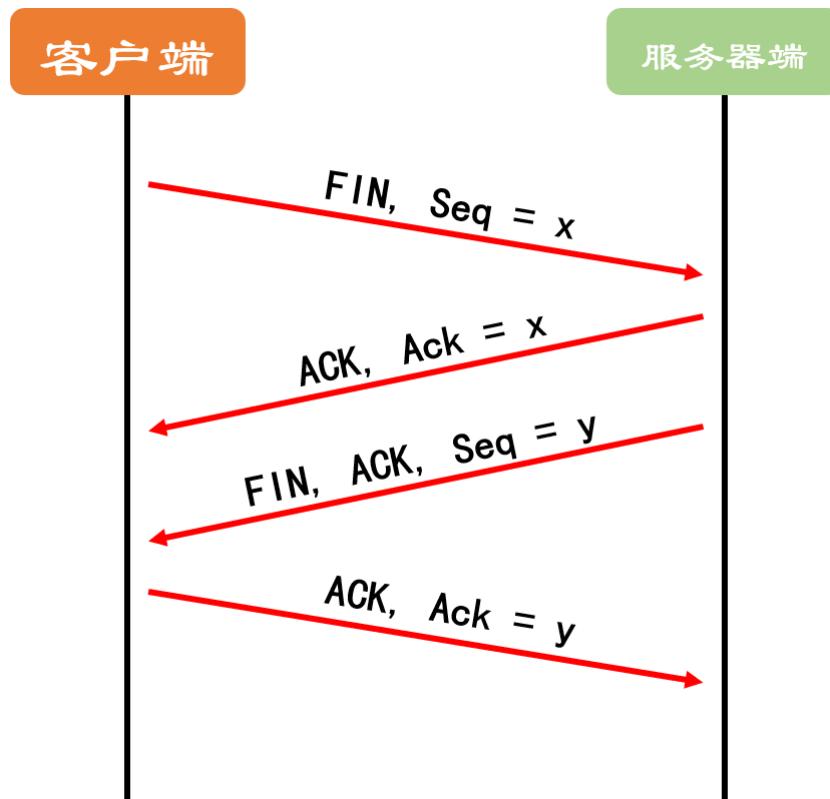
4. 超时重传

本次实验的超时重传部分主要有以下几种实现：

- 当接收端接收到零数据包且陷入循环时，将发送三次当前期望的 Ack，以此刺激发送端重新发送数据；
- 当接收端接收到的 Ack 与预期不符时，将发送期望的 Ack 到发送端，当发送端接收到相同的 Ack 超过三次时，将重新发送窗口中所有 Rec_Ack 为 false 的数据；
- 当接收端超过 Wait_Time 未接收到数据时，也会发送三次当前期望的 Ack，刺激发送端重新发送数据。

5. 断开连接——四次挥手（以发送端主动断开连接为例）

本次实验仿照 TCP 协议，设计了四次挥手断开连接机制，依旧使用停等机制，示意图如下：



6. 状态机

(1) 发送端

sender

data from above :

- ❖ if next available seq # in window, send pkt

timeout(n):

- ❖ resend pkt n, restart timer

ACK(n) in [sendbase,sendbase+N]:

- ❖ mark pkt n as received
- ❖ if n smallest unACKed pkt, advance window base to next unACKed seq #

(2) 接收端

receiver

pkt n in [rcvbase, rcvbase+N-1]

- send ACK(n)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

pkt n in [rcvbase-N,rcvbase-1]

- ACK(n)

otherwise:

- ignore

三、代码实现

(一) 协议设计

本次实验参考 oceanbase 设计，将头文件、宏定义、结构体等写入 `Defs.h` 文件中。

通过将标志位进行宏定义，便于后续使用。

```
1 #define FIN 0b1
2 #define CFH 0b10
3 #define ACK 0b100
4 #define SYN 0b1000
```

将协议报文包装成了 `Message` 结构体，并编写了系列函数，用来初始化结构体、设置标志位、差错检测等。

```
1 #pragma pack(1)
2 struct Message
3 {
4     uint32_t SrcPort;
5     uint32_t DstPort;
6     uint32_t Seq;
7     uint32_t Ack;
8     uint32_t Length;
9     uint16_t Flag;
10    uint16_t Check;
11    char Data[MSS];
12
13    Message() : SrcPort(0), DstPort(0), Seq(0), Ack(0), Length(0), Flag(0),
14    Check(0) { memset(this->Data, 0, MSS); }
15    void Set_CFH() { this->Flag |= CFH; }
16    bool Is_CFH() { return this->Flag & CFH; }
17    void Set_ACK() { this->Flag |= ACK; }
18    bool Is_ACK() { return this->Flag & ACK; }
19    void Set_SYN() { this->Flag |= SYN; }
20    bool Is_SYN() { return this->Flag & SYN; }
21    void Set_FIN() { this->Flag |= FIN; }
22    bool Is_FIN() { return this->Flag & FIN; }
23    bool CheckValid();
24    void Print_Message();
25};

# pragma pack()
```

按照前面叙述，实现了校验位的设置与检测函数。其原理同理论课相同，不再赘述。

```
1 void Message::Set_Check()
2 {
3     this->Check = 0;
4     uint32_t sum = 0;
5     uint16_t *p = (uint16_t *)this;
6     for (int i = 0; i < sizeof(*this) / 2; i++)
7     {
```

```

8         sum += *p++;
9         while (sum >> 16)
10        {
11            sum = (sum & 0xffff) + (sum >> 16);
12        }
13    }
14    this->Check = ~(sum & 0xffff);
15 }
16 bool Message::CheckValid()
17 {
18     uint32_t sum = 0;
19     uint16_t *p = (uint16_t *)this;
20     for (int i = 0; i < sizeof(*this) / 2; i++)
21     {
22         sum += *p++;
23         while (sum >> 16)
24         {
25             sum = (sum & 0xffff) + (sum >> 16);
26         }
27     }
28     return (sum & 0xffff) == 0xffff;
29 }
```

(二) 初始化

发送端与接收端结构相同，此处以发送端为例进行说明。

在 `Defs.h` 中定义了相关套接字等信息，在 `client.cpp` 中编写了 `client_Initial` 函数，初始化发送端网络连接与套接字，并按照连接状态，适时进行错误检测，输出运行日志。

```

1 SOCKET ClientSocket;
2 SOCKADDR_IN ClientAddr;
3 string ClientIP = "127.0.0.1";
4 int ClientAddrLen = sizeof(ClientAddr);
5
6 bool Client_Initial()
7 {
8     WSAStartup(MAKEWORD(2, 2), &wsaData);
9     if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2)
10    {
11        perror("[Client] Error in Initializing Socket DLL!\n");
12        exit(EXIT_FAILURE);
13    }
14    cout << "[Client]" << " Initializing Socket DLL is successful!\n";
15    ClientSocket = socket(AF_INET, SOCK_DGRAM, 0);
16    unsigned long on = 1;
17    ioctlsocket(ClientSocket, FIONBIO, &on);
18    if (ClientSocket == INVALID_SOCKET)
19    {
20        cout << "[Client]" << "Error in Creating Socket!\n";
21        exit(EXIT_FAILURE);
22    }
23 }
```

```

22     return false;
23 }
24 cout << "[Client] " << "Creating Socket is successful!\n";
25 ClientAddr.sin_family = AF_INET;
26 ClientAddr.sin_port = htons(client_Port);
27 ClientAddr.sin_addr.s_un.s_addr = inet_addr(clientIP.c_str());
28
29 if (bind(clientSocket, (SOCKADDR *)&ClientAddr, sizeof(SOCKADDR)) == SOCKET_ERROR)
30 {
31     cout << "[Client] " << "Error in Binding Socket!\n";
32     exit(EXIT_FAILURE);
33     return false;
34 }
35 cout << "[Client] " << "Binding Socket to port " << client_Port << " is
36 successful!\n\n";
37 RouterAddr.sin_family = AF_INET;
38 RouterAddr.sin_port = htons(Router_Port);
39 RouterAddr.sin_addr.s_un.s_addr = inet_addr(RouterIP.c_str());
40 return true;
41 }
```

(三) 建立连接——三次握手

1. 发送端

- 发送第一次握手消息，并开始计时，申请建立连接，然后等待接收第二次握手消息
 - 如果超时未收到，则重新发送
- 收到正确的第二次握手消息后，发送第三次握手消息

```

1 void Connect()
2 {
3     Message con_msg[3];
4     // * First-Way Handshake
5     con_msg[0].Seq = ++Seq;
6     con_msg[0].Set_SYN();
7     int re = Send(con_msg[0]);
8     float msg1_Send_Time = clock();
9     if (re)
10    {
11        con_msg[0].Print_Message();
12        // * Second-Way Handshake
13        while (true)
14        {
15            if (recvfrom(clientSocket, (char *)&con_msg[1], sizeof(con_msg[1]),
16            0, (SOCKADDR *)&RouterAddr, &RouterAddrLen) > 0)
17            {
18                con_msg[1].Print_Message();
19                if (!(con_msg[1].Is_ACK() && con_msg[1].Is_SYN() &&
20                  con_msg[1].CheckValid() && con_msg[1].Ack == con_msg[0].Seq))
21            }
```

```

19         {
20             cout << "Error Message!" << endl;
21             exit(EXIT_FAILURE);
22         }
23     Seq = con_msg[1].Seq;
24     break;
25 }
26 if ((clock() - msg1_Send_Time) > Wait_Time)
27 {
28     int re = Send(con_msg[0]);
29     msg1_Send_Time = clock();
30     if (re)
31     {
32         SetConsoleTextAttribute(hConsole, 12);
33         cout << "!Time Out! -- Send Message to Router! -- First-Way
Handshake" << endl;
34         con_msg[0].Print_Message();
35         SetConsoleTextAttribute(hConsole, 7);
36     }
37 }
38 }
39 }
40 // * Third-way Handshake
41 con_msg[2].Ack = con_msg[1].Seq;
42 con_msg[2].Seq = ++Seq;
43 con_msg[2].Set_ACK();
44 if (Send(con_msg[2]));) con_msg[2].Print_Message();
45 cout << "Third-way Handshake is successful!\n\n";
46 }

```

2. 接收端

- 接收正确的第一次握手消息，发送第二次握手消息，并开始计时，等待接收第三次握手消息
 - 如果超时未收到，则重新发送
- 接收到正确的第三次握手消息，连接成功建立

此处代码结构与发送端基本一致，为避免报告冗长，不再展示。

(四) 数据传输

为避免代码冗余，首先包装了消息发送函数 `Send`。

```

1 int Send(Message &msg)
2 {
3     msg.SrcPort = Client_Port;
4     msg.DstPort = Router_Port;
5     msg.Set_Check();
6     return sendto(clientSocket, (char *)&msg, sizeof(msg), 0, (SOCKADDR
*)&RouterAddr, RouterAddrLen);
7 }

```

1. 发送端

在 Defs.h 中定义了窗口大小 Windows_Size:

```
1 | #define Windows_Size 20
```

为了避免多线程引起的数据读写冲突，定义了一系列基于原子操作的操作数，同时也定义了一个 mutex 用于控制输出。

```
1 | atomic_int Base_Seq(1);
2 | atomic_int Next_Seq(1);
3 | atomic_int Header_Seq(0);
4 | atomic_int Count(0);
5 | int Msg_Num = 0;
6 | atomic_bool Re_Send(false);
7 | atomic_bool Finish(false);
8 | mutex mtx;
```

编写了 `Send_Message` 函数用于数据发送。首先输入文件路径，按照路径寻找文件，获取到文件的名称及大小等信息，并以二进制方式读取文件数据。

```
1 | strcpy(file_name, "");
2 | size_t found = file_path.find_last_of("\\\\");
3 | string file_name = file_path.substr(found + 1);
4 | ifstream file(file_path, ios::binary);
5 | if (!file.is_open())
6 | {
7 |     cout << "[Client] " << "Error in Opening File!" << endl;
8 |     exit(EXIT_FAILURE);
9 | }
10 | file.seekg(0, ios::end);
11 | file_length = file.tellg();
12 | file.seekg(0, ios::beg);
13 | if (file_length > pow(2, 32))
14 | {
15 |     cout << "[Client] " << "File is too large!" << endl;
16 |     exit(EXIT_FAILURE);
17 | }
```

接着编写了一个函数，用于多线程接收 Ack，并根据接收情况进行一些判定。

- 定义了一个用于记录接收的 Ack 的值的 Err_Ack_Num。
- 如果接收到的 Ack 在窗口内，则接收并将对应的 Rec_Ack 置为 true，并从 Base_Seq 起，向后移动窗口至 Rec_Ack 为 false 的位置；
- 接着计算 Rec_Ack 中为 true 的数量 Is_Finish，如果其等于 Msg_Num，表示所有发出去的数据包均收到确认的 ACK，则将 Finish 置为 true，表示发送结束；
- 如果接收到相同的 Ack 的次数超过三次，则将 Re_Send 置为 true，重新发送窗口中所有 Rec_Ack 为 false 的数据。

```

1 void Receive_Ack(bool *Rec_Ack)
2 {
3     int Err_Ack_Num = 0;
4     while (true)
5     {
6         Message ack_msg;
7         if (recvfrom(clientSocket, (char *) &ack_msg, sizeof(ack_msg), 0,
8 (SOCKADDR *)&RouterAddr, &RouterAddrLen))
9         {
10            if (ack_msg.Is_ACK() && ack_msg.CheckValid())
11            {
12                lock_guard<mutex> lock(mtx);
13                ack_msg.Print_Message();
14                cout << "Receive Ack = " << ack_msg.Ack << endl;
15                Rec_Ack[ack_msg.Ack - Header_Seq - 1] = true;
16                int tmp = Base_Seq - 1;
17                for (int j = tmp; j < tmp + windows_size && Rec_Ack[j]; j++)
18                    Base_Seq++;
19                Print_Window(Base_Seq + Header_Seq, Next_Seq + Header_Seq);
20                int Is_Finish = 0;
21                for (int j = 0; j < Msg_Num + 1; j++)
22                {
23                    if (Rec_Ack[j]) Is_Finish++;
24                    else break;
25                }
26                if (Is_Finish == Msg_Num + 1)
27                {
28                    Finish = true;
29                    return;
30                }
31                if (Err_Ack_Num != ack_msg.Ack)
32                {
33                    Err_Ack_Num = ack_msg.Ack;
34                    Count = 0;
35                }
36                else
37                {
38                    Count++;
39                    if (Count == 3)
40                    {
41                        Re_Send = true;
42                        Count = 0;
43                    }
44                }
45            }
46        }
47    }
}

```

在发送文件时，首先按照给定路径对文件进行读取。

```

1 size_t found = file_path.find_last_of("\\\\");
2 string file_name = file_path.substr(found + 1);
3 ifstream file(file_path, ios::binary);
4 if (!file.is_open())
5 {
6     cout << "Error in Opening File!" << endl;
7     exit(EXIT_FAILURE);
8 }
9 file.seekg(0, ios::end);
10 file_length = file.tellg();
11 file.seekg(0, ios::beg);
12 if (file_length > pow(2, 32))
13 {
14     cout << "File is too large!" << endl;
15     exit(EXIT_FAILURE);
16 }

```

接着计算待发送的消息总数 Msg_Num，创建新线程用于接收 Ack，同时按照计算出来的消息总数创建一个 Message 数组，作为缓冲区记录发送的数据；另创建一个 bool 类型的数组 Rec_Ack，用于记录发送的报文是否被确认。

```

1 int complete_num = file_length / MSS;
2 int last_length = file_length % MSS;
3 Header_Seq = Seq;
4 if (last_length != 0) Msg_Num = complete_num + 1;
5 else Msg_Num = complete_num;
6 bool *Rec_Ack = new bool[Msg_Num + 1];
7 memset(Rec_Ack, 0, Msg_Num + 1);
8 thread Receive_Ack_Thread(Receive_Ack, Rec_Ack);
9 Message *data_msg = new Message[Msg_Num + 1];

```

接着循环发送数据：

- 如果 Finish 为 true，说明文件发送完毕，则直接终止发送
- 如果 Re_Send 为 true，说明出现丢失，需要重新发送
 - 此时借助前面定义的 Message 缓冲区以及 Rec_Ack 数组，定位并重新发送窗口中的所有 Rec_Ack 为 false 的数据

```

1 while (!Finish)
2 {
3     if (Re_Send)
4     {
5         for (int i = 0; (i < Next_Seq - Base_Seq) && (!Rec_Ack[Base_Seq + i - 1]); i++)
6         {
7             lock_guard<mutex> lock(mtx);
8             if (Send(data_msg[Base_Seq + i - 1]))
9             {
10                 SetConsoleTextAttribute(hConsole, 12);
11             }
12         }
13     }
14 }

```

```

11             cout << "!Re_Send! -- Send Message to Router!" << endl;
12             SetConsoleTextAttribute(hConsole, 7);
13             data_msg[Base_Seq + i - 1].Print_Message();
14         }
15     else
16     {
17         cout << "Error in Sending Message!" << endl;
18         exit(EXIT_FAILURE);
19     }
20 }
21 Re_Send = false;
22 }
23 if (Next_Seq < Base_Seq + windows_Size && Next_Seq - 1 <= Msg_Num)
24 {
25     if (Next_Seq == 1)
26     {
27         lock_guard<mutex> lock(mtx);
28         strcpy(data_msg[Next_Seq - 1].Data, file_name.c_str());
29         data_msg[Next_Seq - 1].Data[strlen(data_msg[Next_Seq - 1].Data)] =
30         '\0';
31         data_msg[Next_Seq - 1].Length = file_length;
32         data_msg[Next_Seq - 1].Seq = ++Seq;
33         data_msg[Next_Seq - 1].Set_CFH();
34     }
35     else if (Next_Seq - 1 == Msg_Num && last_length)
36     {
37         lock_guard<mutex> lock(mtx);
38         file.read(data_msg[Next_Seq - 1].Data, last_length);
39         data_msg[Next_Seq - 1].Length = last_length;
40         data_msg[Next_Seq - 1].Seq = ++Seq;
41     }
42     else
43     {
44         lock_guard<mutex> lock(mtx);
45         file.read(data_msg[Next_Seq - 1].Data, MSS);
46         data_msg[Next_Seq - 1].Length = MSS;
47         data_msg[Next_Seq - 1].Seq = ++Seq;
48     }
49     if (Next_Seq % 17 == 0)
50     {
51         Next_Seq++;
52         continue;
53     }
54     if (Next_Seq % 19 == 0)
55     {
56         sleep(10);
57         Next_Seq++;
58         continue;
59     }
60     if (Send(data_msg[Next_Seq - 1]))
61     {
62         lock_guard<mutex> lock(mtx);

```

```

62         data_msg[Next_Seq - 1].Print_Message();
63     }
64 }
65 else
66 {
67     lock_guard<mutex> lock(mtx);
68     cout << "Error in Sending Message!" << endl;
69     exit(EXIT_FAILURE);
70 }
71 }
72 }
```

最后重置相关操作数。

```

1 lock_guard<mutex> lock(mtx);
2 Next_Seq = 1;
3 Base_Seq = 1;
4 Finish = false;
5 Re_Send = false;
6 Header_Seq = 0;
7 Msg_Num = 0;
```

2. 接收端

- 首先接收发送端发送的文件头部信息，并根据 CFH 标志位进行确认。
- 接着计算出待接收的消息的总数，并以此创建 Message 数组作为接收缓冲区以及一个 bool 类型的 Rec_Ack 数组，用于记录是否接受到对应的报文；
- 接着循环接收数据，并记录接收到的时间
 - 如果收到的序号小于期望的序号，则丢弃该报文，继续接收其他报文；
 - 如果接收到的序号等于期望的序号，则将对应的 Rec_Ack 置为 true，并向后移动窗口至 Rec_Ack 为 false 的地方；
 - 如果接收到的序号大于期望的序号但小于窗口大小，则缓存该数据，并将其对应的 Rec_Ack 置为 true；
 - 如果接收到的序号超过窗口所能缓存的最大数量，则丢弃该数据；
 - 当接收时间超时时，将当前期望的 Ack 发送三次，刺激发送端进入重传状态；
 - 特别的，如果接收到零数据包，则将当前期望的 Ack 发送三次，刺激发送端进入重传状态。
- 接收完成后，将数据写入到文件中。

为避免报告冗长，仅展示主要的接收部分代码。

```

1 for (int i = 0; i < Msg_Num; i++)
2 {
3     while (true)
4     {
5         Message tmp_msg;
6         if(clock() - Last_Time > 5)
```

```

7      {
8          Message reply_msg;
9          reply_msg.Ack = Base_Seq - 1;
10         reply_msg.Set_ACK();
11         for (int j = 0; j < 3; j++)
12         {
13             if (Send(reply_msg))
14             {
15                 SetConsoleTextAttribute(hConsole, 12);
16                 cout << "Time Out! Trying to Get New Message! Base_Seq = "
17 << Base_Seq << endl;
18                 SetConsoleTextAttribute(hConsole, 7);
19             }
20         }
21         if (recvfrom(ServerSocket, (char *)&tmp_msg, sizeof(tmp_msg), 0,
22 (SOCKADDR *)&RouterAddr, &RouterAddrLen))
23         {
24             tmp_msg.Print_Message();
25             Last_Time = clock();
26             if (tmp_msg.Seq == 0)
27             {
28                 Message reply_msg;
29                 reply_msg.Ack = Base_Seq - 1;
30                 reply_msg.Set_ACK();
31                 for (int j = 0; j < 3; j++)
32                 {
33                     if (Send(reply_msg))
34                     {
35                         SetConsoleTextAttribute(hConsole, 12);
36                         cout << "Zero Message! Trying to Get New Message!
37 Base_Seq = " << Base_Seq << endl;
38                         reply_msg.Print_Message();
39                         SetConsoleTextAttribute(hConsole, 7);
40                     }
41                 }
42             }
43             else
44             {
45                 if (tmp_msg.Seq < Base_Seq)
46                 {
47                     Message reply_msg;
48                     reply_msg.Ack = tmp_msg.Seq;
49                     reply_msg.Set_ACK();
50                     if (Send(reply_msg))
51                     {
52                         reply_msg.Print_Message();
53                         data_msg[tmp_msg.Seq - Header_Seq - 2] = tmp_msg;
54                         Rec_Ack[tmp_msg.Seq - Header_Seq-2] = true;
55                     }
56                     continue;
57                 }
58             }
59         }
60     }
61 }
```

```

56         else
57     {
58         tmp_msg.Print_Message();
59         if (tmp_msg.Checkvalid())
60     {
61         if (tmp_msg.Seq == Base_Seq)
62     {
63         Message reply_msg;
64         reply_msg.Ack = tmp_msg.Seq;
65         reply_msg.Set_ACK();
66         if (Send(reply_msg))
67     {
68             reply_msg.Print_Message();
69             data_msg[tmp_msg.Seq - Header_Seq - 2] =
70             tmp_msg;
71             Rec_Ack[tmp_msg.Seq - Header_Seq - 2] = true;
72             for (int j = tmp_msg.Seq - Header_Seq - 2; j <
73             tmp_msg.Seq - Header_Seq - 2 + Windows_Size && j < Msg_Num; j++)
74         {
75             if (Rec_Ack[j] == true) Base_Seq++;
76             else break;
77         }
78     }
79     else
80     {
81         cout << "Error in Sending Message!";
82         exit(EXIT_FAILURE);
83     }
84     else if (tmp_msg.Seq > Base_Seq && tmp_msg.Seq <
Base_Seq + Windows_Size && tmp_msg.Seq <= Msg_Num + Header_Seq + 1)
85     {
86         Message reply_msg;
87         reply_msg.Ack = tmp_msg.Seq;
88         reply_msg.Set_ACK();
89         if (Send(reply_msg))
90     {
91         reply_msg.Print_Message();
92         data_msg[tmp_msg.Seq - Header_Seq - 2] =
93         tmp_msg;
94         Rec_Ack[tmp_msg.Seq - Header_Seq - 2] = true;
95         break;
96     }
97     else
98     {
99         cout << "Error in Sending Message!";
100        exit(EXIT_FAILURE);
101    }
102    else continue;
103}

```

```

104             else continue;
105         }
106     }
107 }
108 }
109 }
```

(五) 断开连接——四次挥手 (以发送端主动断开连接为例)

1. 发送端

- 发送第一次挥手消息，并开始计时，提出断开连接，然后等待接收第二次挥手消息
 - 如果超时未收到，则重新发送
- 收到正确的第二次挥手消息后，等待接收第三次挥手消息
- 接收到正确的第三次挥手消息，输出日志，准备断开连接
- 再等待 `2 * wait_Time` 时间（确保消息发送完毕），断开连接

```

1 void Disconnect() // * Client端主动断开连接
2 {
3     Message discon_msg[4];
4
5     // * First-Way Wavehand
6     discon_msg[0].Seq = ++Seq;
7     discon_msg[0].Set_FIN();
8     int re = Send(discon_msg[0]);
9     float dismsg0_Send_Time = clock();
10    if (re) discon_msg[0].Print_Message();
11    // * Second-way Wavehand
12    while (true)
13    {
14        if (recvfrom(clientSocket, (char *)&discon_msg[1],
15                     sizeof(discon_msg[1]), 0, (SOCKADDR *)&RouterAddr, &RouterAddrLen) > 0)
16        {
17            discon_msg[1].Print_Message();
18            if (discon_msg[1].Seq < Seq + 1) continue;
19            else if (!(discon_msg[1].Is_ACK() && discon_msg[1].CheckValid() &&
20                      discon_msg[1].Seq == Seq + 1 && discon_msg[1].Ack == discon_msg[0].Seq))
21            {
22                cout << "Error Message!" << endl;
23                exit(EXIT_FAILURE);
24            }
25            Seq = discon_msg[1].Seq;
26            break;
27        }
28        if ((clock() - dismsg0_Send_Time) > Wait_Time)
29        {
30            SetConsoleTextAttribute(hConsole, 12);
31            cout << "!Time Out! -- First-Way Wavehand" << endl;
32            int re = Send(discon_msg[0]);
```

```

31         dismsg0_Send_Time = clock();
32         if (re)
33         {
34             discon_msg[0].Print_Message();
35             SetConsoleTextAttribute(hConsole, 7);
36         }
37     }
38 }
39 // * Third-way Wavehand
40 while (true)
41 {
42     if (recvfrom(ClientSocket, (char *)&discon_msg[2],
43     sizeof(discon_msg[2]), 0, (SOCKADDR *)&RouterAddr, &RouterAddrLen) > 0)
44     {
45         discon_msg[2].Print_Message();
46         if (!(discon_msg[2].Is_ACK() && discon_msg[2].Is_FIN() &&
47         discon_msg[2].CheckValid() && discon_msg[2].Seq == Seq + 1 && discon_msg[2].Ack
48         == discon_msg[1].Seq))
49         {
50             cout << "Error Message!" << endl;
51             exit(EXIT_FAILURE);
52         }
53     }
54     // * Fourth-way Wavehand
55     discon_msg[3].Ack = discon_msg[2].Seq;
56     discon_msg[3].Set_ACK();
57     discon_msg[3].Seq = ++Seq;
58     if (Send(discon_msg[3]))
59     {
60         discon_msg[3].Print_Message();
61     }
62     cout << "Fourth-Way Wavehand is successful!" << endl;
63     Wait_Exit();
64 }
65 void Wait_Exit()
66 {
67     Message exit_msg;
68     float exit_msg_time = clock();
69     while (clock() - exit_msg_time < 2 * Wait_Time)
70     {
71         if (recvfrom(ClientSocket, (char *)&exit_msg, sizeof(exit_msg), 0,
72         (SOCKADDR *)&RouterAddr, &RouterAddrLen))
73         {
74             Seq = exit_msg.Seq;
75             exit_msg.Ack = exit_msg.Seq;
76             exit_msg.Set_ACK();
77             exit_msg.Seq = ++Seq;
78             Send(exit_msg);
79         }

```

```

79    }
80    closesocket(clientSocket);
81    WSACleanup();
82    cout << "Client is closed!" << endl;
83 }

```

2. 接收端

- 接收正确第一次挥手消息，发送第二次挥手消息，同意断开连接
- 发送第三次挥手消息，并开始计时，然后等待接收第四次挥手消息
 - 如果超时未收到，则重新发送
- 接收到正确的第四次挥手消息，输出日志，断开连接

```

1 void Disconnect() // * Router端主动断开连接
2 {
3     Message discon_msg[4];
4     while (true)
5     {
6         // * First-Way Wavehand
7         if (recvfrom(ServerSocket, (char *)&discon_msg[0],
8             sizeof(discon_msg[0]), 0, (SOCKADDR *)&RouterAddr, &RouterAddrLen) > 0)
9         {
10             discon_msg[0].Print_Message();
11             if (!discon_msg[0].Is_FIN() && discon_msg[0].Checkvalid())
12             {
13                 cout << "Error Message!" << endl;
14                 exit(EXIT_FAILURE);
15             }
16             Seq = discon_msg[0].Seq;
17         }
18         // * Second-Way Wavehand
19         discon_msg[1].Ack = discon_msg[0].Seq;
20         discon_msg[1].Seq = ++Seq;
21         discon_msg[1].Set_ACK();
22         if (Send(discon_msg[1])) discon_msg[1].Print_Message();
23         break;
24     }
25     // * Third-way Wavehand
26     discon_msg[2].Ack = discon_msg[1].Seq;
27     discon_msg[2].Seq = ++Seq;
28     discon_msg[2].Set_ACK();
29     discon_msg[2].Set_FIN();
30     int re = Send(discon_msg[2]);
31     float dismsg3_Send_Time = clock();
32     if (re) discon_msg[2].Print_Message();
33     // * Fourth-way Wavehand
34     while (true)
35     {
36         if (recvfrom(ServerSocket, (char *)&discon_msg[3],
37             sizeof(discon_msg[3]), 0, (SOCKADDR *)&RouterAddr, &RouterAddrLen) > 0)
38         {
39             discon_msg[3].Print_Message();
40             if (discon_msg[3].Is_FIN() && discon_msg[3].Checkvalid())
41             {
42                 cout << "Client is closed!" << endl;
43                 closesocket(ServerSocket);
44                 WSACleanup();
45                 exit(EXIT_SUCCESS);
46             }
47         }
48     }
49 }

```

```

36     {
37         discon_msg[3].Print_Message();
38         if (discon_msg[3].Seq < Seq + 1) continue;
39         else if (!(discon_msg[3].Is_ACK() && discon_msg[3].CheckValid() &&
40             discon_msg[3].Seq == Seq + 1 && discon_msg[3].Ack == discon_msg[2].Seq))
41         {
42             cout << "Error Message!" << endl;
43             exit(EXIT_FAILURE);
44         }
45         Seq = discon_msg[3].Seq;
46         cout << "Fourth-Way Wavehand is successful!\n\n";
47         break;
48     }
49     if ((clock() - dismsg3_Send_Time) > Wait_Time)
50     {
51         int re = Send(discon_msg[2]);
52         dismsg3_Send_Time = clock();
53         if (re)
54         {
55             HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
56             cout << "!Time Out! -- Send Message to Router! -- Third-Way
57             Wavehand" << endl;
58             discon_msg[2].Print_Message();
59             SetConsoleTextAttribute(hConsole, 7);
60         }
61     }
62     Exit();
63 void Exit()
64 {
65     closesocket(ServerSocket);
66     WSACleanup();
67     cout << "Server is closed!" << endl;
68 }
```

四、传输测试与性能分析

(一) 传输测试

本次实验中，手动编写了丢包与时延模拟。

```

1 if (Next_Seq % 17 == 0)
2 {
3     Next_Seq++;
4     continue;
5 }
6 if (Next_Seq % 19 == 0)
7 {
8     Sleep(10);
9     Next_Seq++;
10    continue;
11 }

```

1. 本机测试

本次实验在本机上实现了给定测试文件的测试

(1) 三次握手

可以看到，发送端与接收端成功完成三次握手，建立连接。

The image shows two separate Windows PowerShell sessions. Both sessions start with initializing the socket DLL and creating a socket. They then bind the socket to port 65432. The left session shows the server's perspective, where it waits for a connection. The right session shows the client's perspective, where it tries to connect to the server. Both sessions then proceed through the three stages of the handshake: SYN, SYN+ACK, and ACK. Finally, they both prompt the user to select a function to use, with options 1. Receive File and 2. Exit.

(2) helloworld.txt

传输完成，可以看到，累计用时 2455 ms，吞吐率为 632.31 Byte/ms。

The image shows two Windows PowerShell sessions. The left session shows the server sending multiple segments of the file to the client. The right session shows the client receiving these segments and acknowledging them. The exchange continues until the entire file is transferred. Both sessions end with a summary message indicating the file has been finished receiving and the total send time and speed.

右键查看文件属性，可以看到传输前后文件大小没有发生改变；打开文件，可以看到文件成功打开，说明传输无误。

(3) 1.jpg

红色输出即为丢包部分，可以看到能够在丢包情况下实现数据传输。

传输完成，可以看到，累计用时 2904 ms，吞吐率为 639.584 Byte/ms。

```
Windows PowerShell x + 
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 225 ] ACK
Message [SrcPort: 0 ] [DstPort: 0 ] [Length: 0 ] [Seq: 0 ]
Zero Message! Trying to Get New Message! Base_Seq = 226
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 225 ] ACK
Zero Message! Trying to Get New Message! Base_Seq = 226
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 225 ] ACK
Zero Message! Trying to Get New Message! Base_Seq = 226
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 225 ] ACK
Message [SrcPort: 0 ] [DstPort: 0 ] [Length: 0 ] [Seq: 0 ]
Zero Message! Trying to Get New Message! Base_Seq = 226
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 225 ] ACK
Zero Message! Trying to Get New Message! Base_Seq = 226
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 225 ] ACK
Zero Message! Trying to Get New Message! Base_Seq = 226
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 225 ] ACK
Message [SrcPort: 0 ] [DstPort: 0 ] [Length: 0 ] [Seq: 0 ]
Zero Message! Trying to Get New Message! Base_Seq = 226
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 225 ] ACK
Zero Message! Trying to Get New Message! Base_Seq = 226
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 225 ] ACK
Zero Message! Trying to Get New Message! Base_Seq = 226
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 225 ] ACK
Message [SrcPort: 54321 ] [DstPort: 65432 ] [Length: 8168 ] [Seq: 226 ]
Message [SrcPort: 54321 ] [DstPort: 65432 ] [Length: 8168 ] [Seq: 226 ]
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 226 ] ACK
Finish Receiving!
Please select the function you want to use:
1. Receive File
2. Exit
> |
```

```
Windows PowerShell x + 
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 225 ] ACK
Receive Ack = 225
[ Window ] Base: 226 Next: 233 No_Verify
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 225 ] ACK
Receive Ack = 225
[ Window ] Base: 226 Next: 233 No_Verify
!Re_Send! -- Send Message to Router!
Message [SrcPort: 54321 ] [DstPort: 65432 ] [Length: 8168 ] [Seq: 226 ]
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 225 ] ACK
Receive Ack = 225
[ Window ] Base: 226 Next: 233 No_Verify
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 225 ] ACK
Receive Ack = 225
[ Window ] Base: 226 Next: 233 No_Verify
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 225 ] ACK
Receive Ack = 225
[ Window ] Base: 226 Next: 233 No_Verify
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 225 ] ACK
Receive Ack = 225
[ Window ] Base: 226 Next: 233 No_Verify
!Re_Send! -- Send Message to Router!
Message [SrcPort: 54321 ] [DstPort: 65432 ] [Length: 8168 ] [Seq: 226 ]
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 226 ] ACK
Receive Ack = 226
[ Window ] Base: 233 Next: 233 No_Verify
Finish Sending File!
Send Time: 2904 ms
Send Speed: 639.584 Byte/ms

Please select the function you want to use:
1. Send File
2. Exit
> |
```

右键查看文件属性，可以看到传输前后文件大小没有发生改变；打开文件，可以看到文件成功打开，说明传输无误。

(4) 2.jpg

传输完成，可以看到，累计用时 4854 ms，吞吐率为 1215.18 Byte/ms。

```
Windows PowerShell x + v
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 948 ] ACK
Message [SrcPort: 54321 ] [DstPort: 65432 ] [Length: 8168 ] [Seq: 949 ]
Message [SrcPort: 54321 ] [DstPort: 65432 ] [Length: 8168 ] [Seq: 949 ]
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 949 ] ACK
Message [SrcPort: 54321 ] [DstPort: 65432 ] [Length: 8168 ] [Seq: 950 ]
Message [SrcPort: 54321 ] [DstPort: 65432 ] [Length: 8168 ] [Seq: 950 ]
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 950 ] ACK
Message [SrcPort: 54321 ] [DstPort: 65432 ] [Length: 8168 ] [Seq: 951 ]
Message [SrcPort: 54321 ] [DstPort: 65432 ] [Length: 8168 ] [Seq: 951 ]
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 951 ] ACK
Message [SrcPort: 54321 ] [DstPort: 65432 ] [Length: 8168 ] [Seq: 952 ]
Message [SrcPort: 54321 ] [DstPort: 65432 ] [Length: 8168 ] [Seq: 952 ]
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 952 ] ACK
Message [SrcPort: 54321 ] [DstPort: 65432 ] [Length: 8168 ] [Seq: 953 ]
Message [SrcPort: 54321 ] [DstPort: 65432 ] [Length: 8168 ] [Seq: 953 ]
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 953 ] ACK
Message [SrcPort: 54321 ] [DstPort: 65432 ] [Length: 8168 ] [Seq: 954 ]
Message [SrcPort: 54321 ] [DstPort: 65432 ] [Length: 8168 ] [Seq: 954 ]
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 954 ] ACK
Message [SrcPort: 54321 ] [DstPort: 65432 ] [Length: 8168 ] [Seq: 955 ]
Message [SrcPort: 54321 ] [DstPort: 65432 ] [Length: 8168 ] [Seq: 955 ]
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 955 ] ACK
Message [SrcPort: 54321 ] [DstPort: 65432 ] [Length: 1209 ] [Seq: 956 ]
Message [SrcPort: 54321 ] [DstPort: 65432 ] [Length: 1209 ] [Seq: 956 ]
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 956 ] ACK
Finish Receiving!
Please select the function you want to use:
1. Receive File
2. Exit
>
```

```
Windows PowerShell x + v
[ Window ] Base: 950 Next: 957 No_Ver
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 950 ] ACK
Receive Ack = 950
[ Window ] Base: 951 Next: 957 No_Ver
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 951 ] ACK
Receive Ack = 951
[ Window ] Base: 952 Next: 957 No_Ver
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 952 ] ACK
Receive Ack = 952
[ Window ] Base: 953 Next: 957 No_Ver
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 953 ] ACK
Receive Ack = 953
[ Window ] Base: 954 Next: 957 No_Ver
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 954 ] ACK
Receive Ack = 954
[ Window ] Base: 955 Next: 957 No_Ver
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 955 ] ACK
Receive Ack = 955
[ Window ] Base: 956 Next: 957 No_Ver
Message [SrcPort: 65432 ] [DstPort: 54321 ] [Length: 0 ] [Seq: 0 ] [Ack: 956 ] ACK
Receive Ack = 956
[ Window ] Base: 957 Next: 957 No_Ver
Finish Sending File!
Send Time: 4854 ms
Send Speed: 1215.18 Byte/ms
Please select the function you want to use:
1. Send File
2. Exit
> |
```

右键查看文件属性，可以看到传输前后文件大小没有发生改变；打开文件，可以看到文件成功打开，说明传输无误。

(5) 3.jpg

传输完成，可以看到，累计用时 7529 ms，吞吐率为 1589.72 Byte/ms。

```
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 0] [Ack: 2607] ACK
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2608]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2608]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 0] [Ack: 2608] ACK
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2609]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2609]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 0] [Ack: 2609] ACK
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2609]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2609]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 0] [Ack: 2609] ACK
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2610]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2610]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 0] [Ack: 2610] ACK
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2610]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2610]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 0] [Ack: 2610] ACK
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2611]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2611]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 0] [Ack: 2611] ACK
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2611]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2611]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 0] [Ack: 2611] ACK
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2612]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2612]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 0] [Ack: 2612] ACK
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2612]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2612]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 0] [Ack: 2612] ACK
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2613]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2613]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 0] [Ack: 2613] ACK
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2613]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2613]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 0] [Ack: 2613] ACK
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2614]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2614]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 0] [Ack: 2614] ACK
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2614]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 8168] [Seq: 2614]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 0] [Ack: 2614] ACK
Message [SrcPort: 54321] [DstPort: 65432] [Length: 2874] [Seq: 2615]
Message [SrcPort: 54321] [DstPort: 65432] [Length: 2874] [Seq: 2615]
Message [SrcPort: 65432] [DstPort: 54321] [Length: 0] [Seq: 0] [Ack: 2615] ACK
Finish Receiving!
Please select the function you want to use:
1. Receive File
2. Exit
>
```

```
[ Window ] Base: 2609 Next: 2616 No_Verify_Num: 2 Residue: 18
Message [SrcPort: 65432] [DstPort: 54321] [Length: 0] [Seq: 3] [Ack: 2] ACK
Third-Way Handshake is successful!
Please select the function you want to use:
1. Send File
2. Exit
> 2
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 4] FIN
Message [SrcPort: 65432] [DstPort: 54321] [Length: 0] [Seq: 5] [Ack: 4] ACK
Message [SrcPort: 65432] [DstPort: 54321] [Length: 0] [Seq: 6] [Ack: 5] ACK FIN
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 4] FIN
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 4] FIN
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 7] [Ack: 6] ACK
Fourth-Way Wavehand is successful!
Server is closed!
PS E:\刘修铭\南开大学\个人材料\课程\2023-2024 第1学期\计算机网络 张建忠, 徐敬东\Computer...
```

```
Please select the function you want to use:
1. Send File
2. Exit
> 1
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 4] FIN
!Time Out! -- First-Way Wavehand
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 4] FIN
!Time Out! -- First-Way Wavehand
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 4] FIN
Message [SrcPort: 65432] [DstPort: 54321] [Length: 0] [Seq: 5] [Ack: 4] ACK
Message [SrcPort: 65432] [DstPort: 54321] [Length: 0] [Seq: 6] [Ack: 5] ACK FIN
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 7] [Ack: 6] ACK
Fourth-Way Wavehand is successful!
Client is closed!
PS E:\刘修铭\南开大学\个人材料\课程\2023-2024 第1学期\计算机网络 张建忠, 徐敬东\Computer...
```

右键查看文件属性，可以看到传输前后文件大小没有发生改变；打开文件，可以看到文件成功打开，说明传输无误。

(6) 四次挥手

可以看到发送端与接收端成功完成四次挥手，断开连接。

```
Please select the function you want to use:
1. Receive File
2. Exit
> 2
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 4] FIN
Message [SrcPort: 65432] [DstPort: 54321] [Length: 0] [Seq: 5] [Ack: 4] ACK
Message [SrcPort: 65432] [DstPort: 54321] [Length: 0] [Seq: 6] [Ack: 5] ACK FIN
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 4] FIN
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 4] FIN
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 7] [Ack: 6] ACK
Fourth-Way Wavehand is successful!
Server is closed!
PS E:\刘修铭\南开大学\个人材料\课程\2023-2024 第1学期\计算机网络 张建忠, 徐敬东\Computer...
```

```
Please select the function you want to use:
1. Send File
2. Exit
> 1
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 4] FIN
!Time Out! -- First-Way Wavehand
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 4] FIN
!Time Out! -- First-Way Wavehand
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 4] FIN
Message [SrcPort: 65432] [DstPort: 54321] [Length: 0] [Seq: 5] [Ack: 4] ACK
Message [SrcPort: 65432] [DstPort: 54321] [Length: 0] [Seq: 6] [Ack: 5] ACK FIN
Message [SrcPort: 54321] [DstPort: 65432] [Length: 0] [Seq: 7] [Ack: 6] ACK
Fourth-Way Wavehand is successful!
Client is closed!
PS E:\刘修铭\南开大学\个人材料\课程\2023-2024 第1学期\计算机网络 张建忠, 徐敬东\Computer...
```

2. 局域网下联机测试

本次实验中，还借助局域网进行联机测试。仅以 3.jpg 为例进行展示。

传输完成，可以看到，累计用时 6104 ms，吞吐率为 1960.84 Byte/ms。

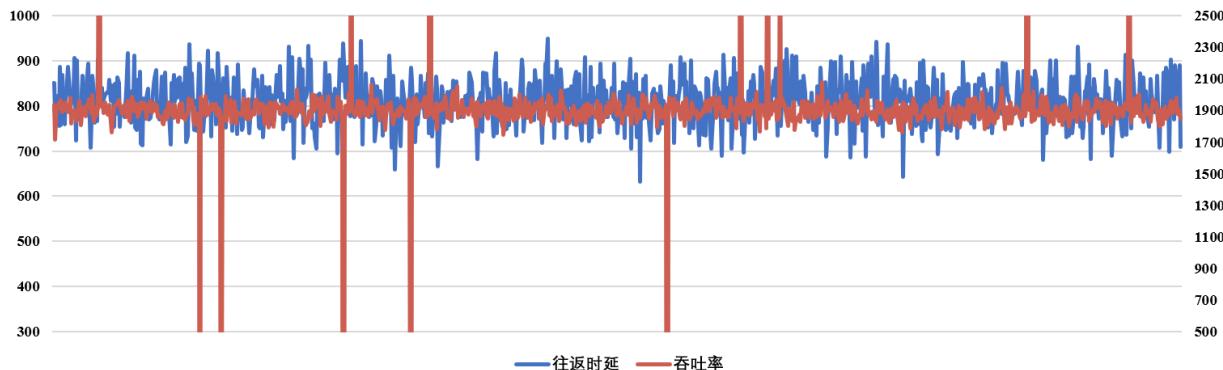
```
[ Window ] Base: 1469 Next: 1471 No_Verify_Num: 2 Residue: 18
Message [SrcPort: 65432] [DstPort: 54321] [Length: 0] [Seq: 0]
Receive Ack = 1469
[ Window ] Base: 1470 Next: 1471 No_Verify_Num: 1 Residue: 19
Message [SrcPort: 65432] [DstPort: 54321] [Length: 0] [Seq: 0]
Receive Ack = 1470
[ Window ] Base: 1471 Next: 1471 No_Verify_Num: 0 Residue: 20
Finish Sending File!
Send Time: 6104 ms
Send Speed: 1960.84 Byte/ms
```

右键查看文件属性，可以看到传输前后文件大小没有发生改变；打开文件，可以看到文件成功打开，说明传输无误。

经过测试，文件在丢包及时延条件下均能够正确传输，说明程序设计成功。

(二) 性能分析

在上面的传输测试中，添加日志输出，将传输时间、吞吐率、往返时延予以输出，进行数据清洗，然后使用 `excel` 绘制了实时吞吐率与实时往返时延的数据分析折线图。



可以直观看到，实时吞吐率逐渐提升并稳定在约 1900 Byte/ms ，而实时往返时延稳定在约 $800 \mu\text{s}$ ，偶尔会有波动。

以上数据仅对本次实验负责。

五、问题反思

(一) 协议格式错乱

在结构体定义代码处的 `#pragma pack(1)` 是一个编译器指令，用于告诉编译器以最小的字节对齐单位对结构体进行打包，该处即以 16 字节进行对齐。如果不加该指令，编译的时候可能会由于优化等过程改变原有的数据报文设计。

(二) 完善重传机制

本次实验中借鉴快速重传机制重新设计了超时重传机制，这样既可以保证数据传输的完整可靠，又实现了按照网络带宽实时确定重传时间，保证了传输效率。

同时，添加了类似于防睡眠机制，当长时间未收到数据时，将发送刺激性信息，刺激发送端继续发送数据。

(三) 全 0 数据

数据传输时，有时会出现数据全 0 的情况，即源端口号、目的端口号、数据段等均为 0 的情况，目前仍未曾知晓产生原因。为了避免其对运行的干扰，在接收端做了对应的修改。即，当接收到全 0 数据时，将向发送端发送等待的最大 Ack，刺激其进行快速重传操作。

```
1 if (tmp_msg.Seq == 0)
2 {
3     Message reply_msg;
4     reply_msg.Ack = Base_Seq - 1;
5     reply_msg.Set_ACK();
```

```
6     for (int j = 0; j < 3; j++)
7     {
8         if (Send(reply_msg))
9         {
10             SetConsoleTextAttribute(hConsole, 12);
11             cout << "Zero Message! Trying to Get New Message! Base_Seq = " <<
12             Base_Seq << endl;
13             reply_msg.Print_Message();
14             SetConsoleTextAttribute(hConsole, 7);
15         }
16     }
```