

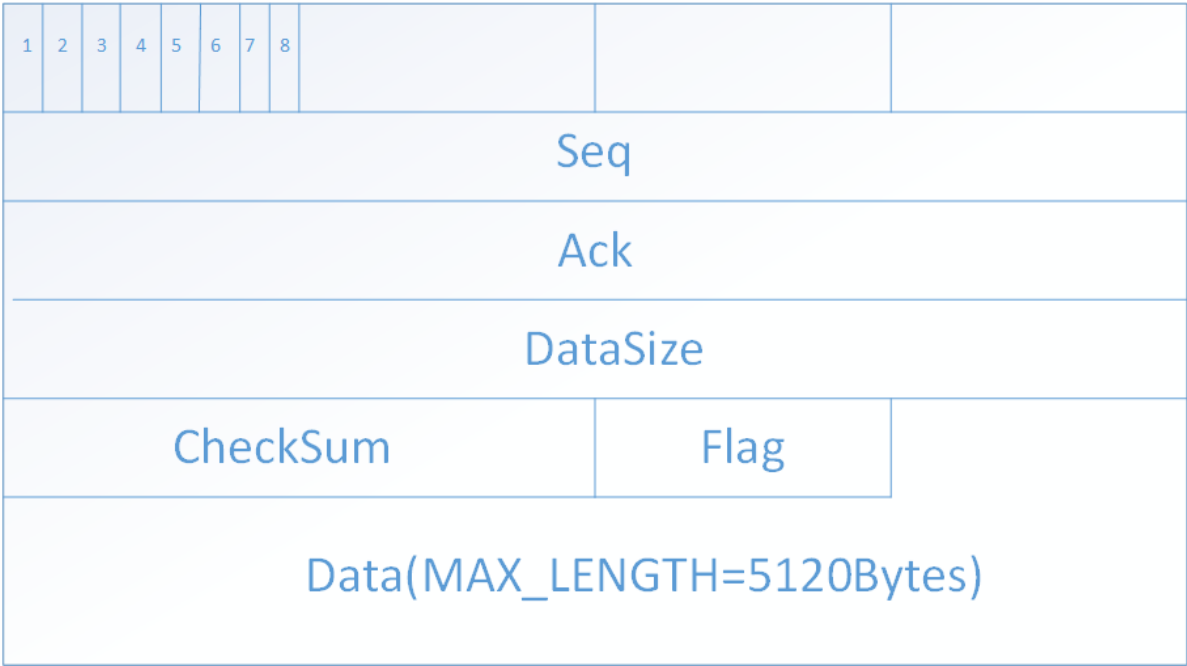
实验3-2：基于滑动窗口的流量控制机制

2013605

协议设计

本次实验采用 GBN 实现可靠数据传输

报文结构:



定义:

```
1  #define MAX_DATA_SIZE 10240
2  struct RDTHead
3  {
4      unsigned int seq;//序列号，发送端
5      unsigned int ack;//确认号码，发送端和接收端用来控制
6      unsigned short checkSum;//校验和 16位
7      unsigned int dataSize;          //标识发送的数据的长度,边界判断与校验和
8      char flag;                      //ACK, FIN, SYN, END
9
10     RDTHead()
11     {
12         this->seq=this->ack=0;
13         this->checkSum=this->dataSize=this->flag=0;
14     }
15 };
16
17 struct RDTPacket
18 {
19     /* data */
20     RDTHead head;
21     char data[MAX_DATA_SIZE];
22 };
```

如图所示，数据报文由报文头和数据部分组成。其都为定长。

报文头

Seq：表示发送的报文的序列号，接收端识别并确认。最大值为4294967295

Ack：与SEQ对应。表示接收端对收到的报文的序列号的确认。

CheckSum：校验和，可以确认报文在传输过程中是否受到损坏，用于差错检测。

Flag：用于握手和挥手过程的标识。主要用到了低四位标识不同的包。

DataSize：标识数据部分实际有效大小，用来确定传输文件的边界。

SYN 0x1：用于三次握手

ACK 0x2：用于三次握手和四次挥手

FIN 0x4：用于四次挥手

END 0x8：用于标识单个文件传输完毕。

报文数据

由于路由程序转包的最大包大小为 15000 字节，所以设计报文的数据部分大小不超过15000-`sizeof(header)` 字节即可。

流水线协议：Go-Back-N (GBN)

- 允许发送端发出N个未得到确认的分组
- 采用累积确认，只确认连续正确接收分组的最大序列号，可能接收到重复的ACK
- 发送端设置定时器，定时器超时，重传所有未确认的分组

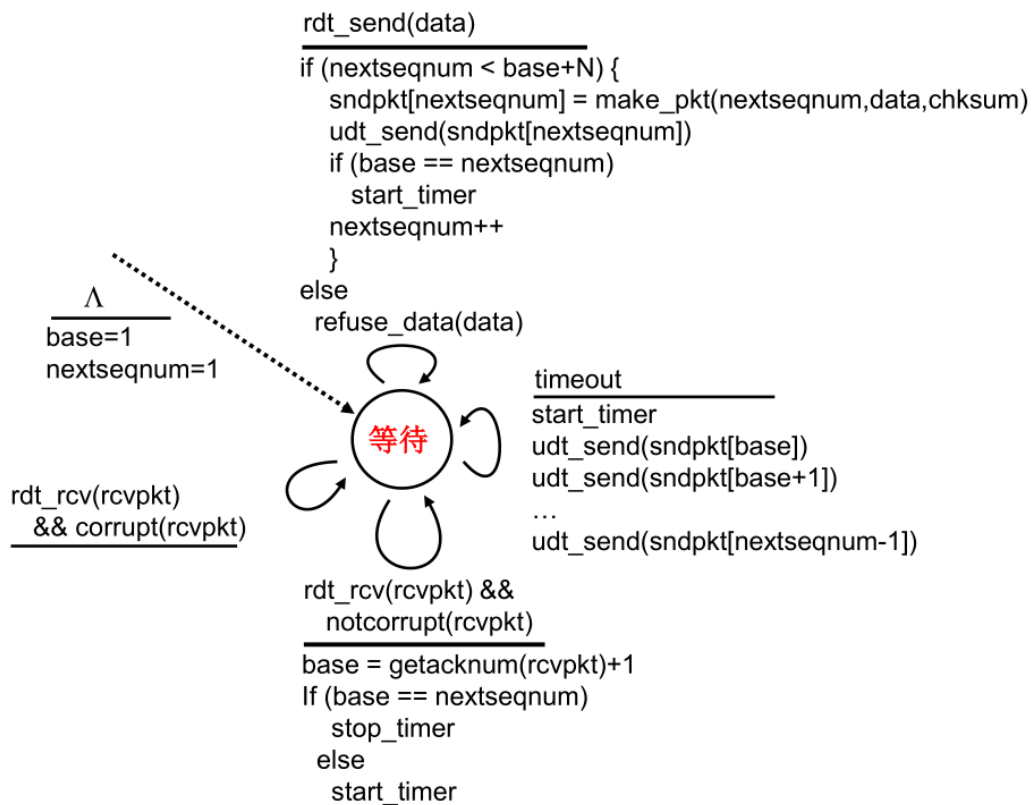
为了实现基于滑动窗口的流量控制机制，发送端需要维护了一个缓冲区，如图所示为发送端的发射缓冲区：



- **base**：窗口底部，代表**base-1**及以前部分消息已经全部传输完毕，且收到了对方返回的对应ACK，`msgsend[base]`是下面要接收对应ACK的消息
- **N**：窗口大小，本次实验取N=8
- **nextseqnum**：下一条要发送的消息

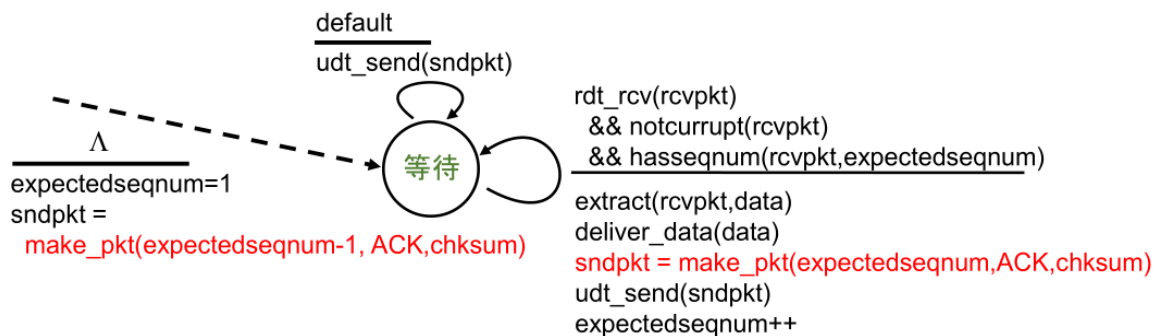
发送端

■ GBN发送端扩展FSM



接收端

- 只使用ACK，确认按序正确接收的最高序号分组
 - 会产生重复的ACK，需要保存希望接收的分组序号 (expectedseqnum)
- 失序分组（未按序到达）处理
 - 不缓存、丢弃
 - 重发ACK，确认按序正确接收的最高序号分组

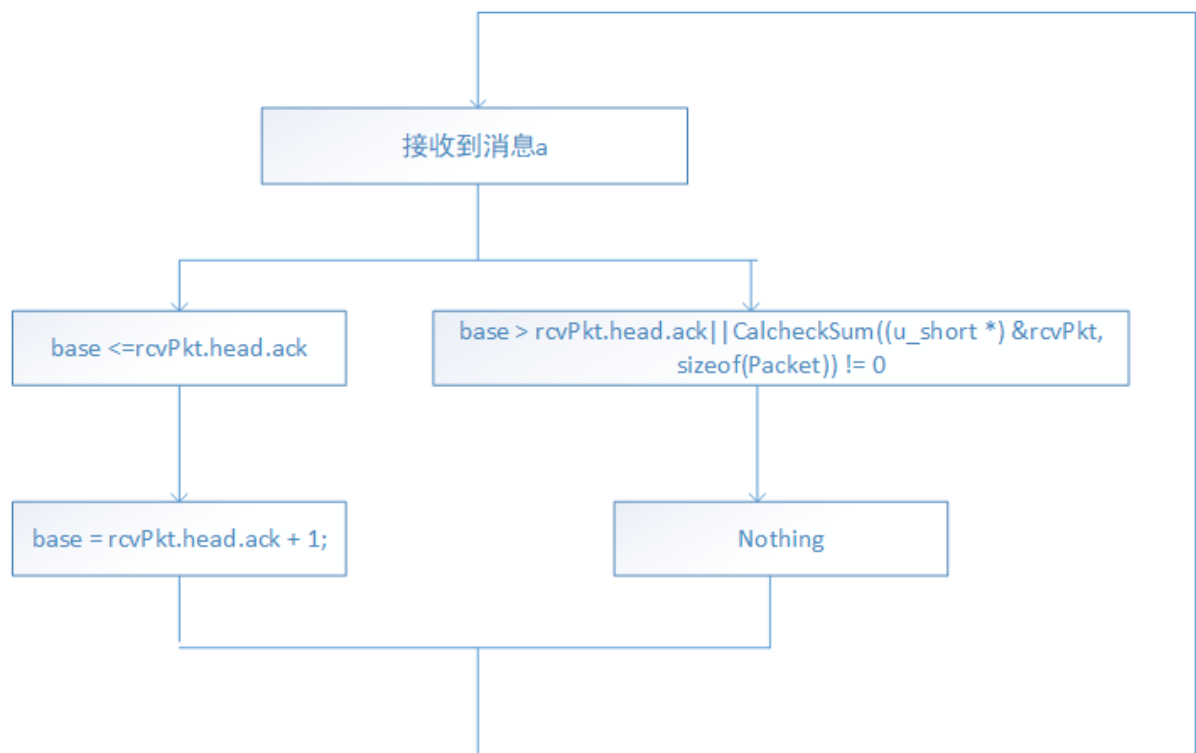


具体实现

发送端

为了实现消息的同时收发，提高效率，使用多线程编程。

client的接收线程



```

1  DWORD WINAPI clientRecv(LPVOID lparam)
2  {
3
4      Parameters *p = (Parameters *)lparam;
5      int packetNum = int(p->fileLen / MAX_DATA_SIZE);
6      int remain = p->fileLen % MAX_DATA_SIZE ? 1 : 0;
7      packetNum += remain;
8      char *dataBuffer = new char[MAX_DATA_SIZE], *pktBuffer = new
char[sizeof(Packet)];
9      Packet rcvPkt;
10     int addrLen = sizeof(p->serverAddr);
11     while (true)
12     {
13         if (recvfrom(p->clientSocket, pktBuffer, sizeof(Packet), 0,
(SOCKADDR *)&p->serverAddr, &addrLen) > 0)
14         {
15             memcpy(&rcvPkt, pktBuffer, sizeof(Packet));
16
17             if (base > rcvPkt.head.ack || Calchecksum((u_short *) &rcvPkt,
sizeof(Packet)) != 0) //忽略相同的ACK
18             {
19                 //cout << "收到错误的ACK:    " << rcvPkt.head.ack << "期望收到
的ACK:    " << base << endl;
20             }
21             else
22             {
23                 start = clock();
24                 //cout << "收到确认 " << rcvPkt.head.ack << endl;
25                 base = rcvPkt.head.ack + 1;
26             }
27
28
29             if (base == nextseqnum && base == packetNum + 1)
30             {
31                 Head endPacket;
32                 setEND(endPacket.flag);

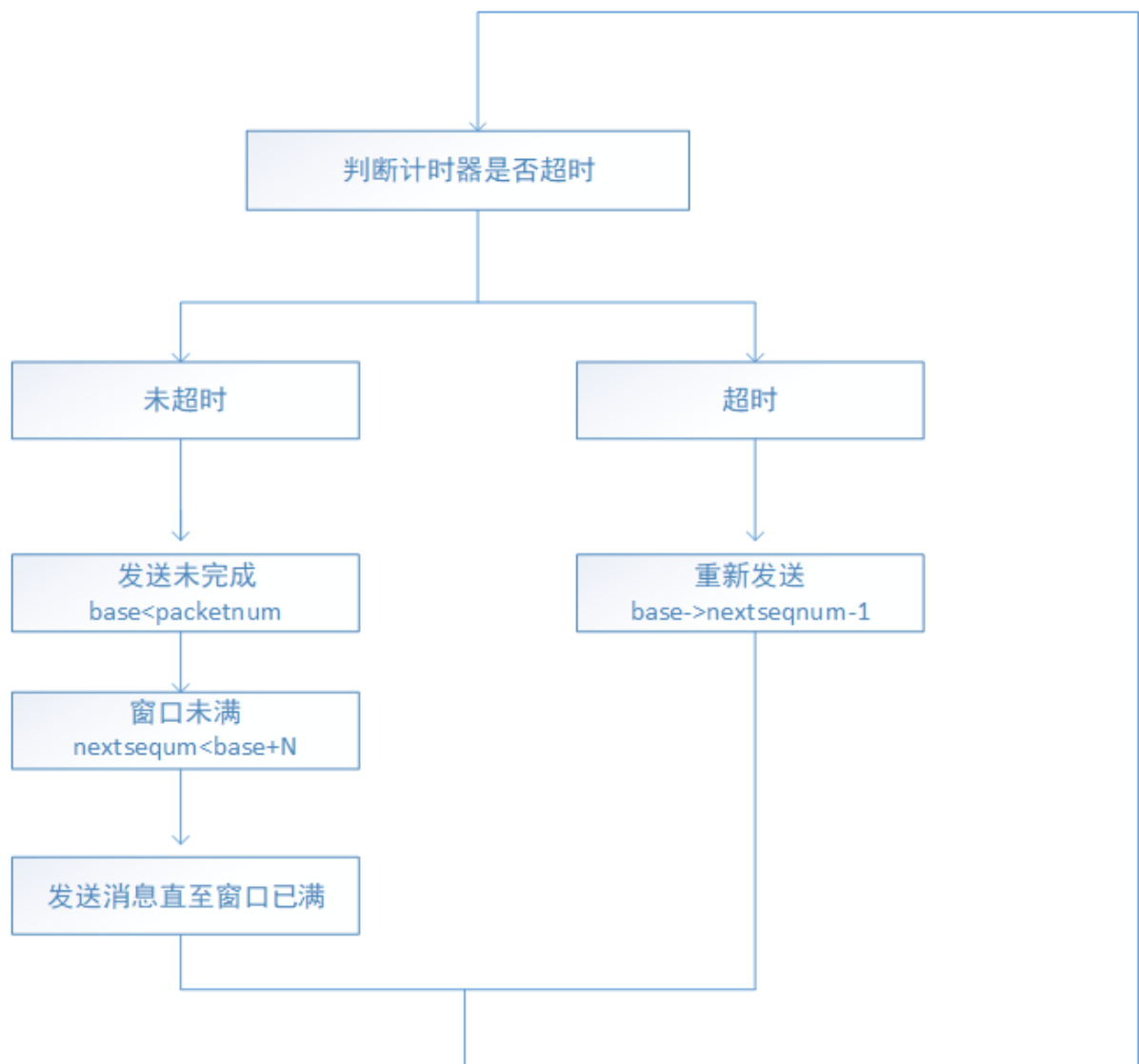
```

```

33         endPacket.checkSum = CalcheckSum((u_short *)&endPacket,
sizeof(endPacket));
34         memcpy(pktBuffer, &endPacket, sizeof(endPacket));
35         sendto(p->clientSocket, pktBuffer, sizeof(endPacket), 0,
(SOCKADDR *)&p->serverAddr, addrLen);
36
37         u_long imode = 1;
38         ioctlsocket(p->clientSocket, FIONBIO, &imode); //先进入非阻塞
模式
39         start = clock();
40         while (recvfrom(p->clientSocket, pktBuffer,
sizeof(endPacket), 0, (SOCKADDR *)&p->serverAddr, &addrLen) <= 0)
41         {
42             if (clock() - start >= MAX_TIMEOUT)
43             {
44                 memcpy(pktBuffer, &endPacket, sizeof(endPacket));
45                 sendto(p->clientSocket, pktBuffer,
sizeof(endPacket), 0, (SOCKADDR *)&p->serverAddr, addrLen);
46                 start = clock();
47             }
48         }
49         if (((Head *) (pktBuffer))->flag & ACK &&
CalcheckSum((u_short *)pktBuffer, sizeof(Head)) == 0)
50         {
51             cout << "文件传输完成" << endl;
52             isStop = true;
53             return 0;
54         }
55     }
56 }
57 }
58 }
59 }

```

client的发送线程



```

1  DWORD WINAPI clientSend(LPVOID lparam)
2  {
3      Parameters *p = (Parameters *)lparam;
4      int packetNum = int(p->fileLen / MAX_DATA_SIZE);
5      int remain = p->fileLen % MAX_DATA_SIZE ? 1 : 0;
6      packetNum += remain;
7      cout << "总共需要传输" << packetNum << "个数据包" << endl;
8      int dataSize;
9      int addrLen = sizeof(p->serverAddr);
10     char *dataBuffer = new char[MAX_DATA_SIZE], *pktBuffer = new
char[sizeof(Packet)];
11
12     start = clock();
13     while (true)
14     {
15         if (isStop == true)
16             return 0;
17         if (clock() - start > MAX_TIMEOUT)
18         {
19             cout << "超时! 重传" << base << "到" << nextseqnum - 1 << "号数据"<<
endl;
20             int count = nextseqnum - base;
21             int tmp = base;
22             for (int i = 0; i < count; i++)
23             {

```

```

24         dataSize = MAX_DATA_SIZE;
25         if ((tmp)*MAX_DATA_SIZE > p->fileLen) //
26         {
27             dataSize = p->fileLen - (tmp - 1) * MAX_DATA_SIZE;
28         }
29         memcpy(dataBuffer, p->fileBuffer + (tmp - 1) *
MAX_DATA_SIZE, dataSize);
30         Packet sendPkt = mkPacket(tmp, dataBuffer, dataSize);
31         memcpy(pktBuffer, &sendPkt, sizeof(Packet));
32         sendto(p->clientSocket, pktBuffer, sizeof(Packet), 0,
(SOCKADDR *)&p->serverAddr, addrLen);
33         //cout << tmp << "号数据包已经重新发送" << endl;
34         tmp++;
35     }
36     start = clock();
37 }
38 else
39 {
40     while (nextseqnum <= packetNum)
41     {
42         if (nextseqnum < base + windowSize)
43         {
44             dataSize = MAX_DATA_SIZE;
45             if ((nextseqnum)*MAX_DATA_SIZE > p->fileLen) //
46             {
47                 dataSize = p->fileLen - (nextseqnum - 1) *
MAX_DATA_SIZE;
48             }
49             memcpy(dataBuffer, p->fileBuffer + (nextseqnum - 1) *
MAX_DATA_SIZE, dataSize);
50             Packet sendPkt = mkPacket(nextseqnum, dataBuffer,
dataSize);
51             memcpy(pktBuffer, &sendPkt, sizeof(Packet));
52             sendto(p->clientSocket, pktBuffer, sizeof(Packet), 0,
(SOCKADDR *)&p->serverAddr, addrLen);
53             if (base == nextseqnum) {
54                 start = clock();
55             }
56             nextseqnum++;
57             cout << "base: " << base << "nextseqnum: " <<
nextseqnum << "end: " << base + windowSize << endl;
58             //start = clock();
59         }
60         else
61             break;
62     }
63 }
64 }
65 }

```

接收端

- 按顺序接收对方发来的消息seqnum (收到的消息序号) ==expectedSeq, 且校验和正确, 返回对应的ACK (ackseq=expectedSeq)

- 如果发生消息失序, 即`expectedSeq!=recvseqnum`, 或校验和错误, 则丢弃消息, 返回
`ACK=recvseqnum-1ACK`

```
1  bool recv(char *fileBuffer, SOCKET &socket, SOCKADDR_IN &addr, unsigned long
    &filelen) {
2      filelen = 0;
3      int addrLen = sizeof(addr);
4      u_int expectedSeq = 1;
5      int dataLen;
6
7      char *pktBuffer = new char[sizeof(Packet)];
8      Packet recvPkt, sendPkt= mkPacket(-1);
9
10     while (true) {
11         memset(pktBuffer, 0, sizeof(Packet));
12         recvfrom(socket, pktBuffer, sizeof(Packet), 0, (SOCKADDR *) &addr,
            &addrLen);
13         memcpy(&recvPkt, pktBuffer, sizeof(Packet));
14         if(recvPkt.head.seq==expectedSeq && Calchecksum((u_short*)&recvPkt,
            sizeof(Packet))==0){
15             //收到正确的ack;
16             sendPkt = mkPacket(expectedSeq);
17             memcpy(pktBuffer, &sendPkt, sizeof(Packet));
18             sendto(socket, pktBuffer, sizeof(Packet), 0, (SOCKADDR *) &addr,
                addrLen);
19
20             dataLen = recvPkt.head.dataSize;
21             memcpy(fileBuffer + filelen, recvPkt.data, dataLen);
22             filelen += dataLen;
23             cout<<"send ack"<<expectedSeq<<endl;
24             expectedSeq++;
25
26             continue;
27         }
28         if (isEND(recvPkt.head.flag) && Calchecksum((u_short*)&recvPkt,
            sizeof(Head))==0) {
29             cout << "传输完毕" << endl;
30             Head endPacket;
31             setACK(endPacket.flag);
32             endPacket.checkSum = Calchecksum((u_short *) &endPacket,
                sizeof(Head));
33             memcpy(pktBuffer, &endPacket, sizeof(Head));
34             sendto(socket, pktBuffer, sizeof(Head), 0, (SOCKADDR *) &addr,
                addrLen);
35             return true;
36         }
37
38
39         cout<<"wait head:"<<expectedSeq<<endl;
40         cout<<"recv head:"<<recvPkt.head.seq<<endl;
41         memcpy(pktBuffer, &sendPkt, sizeof(Packet));
42         sendto(socket, pktBuffer, sizeof(Packet), 0, (SOCKADDR *) &addr,
            addrLen);
43     }
44 }
```


实验展示

路由器Router设置：

Router

路由器IP: 127 . 0 . 0 . 1

服务器IP: 127 . 0 . 0 . 1

端口: 8887

服务器端口: 8888

丢包率: 5 %

延时: 0 ms

确定

修改

日志

Router Ready!
Misscount :20 .
Delay :0 ms .

三次握手建立连接

PS D:\cpp_vscode\commpter_network\lab3-2改> .\client.exe

第一次握手,进入 SYN_SEND状态
第二次握手成功
第三次握手进入 TIME_WAIT状态
第三次握手进入 Established状态
请输入需要传输的文件名
1.jpg
开始进行传输,文件大小为: 1857353
总共需要传输363个数据包

PS D:\cpp_vscode\commpter_network\lab3-2改> .\server.exe

第一次握手成功
第二次握手进入 SYN_RECV状态
第三次握手进入 Established状态
开始接受文件
send ack1
send ack2
send ack3

文件传输与重传演示

base: 1nextseqnum: 2end: 9	第二次握手进入SYN_RECV状态
base: 1nextseqnum: 3end: 9	第三次握手进入Established状态
base: 1nextseqnum: 4end: 9	开始接受文件
base: 2nextseqnum: 5end: 10	send ack1
base: 2nextseqnum: 6end: 10	send ack2
base: 2nextseqnum: 7end: 10	send ack3
base: 2nextseqnum: 8end: 10	send ack4
base: 2nextseqnum: 9end: 10	send ack5
base: 3nextseqnum: 10end: 11	send ack6
base: 3nextseqnum: 11end: 11	send ack7
base: 4nextseqnum: 12end: 12	send ack8
base: 5nextseqnum: 13end: 13	send ack9
base: 6nextseqnum: 14end: 14	send ack10
base: 7nextseqnum: 15end: 15	send ack11
base: 8nextseqnum: 16end: 16	send ack12
base: 9nextseqnum: 17end: 17	send ack13
base: 10nextseqnum: 18end: 18	send ack14
base: 11nextseqnum: 19end: 19	send ack15
base: 12nextseqnum: 20end: 20	send ack16
base: 13nextseqnum: 21end: 21	send ack17
base: 14nextseqnum: 22end: 22	wait head:18
base: 15nextseqnum: 23end: 23	recv head:19
base: 17nextseqnum: 24end: 25	wait head:18
base: 17nextseqnum: 25end: 25	recv head:20
base: 18nextseqnum: 26end: 26	wait head:18
超时！重传18到25号数据	recv head:21
base: 19nextseqnum: 27end: 27	wait head:18
base: 20nextseqnum: 28end: 28	recv head:22
base: 22nextseqnum: 29end: 30	wait head:18
base: 22nextseqnum: 30end: 30	recv head:23
base: 23nextseqnum: 31end: 31	wait head:18
base: 25nextseqnum: 32end: 33	recv head:24
base: 26nextseqnum: 33end: 34	wait head:18
base: 26nextseqnum: 34end: 34	recv head:25

传输完成于四次挥手断开连接

超时！重传354到361号数据	传输完毕
base: 355nextseqnum: 363end: 363	第一次挥手客户端请求断开
base: 356nextseqnum: 364end: 364	第二次挥手进入CLOSE-WAIT状态
文件传输完成	第三次挥手进入LAST-ACK状态
客户端发起第一次挥手进入FIN-WAIT-1状态	第四次挥手进入closed状态
客户端进入FIN-WAIT-2状态	请按任意键继续... █
第三次挥手	
第四次挥手进入TIME-WAIT状态	
第四次挥手超时重传	
第四次挥手超时重传	
第四次挥手进入closed状态	
成功关闭连接	
请按任意键继续... █	

结果对比

