南开大学

Nankai University

南　　开　　大　　学

网 络 空 间 安 全 学 院

计算机网络实验报告

编程作业 3 基于 UDP 服务设计可靠传输协议并编程实现[1]

于文明

年级：2020 级

专业：信息安全

指导教师：徐敬东，张建忠

2022 年 11 月 19 日

摘要

关键字：socket UDP rdt3.0

目录

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、确认重传等。流量控制采用停等机制，完成给定测试文件的传输。

作业要求

- 数据报套接字：UDP；
- 建立连接：实现类似 TCP 的握手、挥手功能；
- 差错检测：计算校验和；
- 确认重传：rdt2.0、rdt2.1、rdt2.2、rdt3.0 等，亦可自行设计协议；
- 单向传输：发送端、接收端；
- 有必要日志输出。

二、 协议设计 [1]

本实验基于 UDP 传输层协议，为了实现可靠传输，在应用层应用了类似 rdt3.0 协议的实现，并设计了相应的自动状态机，同时也设计了相应的报文格式。

(一) 报文格式

报文格式设计参照 UDP 协议报文设计和 TCP 报文设计, 具体如下图所示:

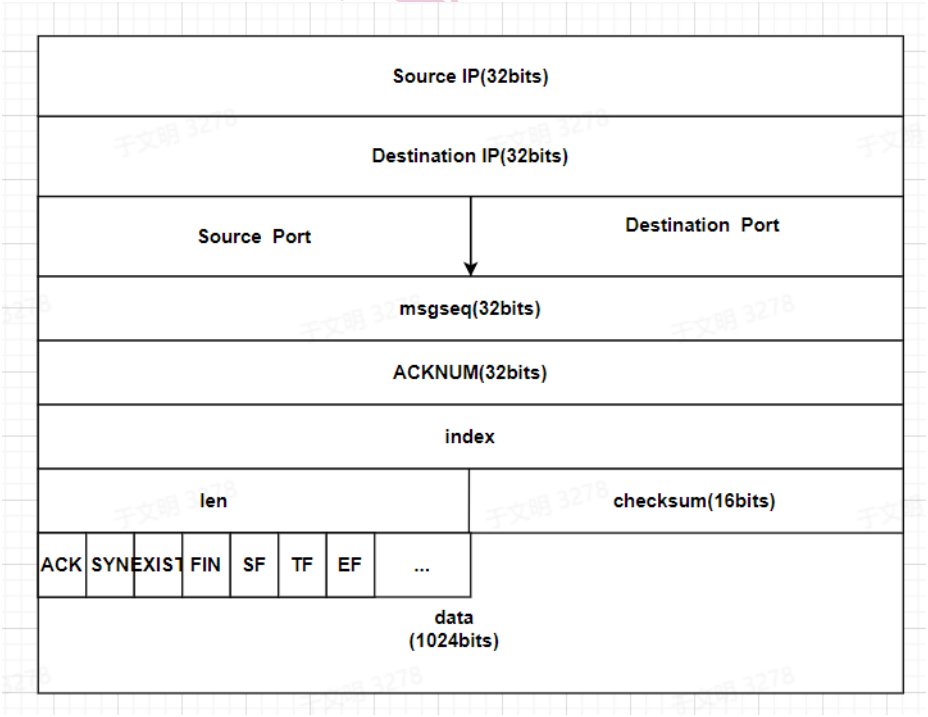

图 1: 报文格式

上图中 TCP 报文中每个字段的含义如下:

发送端 ip 地址和接收端 ip 地址 (32 bits)

源端口和目的端口 (16bits)

消息序列号字段 msgseq, 每次发送数据包，msgseq 增加 1

确认好字段 acknum, 它表示接收方期望收到发送方下一个报文段的第一个字节数据的编号。其值是接收计算机即将接收到的下一个序列号，也就是下一个接收到的字节的序列号加 1。

标志位字段 (flag，16bits)

ACK：表示前面的确认号字段是否有效。只有当 ACK=1 时，前面的确认号字段才有效.

SYN：在建立连接时使用，用来同步序号。当 SYN=1,ACK=0 时，表示这是一个请求建立连接的字段；当 SYN=1,ACK=1 时，表示对方

EXIST 表示数据包有效, 由于 socket 采用非阻塞的方式，需要 EXIST 标志位判断是否数据包有效

FIN：标记是否断开连接

SF: 表示开始发送文件

TF：表示传输文件数据包

EF：表示文件发送完毕

(二) 流程图

在设计中，为了发送的效率，建立连接只需要实现两次握手，同样断开连接也只需要实现两次挥手，数据包采用停等机制发送，即每次 client 端发送一条消息后，需要等待对方回复 == 对应 == 的 ACK，才能发送下一条消息。如果数据包错误或者超时，都将会采用错误重传。

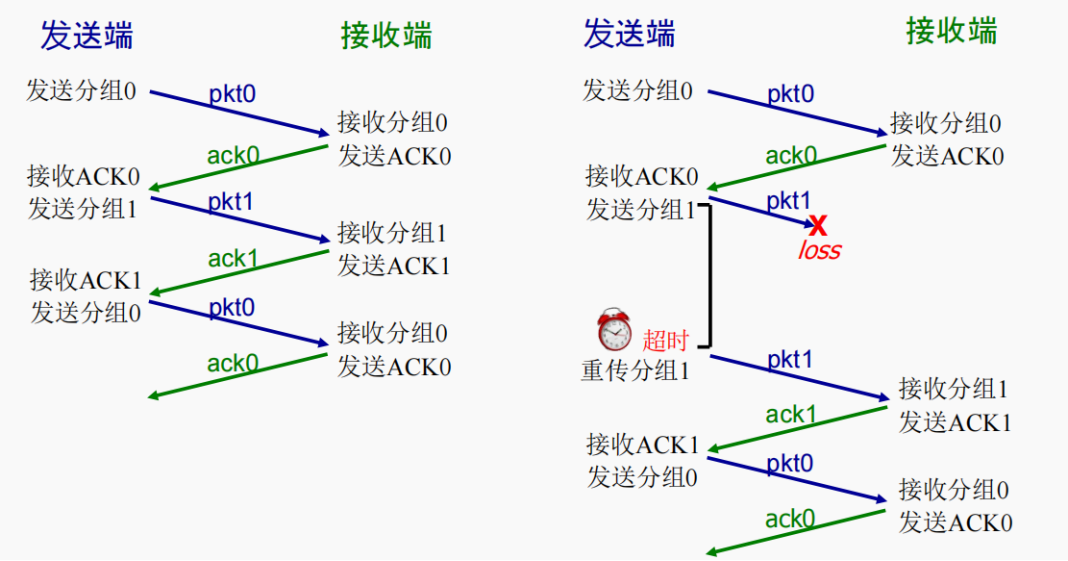

图 2: 流程图

同 TCP 类似，第一次握手发送端发送 SYN 数据包，接收端在接收到 SYN 包后返回一个 ACK 确认包，第二次握手完成，连接建立成功；两次挥手的程序逻辑与连接建立类似，client 端发送含有 FIN 的消息，server 端收到后回复对应的 ACK，连接断开

同时因为需要实现测试文件的单向传输，下面将简单介绍发送端和接收端的交互流程：

Client 端

- 初始状态 0

- 建立连接，发送 SYN 数据包，接收端在接收到 SYN 包后返回一个 ACK 确认包，连接建立成功
- **发送文件**
- 首先，发送 SF 数据包，提示将要发送文件
- 然后，将读入的文件数据放入数据包的 data 字段，TF 置位，发送数据包
- 文件发送完毕之后，发送 EF 数据包，提示文件发送完毕
- **断开连接**：发送 FIN 数据包，收到对应的 ACK 之后，断开连接，回到初始状态 0

Server 端

- 建立连接，收到 SYN 数据包，回复对应的 ACK 数据包，连接建立成功
- **接收文件**
- 接收到 SF 包，恢复相应的 ACK, 进入接收文件函数
- 接收 TF 数据包，将对应的数据相应的文件 (outfile)
- 接收到 EF 数据包，文件接收完成，回到主流程
- **断开连接**，接收 FIN 数据包，回复对应的 ACK, 连接断开

具体的交互流程如下

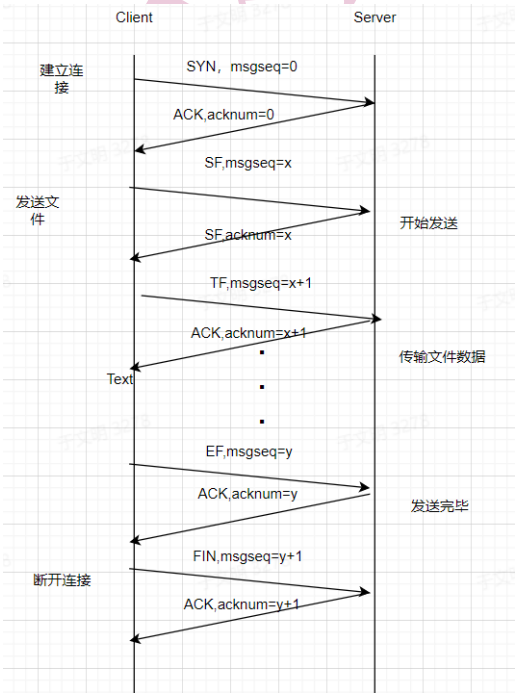

图 3: CS 交互流程图

(三) 状态转换 (FSM)

本实验根据课程上讲述的 rdt3.0 的有限状态自动机，实现了自己的 FSM，这里以发送端为例说明:

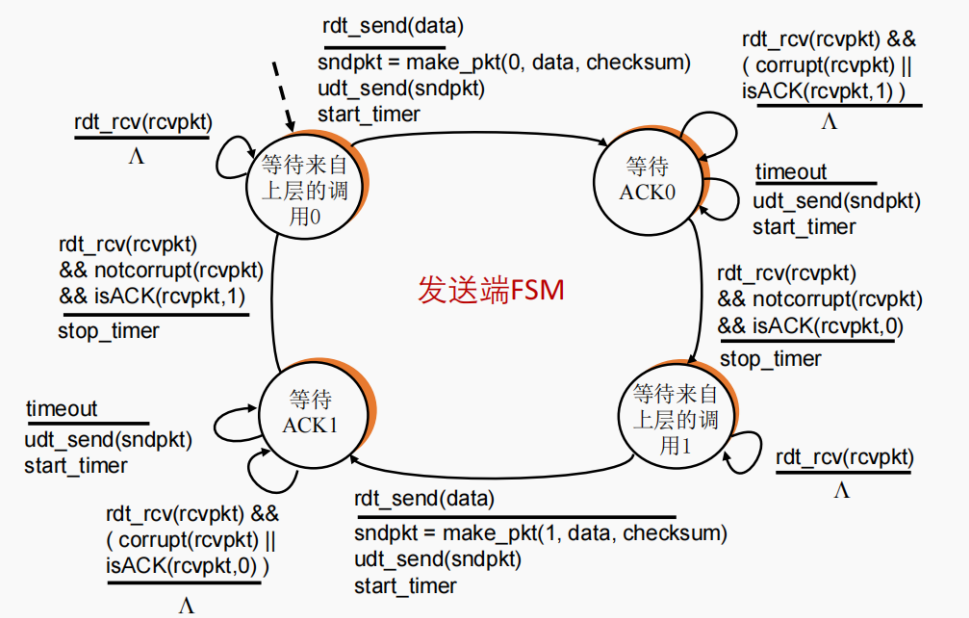

图 4: 发送端状态机

上图是 rdt3.0 的标准状态机，本实验根据需求做出了适当的简化，因为发送端和接收端的状态机类似，这里以接收端为例，具体如下:

- 首先是初始状态 0，表示没有建立连接时所处的状态
- 建立连接后，进入状态 1，如果因为数据包传输错误或者超时，支持重传机制
- 发送文件 SF 包后，进入状态 2，等待发送文件数据
- 进入文件传输 TF 包，进入状态 3，传输文件数据
- 文件传输完毕，发送 EF 包，退出状态 3，进入状态 2
- 断开连接，退回初始状态 0

三、 核心代码

(一) 数据包结构体的实现

数据报的结构体 (struct packet) 和相关的函数实现都放在了 rdt3.h 头文件中，其中 packet 的成员按照数据报文设计的格式进行定义，为了便于 flag 标志位的相关操作，预定义了相关标志位的宏定义

packet 成员变量

```
1 #define ACK 0x0
2 #define SYN 0x1
```

```
#define EXIST 0x2
#define FIN 0x3
#define SF 0x4
#define TF 0x5
#define EF 0x6
#pragma pack(1)//一下内容按1byte对齐，如果没有的话会以4bytes对齐
struct packet {
        //定义发送数据包格式
        DWORD SrcIP, DstIP;//发送端ip和接收端ip
        u_short SrcPort, DstPort;//发送端端口号和接收端端口号
        int msgseq;//消息序列号
        int acknum;//acknowledgement number
        unsigned short flag;//标志位
        int index;//用于描述文件大小
        int filelength;
        unsigned short len;//数据长度
        unsigned short checksum;//校验和
        char data[1024];//传输的数据
```

结构体的函数 **1. 初始化函数**

值得注意的是这里的内容需要按照 1byte 对齐，否则可能会造成数据对齐错误 pragma pack(1)//内容需要按 1byte 对齐

packet 初始化

```
        packet() {
                this->SrcIP = NULL;
                this->DstIP = NULL;
                this->SrcPort = 6666;
                this->DstPort = 5678;
                this->flag = 0;
                this->msgseq = 0;
                this->acknum = 0;
                this->index = 0;
                this->filelength = 0;
                this->len = 0;
                this->checksum = 0;
        };
        ...
#pragma pack()//恢复
```

2. 标志位置位函数 (这里以 set_ack 为例，其他置位函数类似)

设置标志位

```
void packet::set_ack(packet b)
{
        this->flag=this->flag | (1u<<ACK);
        this->acknum = b.msgseq;
}
```

3. 设置校验和

在设置校验和的函数中，以 16 位作为分组长度，定义 u_short 指针 tmp 求取对应位置的数据值如果数据和大于 0xffff, 对其进行移位和按位与 0xffff 操作, 最后将求取的 sum 取反赋值给 checksum

设置校验和

```
void packet::setchecksum(int packetlen)
{//计算校验和
        //将数据以1byte进行处理
        u_long sum = 0;
        int count = (packetlen + 1) / 2;
        u_short* tmp = (u_short*)this;// 这里需要设置unsigned类型, 否则会有正
            负号
        while (count--) {
                sum += *(tmp++);
        }
        sum = (sum >> 16) + (sum&0xffff);
        sum += (sum >> 16);
        this->checksum = ~sum;
        //cout << "sum" << sum << "checksum"<< this->checksum << endl;
        return;
}
```

检查校验和的过程同设置校验和类似，求取的 sum 如果等于 0xffff，校验和正确, 发送 ack 确认包；否则，发送 ack=0 的包，进行重传.

(二) 消息处理流程

1. socket 编程

同之前的 TCP 编程类似，分别初始化 Client 和 Server 的 socket 和 addr。需要注意的是，这里的 socket 需要设置成 UDP 协议，因为 UDP 的的非阻塞特性，需要设置成非阻塞状态。

socket 编程

```
        if(!WSAinit())
                printf("[WSA,SUCCESS]  WSAinit Success\n");
        SOCKET sockClient = socket(AF_INET, SOCK_DGRAM, 0);
        //设置套接字非阻塞
        u_long imode = 1;
        ioctlsocket(sockClient, FIONBIO, &imode);
        //string ADDRSER;
        //cout << "请输入服务器的地址:";
        //getline(cin, ADDRSER);
        DstIP = inet_addr("127.0.0.1");
        HostIP = inet_addr("127.0.0.1");
        SOCKADDR_IN addrSer;
        addrSer.sin_family = AF_INET;
        addrSer.sin_port = htons(SerPORT);
        //addrSer.sin_addr.S_un.S_addr = inet_addr(ADDRSER.c_str());
```

```
        addrSer.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
        SOCKADDR_IN addrCli;
        addrCli.sin_family = AF_INET;
        addrCli.sin_port = htons(CliPORT);
        //addrSer.sin_addr.S_un.S_addr = inet_addr(ADDRSER.c_str());
        addrCli.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
        if (bind(sockClient, (SOCKADDR*)&addrCli, sizeof(SOCKADDR)) == -1)
                cout << "client bind error\n";
```

2. 建立连接和断开

建立连接的过程已经在上述过程叙述了，只需要两次握手，Client 端发送包含 syn；同时，断开连接同建立连接的操作类似，同样只需要两次挥手.

建立连接 (客户端)

建立连接

```
int buildConnectCli(SOCKET& sock, SOCKADDR_IN& addr) {
        /*
        客户端发起连接
        */
        packet a, b;
        a.set_syn();
        //设置发送序列号
        a.msgseq = sendseq++;
        //cout << "设置序列号.\n";
        if (stopWaitsend(sock,addr,a, b))
                return 1;//停等发送机制，发送消息a,发生超时事件时重传，如果收到了对应的ack则b返回1,超过最大重传次数返回0
        else
                return 0;
};
```

断开连接 (客户端)

建立连接

```
int disConnect(SOCKET& sock, SOCKADDR_IN& addr) {
//2次挥手
        packet a, b;
        a.set_fin();
        if (stopWaitsend(sock, addr, a, b))
                return 1;
        else
                return 0;
}
```

3. 消息发送和接收

为了方便结构体的发送和接收，封装了 sendpak 和 recvpak 函数，具体的就是设置 exist 和设置校验和

发送数据包

```
void sendpak(SOCKET &sock,SOCKADDR_IN &addr,packet& a)
{
        a.set_exist();
        a.setchecksum(sizeof(a));//设置校验和
        sendto(sock, (char*)&a, sizeof(packet), 0, (sockaddr*)&addr, sizeof(
            addr));
}
```

4. 状态转换实现 (FSM)

发送数据包

```
int FSM(SOCKET& sock, SOCKADDR_IN& addr, packet& b) {
        if (b.get_syn())
        {
                if (buildConnectSer(sock, addr, b))
                {
                        status = 1;//设置状态
                        cout << "[CONNECT,SUCCESS]  建立连接成功" << endl;
                }
                else
                        cout << "[CONNECT,ERROR]  连接建立失败\n";
        }
        else if (b.get_sf())
        {
                if (status)
                {
                        cout << "[FILE,RECV]  开始接收文件...\n";
                        recvfile(sock, addr, b);
                }
                else
                        cout << "[ERROR]  请先建立连接\n";
        }
        else if (b.get_fin())
        {
                if (disConnectSer(sock, addr,b))
                {
                        cout << "[DISCONNECT,SUCCESS]  断开连接" << endl;
                        status = 0;
                        sendseq = 0;
                }
        }
        return 1;
```

```
}
```

接收数据包

```
接收数据包时需要先将传入的数据包对应数据段data清空，防止受到之前消息的影响
void recvpak(SOCKET &sock,SOCKADDR_IN &addr,packet& a)
{
        memset(a.data, 0, sizeof(a.data));//清空数据，防止收到之前消息的影响
        int addrlen = sizeof(addr);
        recvfrom(sock, (char*)&a, sizeof(packet), 0, (sockaddr*)&addr,&
            addrlen);
}
```

(三) 异常处理机制

在发送和接收数据包的过程中，有以下几种异常情况:

- ACK 丢包，处理方式是 client 端超时重传，如果一段时间后没有收到 ACK 确认包，就重传相应的数据包
- 数据包报文信息丢失或则被篡改，这里采用校验和差错检测的方式，如果 server 端检查校验和不正确，将会发送一个 ack=0 的包，client 端收到 ack=0 的包之后重传数据包.
- server 端发送了 ACK，但是 client 端没有收到，这时仍然需要重传
- client 端收到两条 ack 且 acknum 相同，取较早收到的 ack 就好.

1. 差错检测

使用校验和的方式检测数据包是否丢失，校验和分组长度为 16bits, 具体实现在 packet 结构体内部，这里通过 server 端的一处引用来说明相关操作

检查校验和

```
    int check = a.checkchecksum(sizeof(a));//检测校验和
        if (check)//检测成功
        {
                cout << "[RECV]  收到数据包(checked,checksum="<<a.checksum<<"
                    )" << endl;
                b.set_ack(a);//回复对于收到消息a的ack消息b
                b.msgseq = sendseq++;
                cout << "[SEND,ACK]  发送确认包(msgseq=" << b.msgseq << "
                    acknum=" << b.acknum << ")\n";
                sendpak(sock,addr,b);
                memset((char*)&b, 0, sizeof(packet));//防止干扰下一次消息接收
                return 1;
        }
        else
        {
                packet b;
                sendpak(sock, addr, b);
```

```
                continue;
        }
```

在上述 server 端的代码中，如果校验和检查无误的话，输出相应的日志，同时设置 ack，返回 ACK 包，清空缓冲区数据，防止干扰下一次消息接收.

2. 确认重传

在前面的叙述中已经提到，涉及到确认重传相关内容的主要有以下三种情况

- server 端检测校验和出错，发送 ack=0 数据包
- server 端发送的 ack 丢包
- 超时未收到 ACK 确认包

预先已经在 rdt3.h 中定义了最大重传次数 (10) 和最长等待时间（1s）

确认重传

```
bool stopWaitsend(SOCKET& sock, SOCKADDR_IN& addr, packet& a, packet b)
{
        sendpak(sock, addr,a);
        clock_t start = clock();//计时
        int flag = 0;//
        while (1) {
                recvpak(sock, addr, b);
                if (b.get_ack() && b.acknum == a.msgseq)
                {//收到确认
                        return 1;
                }
                clock_t end = clock();
                if (flag == MAX_SEND_TIMES)//重发10次失败
                        return 0;
                if ((end - start) / CLOCKS_PER_SEC >= MAX_WAIT_TIME)
                {
                        flag++;
                        start = clock();//重置计时器
                        cout << "[RESEND]  第" << flag << "次重传" << endl;
                        sendpak(sock, addr, a);//重传
                }
        }
        return 0;
}
```

(四) 停等机制实现

在 server 端，recvfrom 被设置成了非阻塞状态，采用停等机制处理接收到的数据包, 封装了 stopWaitrecv() 函数. 在函数内部，主要通过检测校验和判断是否丢包，如果数据包正常，就发回确认包；否则发回 ack=0 的数据包，启动重传机制。

停等接收

```
bool stopWaitrecv(SOCKET& sock, SOCKADDR_IN& addr, packet& a, packet b)
{
        int flag = 0;
        while (1)
        {
                recvpak(sock,addr,a);//收到对方发来的消息a
                if (a.get_exist())//因为将recv函数设成了非阻塞, 所以需要检测收到的消息是否为空
                {
                        int check = a.checkchecksum(sizeof(a));//检测校验和
                        if (check)//检测成功
                        {
                                cout << "[RECV]  收到数据包(checked,checksum="<<a.checksum<<")" << endl;
                                b.set_ack(a);//回复对于收到消息a的ack消息b
                                b.msgseq = sendseq++;
                                cout << "[SEND,ACK]  发送确认包(msgseq=" << b.msgseq << "acknum=" << b.acknum << ")\n";
                                sendpak(sock,addr,b);
                                memset((char*)&b, 0, sizeof(packet));//防止干扰下一次消息接收
                                return 1;
                        }
                        else
                        {
                                packet b;
                                sendpak(sock, addr, b);
                                continue;
                        }
                }
        }
        cout << "[RECV,ERROR]  接收失败" << endl;
        return 0;
}
```

(五) 文件传输

1. 文件发送

同上面描述的一样, 首先 Client 端会发送包含文件名 name 的数据包, 同时 flag 的 SF 位置位, 表示开始发送文件, Server 端在接收到 SF 包后, 检验校验和无误后发送 ACK 确认包, 同时创建一个名字为 name 的文件

2. 中间数据包传输

读入文件，因为 data 的长度为 1024bit，因此如果文件过大需要多次发送数据包。相同的时，数据包都将 TF 位置位，表示传输文件数据，同时传输文件数据包序列号 index 和数据包长度 len

server 端在接收到 TF 包后，会将接收到的相应的 data 数据写入对应的文件中，并返回 ACK 包

3. 文件发送完毕

client 会发送一个 EF 置位的数据包，标识文件发送完毕；server 在检测无误后返回 ACK

4. 具体代码

同文件有关的函数及预定义都放在了 FILE.h 头文件实现 **Client 端 (包含性能评估指标的计算)**

检查校验和

```
int sendfile(SOCKET& sock, SOCKADDR_IN& addr, char* name)
{
        char content[1024];
        int length = 0;
        int index = 0;
        //发送文件名
        clock_t start = clock();
        struct packet a, b;
        //a.index = index;
        //a.filelength = length;
        int t = strlen(name);

        memset(a.data, 0, sizeof(a.data));
        memcpy(a.data, name, t);

        a.set_sf();
        a.msgseq = sendseq++;
        //发送ef包;
        if (!stopWaitsend(sock, addr, a, b))
        {
                cout << "[FILE,TRANS,ERROR] 文件传输失败\n";
                return 0;
        }
        cout << "[FILE,READ]  读入文件...\n";
        index = 0;
        length = 0;
        ifstream in(name, ifstream::binary);//以二进制方式读入文件
        if (!in)
        {
                cout << "[FILE,ERROR]  文件无效\n";
                return 0;
```

```
32          }
33          char line = in.get();;
34          while (in)//读入失败或完成退出循环
35          {//读取文件内容存在content数组
36
37                  content[length % 1024] = line;//每行1024个字节，对应data成员
                        长度为1024
38                  length++;
39                  if (length % 1024 == 0)
40                  {
41                          packet tmp;
42                          tmp.set_tf();
43                          tmp.index = index;
44                          tmp.filelength = length;
45                          memset(tmp.data, 0, sizeof(tmp.data));
46                          memcpy(tmp.data, content, sizeof(content));
47                          memset(content, 0, sizeof(content));
48                          tmp.msgseq = sendseq++;
49                          //cout << tmp.filelength << endl;
50                          if (!stopWaitsend(sock, addr, tmp, b))
51                          {
52                                  cout << "[FILE,SEND,ERROR]  文件发送失败\n";
53                                  return 0;
54                          }
55                          index++;
56                          length = 0;
57                          //cout << index << endl;
58                  }
59                  line = in.get();
60          }
61          if (length != 0) {
62                  packet tmp;
63                  tmp.set_tf();
64                  tmp.index = index;
65                  tmp.filelength = length;
66                  memset(tmp.data, 0, sizeof(tmp.data));
67                  memcpy(tmp.data, content, sizeof(content));
68                  memset(content, 0, sizeof(content));
69                  tmp.msgseq = sendseq++;
70                  if (!stopWaitsend(sock, addr, tmp, b))
71                  {
72                          cout << "[FILE,SEND,ERROR]  文件发送失败\n";
73                          return 0;
74                  }
75          }
76          in.close();
77          //发送文件结束确认包
78          packet p1, p2;
```

```
        p1.set_ef();
        cout << "[SEND,EF]发送ef包\n";
        if (!stopWaitsend(sock, addr, p1, p2))
        {
                cout << "[ERROR] 出错0\n";
                return 0;
        }
        //吞吐率计算
        clock_t end = clock();
        double dtime = (double)(end - start) / CLOCKS_PER_SEC;
        cout << "[TOTAL TIME]  总用时:" << dtime << endl;
        cout << "[RATE]  吞吐率:" << (double)(index + 1) * sizeof(packet) * 8
             / dtime / 1024 / 1024 << "Mbps" << endl;
        return 1;
}
```

server 端 (涉及到文件接收和写入)

接收文件

```
int recvfile(SOCKET sock, SOCKADDR_IN& addr, packet& a)
{
        char content[1024];
        packet p;
        p.set_ack(a);
        a.msgseq = sendseq++;
        //cout << "发送pak...\n";
        sendpak(sock,addr,p);
        //获取文件信息
        int index = a.index;
        int length = a.filelength;
        char name[100];
        memset(name, 0, 100);
        memcpy(name, a.data, 100);
        while (1) {
                packet tmp,b;
                stopWaitrecv(sock, addr, tmp,b);
                if (tmp.get_tf())
                {
                        //cout << "开始传输...\n";
                        memset(content, 0, sizeof(content));
                        //cout << "size" << sizeof(tmp.data) << "len" << tmp.
                            filelength << endl;
                        memcpy(content, tmp.data, tmp.filelength);
                        //cout << "before outfile\n";
                        outfile(name, content, tmp.filelength);
                }
                if (tmp.get_ef())
                {
                        cout << "收到ef包" << endl;
```

```
                        break;
                }
        }
        cout << name << "文件接收成功\n";
        return 1;
}
```

写入文件

```
void outfile(char* name, char content[1024], int length)
{
        //cout << "this is outfile\n";
        ofstream out;
        out.open(name,ios::binary|ios::app | ios::out);
        for (int i = 0; i < length; i++)
        {
                out << content[i];
        }
        out.close();
}
```

(六) 日志输出

具体的日志输出按照 [标志]+ 信息的格式输出，比如 [SEND] 表示发送数据包 [RECV] 表示接收数据包，[WSA] 表示 wsa 初始化，[FILE] 表示文件相关操作,[SUCCESS] 表示操作成功，[ERROR] 表示操作出错。允许多个状态同时存在，比如 [FILE,SEND,SUCCESS] 表示成功发送文件相关数据包, 具体结果将会在实验结果中展示。

四、 实验结果

(一) 建立连接

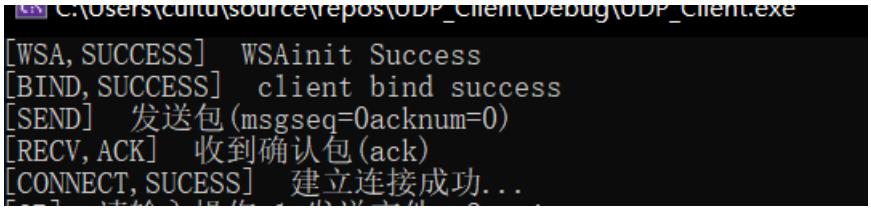

图 5: 建立连接客户端

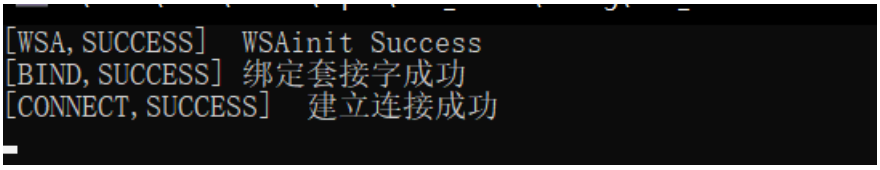

图 6: 建立连接服务端

(二) 断开连接

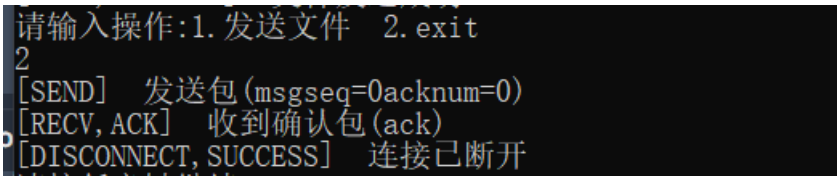

图 7: 断开连接客户端

图 8: 断开连接服务端

(三) 文件发送

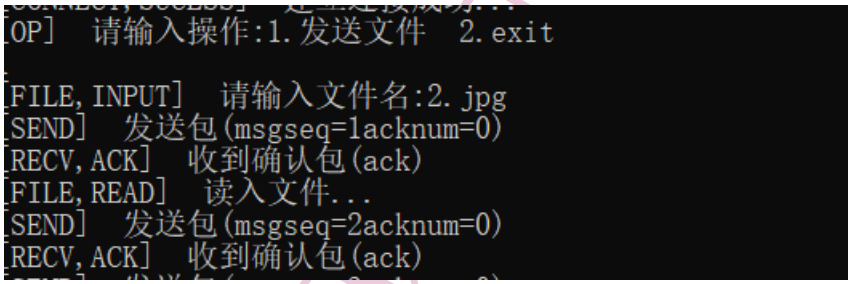

图 9: 文件发送 SF 包 (Client)

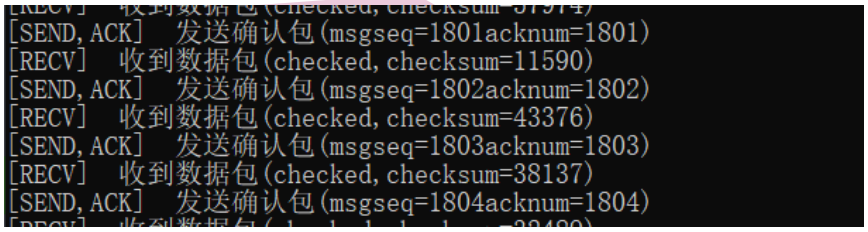

图 10: 文件发送 TF 包 (Server)

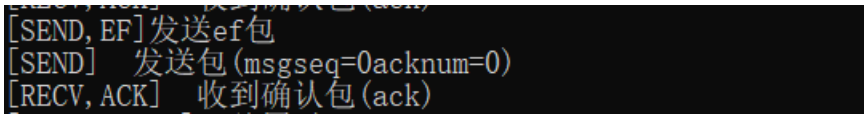

图 11: 文件发送完毕 EF 包 (Client 端)

(四) 日志输出

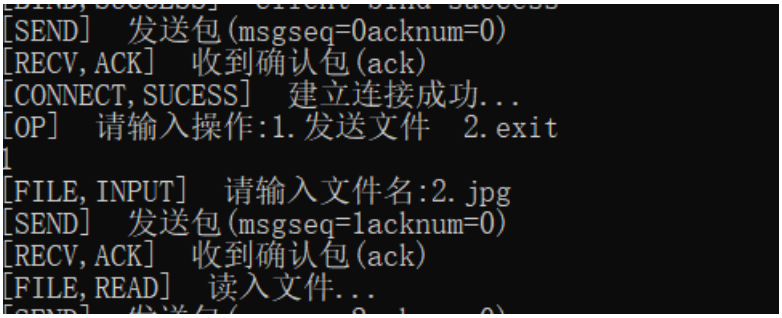

图 12: 日志 (Client)

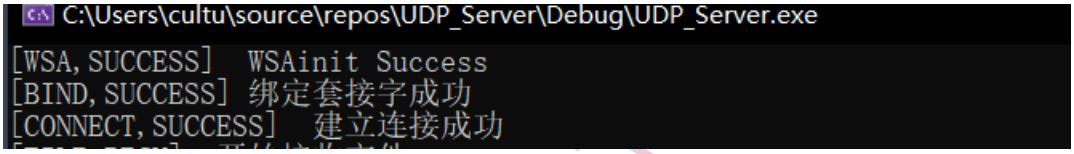

图 13: 日志 (Server)

(五) 性能评估

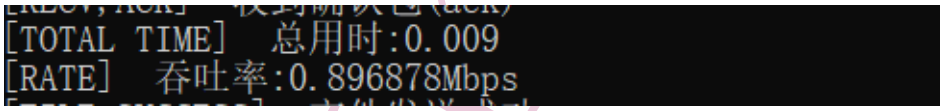

图 14: 测试样例 1(.txt,17bit)

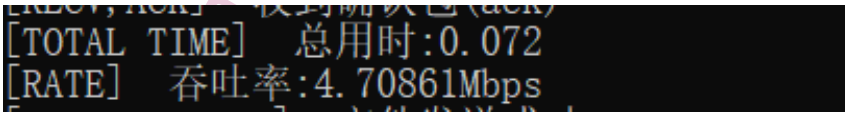

图 15: 测试样例 2(.bmp,41.1KB)

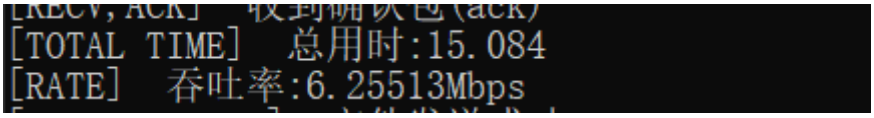

图 16: 测试样例 3(.jpg,11.4MB)

(六) 附录

本实验相关的资源 (源文件,测试文件,实验结果截图等) 都已上传至 github(https://github.com/Metetor/network

参考文献

[1] 基思.W. 罗斯詹姆斯.E. 库罗斯. **计算机网络*:* 自顶向下方法**. 机械工业出版社, 2009.