

计算机网络实验3.1-基于UDP服务设计可靠传输协议(停等版本)

实验要求

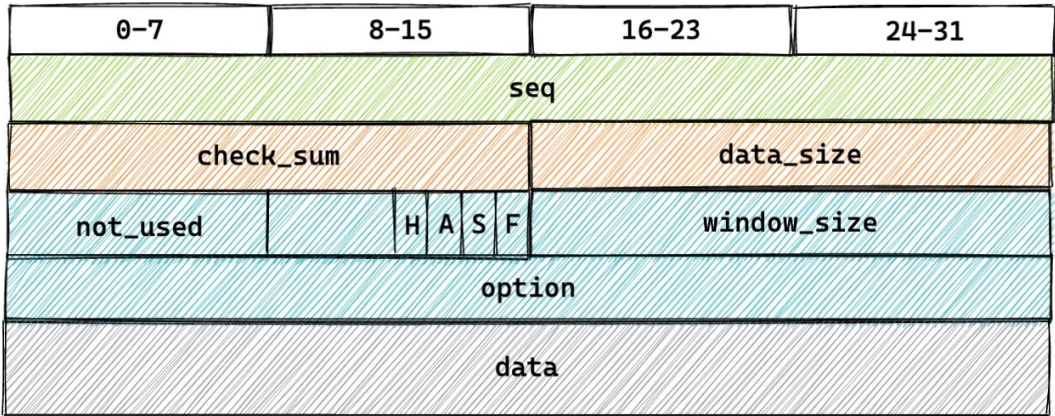
利用数据报套接字在用户空间实现面向连接的可靠数据传输，功能包括：建立连接、差错检测、确认重传等。流量控制采用停等机制，完成给定测试文件的传输。

程序流程展示

协议设计

在本次实验中采用基于rdt3.0的协议设计。

报文结构



如图所示，报文头长度共 128Bits。下面介绍报文结构设计的思路。

首先，注意到我们的实验只需要从客户端到服务器单向传输数据，因此我们事实上整个实验只需要一个序列号字段即可满足需求。对于发送端对应 TCP 中的 seq,接收端对应 TCP 中的 ack。

下面是十六位校验和以及数据报字段长度，与 TCP 相同。

目前使用 u_short 来存放 flag。其字段含义如下：

F:FIN

S:SYN

A:ACK

H:FILE_HEAD

FILE_HEAD 用于指示接收端此报文包含文件信息的字段。

window_size 本次实验还没有用到。

option 为可选字段，在本次实验中暂时用于存放文件名。

data 的最大长度可以调节，本次实验定义为1024字节。

此部分定义代码段如下；

```
#define MAX_SIZE 1024
#define DATA 0x0
#define FIN 0x1
#define SYN 0x2
#define ACK 0x4
#define ACK_SYN 0x6
#define ACK_FIN 0x5
#define FILE_HEAD 0x8
// datagram format:
#pragma pack(1)
struct packet_head {
    u_int seq;
    u_short check_sum;
    u_short data_size;
    u_short flag;
    u_short window_size;
    u_int option;

    packet_head() {
        seq = 0;
        check_sum = 0;
        data_size = 0;
    }
};
```

```

    flag = 0;
    window_size = 0;
    option = 0;
}
};

struct packet {
    packet_head head;
    char data[MAX_SIZE]{};

    packet() {
        packet_head();
        memset(data, 0, MAX_SIZE);
    }
};

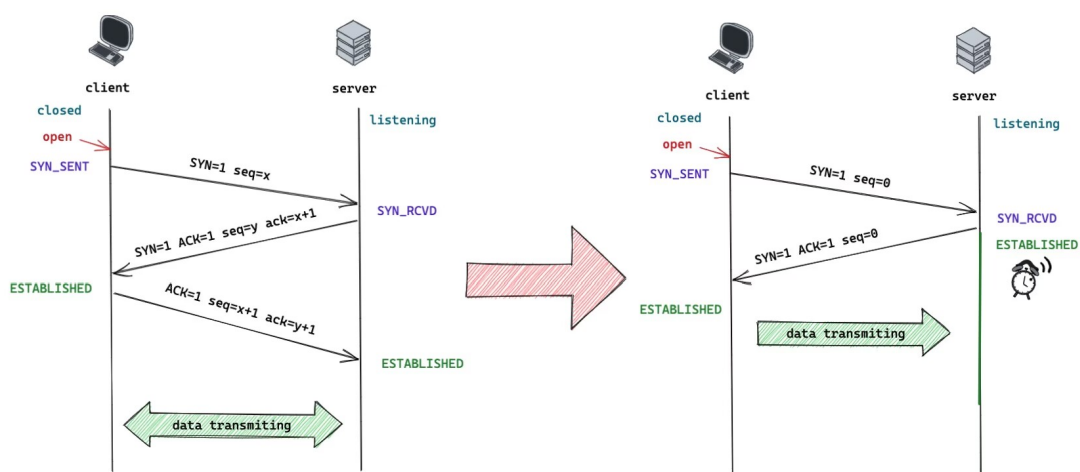
#pragma pack(1)

```

`#pragma pack(1)` 用于指示结构体内容按1Byte对齐，以保证报文大小是我们期望的紧凑形式。

建连和断连

依然是注意到单向传输的特点，对握手和挥手的过程也进行了优化：



左边是TCP三次握手的过程，右侧是我为本次实验设计的握手过程。

TCP第三次握手的目的是“server”需要知道“client”能够收到他的应答。这在server向client发送数据时是有必要的，而本次实验只要发送端知道接收端能发能收，就可以放心的向其发送文件，握手成功。

接收端在收到发送端的握手信息后就可以准备好接受文件了。此时接收端预料的应当是发送端发送文件信息。但是这时候如果发送端断线了，接收端显然不能干等着，否则在真实情景下完全可以发起类似SYN洪泛攻击的行为。因此我们需要设置定时器，如果在这段时间发送端没有任何信息发来，这时应当释放资源并退出。在此次实验中这个最大时间设置的是1min。

同时，虽然此次实验假设接收端向发送端发送数据丢包率为0，但是在实验中仍旧考虑了这种情况：如果接收端的ACK丢了，会发生什么情况？因此在接收端准备接受文件时，仍旧检查收到的是否是握手信息。如果是握手信息，那么重置上述提到的定时器，仍旧停留在等待文件信息的状态。

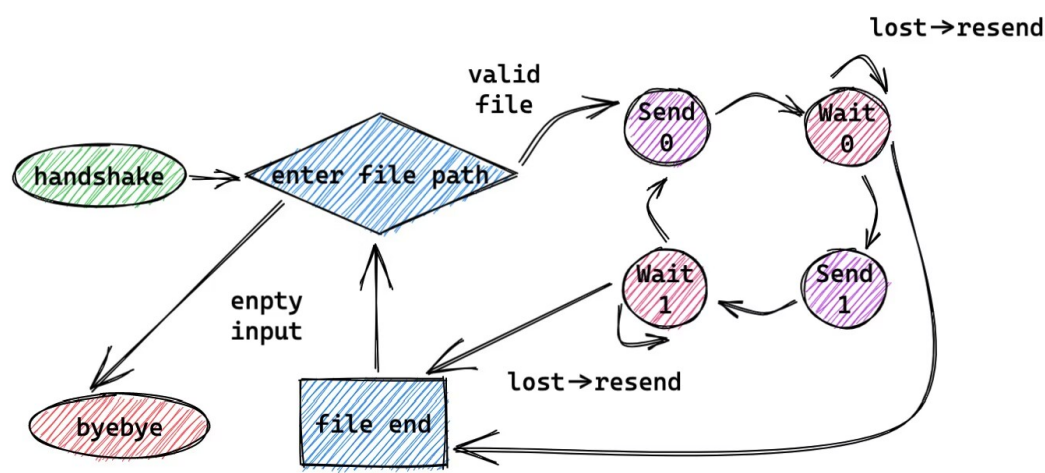
断连的过程和上述分析类似，也是只需要两次即可。

另外，不需要文件结束位的原因是因为接受者在得到文件信息的时候就知道文件大小，从而知道有几个数据包，什么时候结束。

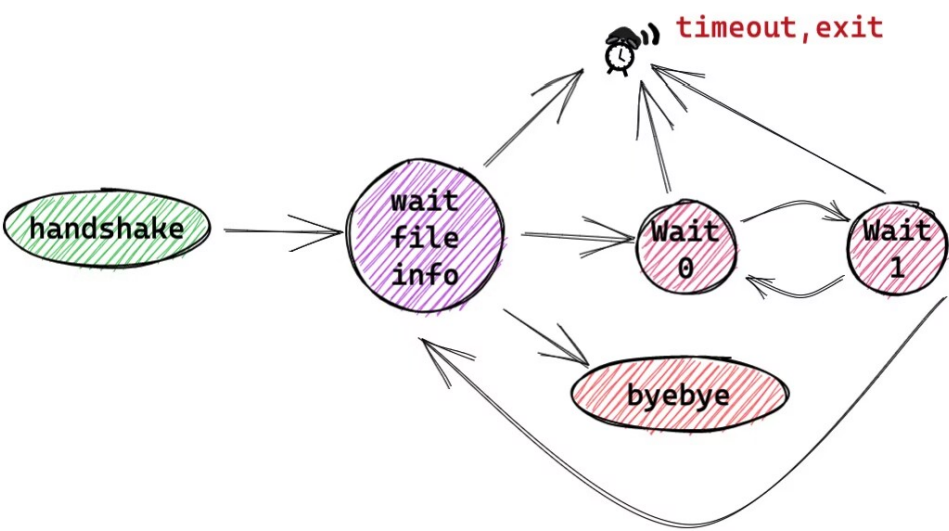
流程设计

程序支持一次建连发送多个文件。

服务器



客户端



发送文件数据时遵循rdt3.o的整个过程，也即，在这次实验中序列号暂时只用到0和1。

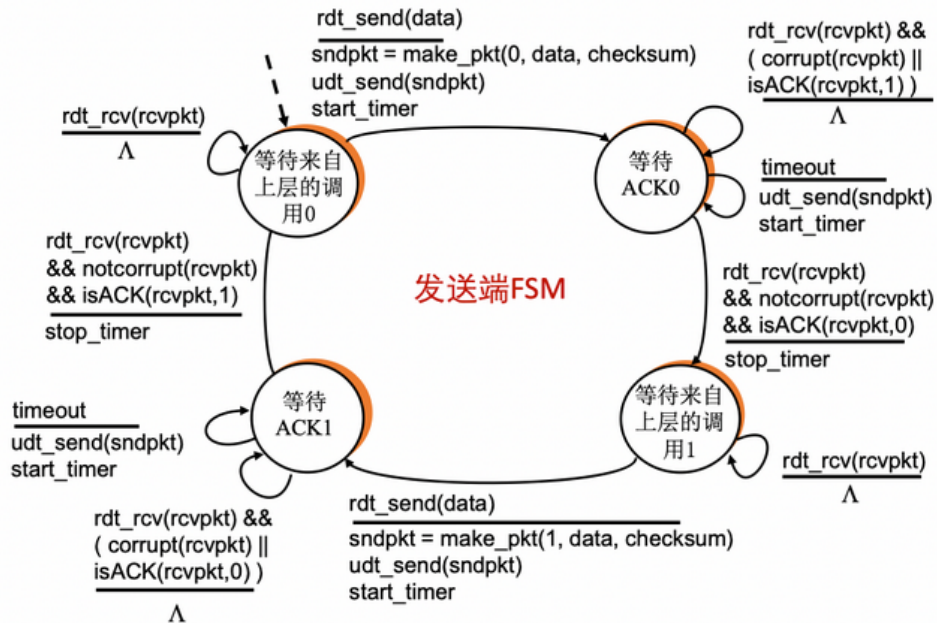
程序代码解释

文件发送过程

发送端

首先对照发送端的状态机进行分析：

■ rdt3.0: 发送端状态机



程序中的函数名与状态机中名称基本一致，思路也非常清晰。主要的变动为把 `waitACK0` 和 `waitACK1` 合并到了函数中，而不是作为单独的状态出现。这样做的原因是由于握手和挥手阶段的等待过程和文件传输过程中完全一致，通过相同发代码能够将过程统一起来。

```

while(pkt_no < pkt_total)
{
    pkt_data_size = min(MAX_SIZE, file_len - pkt_no * MAX_SIZE);
    switch(stage)
    {
        case SEND0:
        {
            packet sndpkt = make_pkt(DATA, 0,
pkt_data_size, file_data + pkt_no * MAX_SIZE);
            udt_send(sndpkt);
            if (!wait_ACK0(sndpkt)) {
                print_message("Failed when sending packet
number " + to_string(pkt_no), ERR);
                return 1;
            }
            print_message("Sent packet number " +
to_string(pkt_no) + " with seq 0", DEBUG);
        }
    }
}
    
```

```

        pkt_no++;
        stage = SEND1;
        break;
    }
    case SEND1:
    {
        packet sndpkt = make_pkt(DATA, 1,
pkt_data_size, file_data + pkt_no * MAX_SIZE);
        udt_send(sndpkt);
        if (!wait_ACK1(sndpkt)) {
            print_message("Failed when sending packet
number " + to_string(pkt_no), ERR);
            break;
        }
        print_message("Sent packet number " +
to_string(pkt_no)+" with seq 1", DEBUG);
        pkt_no++;
        stage=SEND0;
        break;
    }
    default:
        break;
}
}

```

下面是waitACK相关函数的实现，以waitACK0为例：

首先由于需要处理超时事件，发送端和接收端所有的·socket都是非阻塞状态的。对于发送端while条件中的rdt_rcv是非阻塞的，以便在循环内判断超时进行消息重发。当没有收到消息时返回0，收到消息时返回1。循环内的重发若超过一定次数（MAX_RESEND_TIMES,其值为10），便可认为接收端由于意外断连，不再向其发送消息，程序退出。

```

bool wait_ACK0(packet sndpkt) {
    int resend_times = 0;
    //start a timer
    clock_t start = clock();

```

```

packet rcvpkt;
//non-blocking receive here
while (!rdt_rcv(rcvpkt) || isACK(rcvpkt, 1) || corrupt(rcvpkt))
{
    if (timeout(start)) {
        udt_send(sndpkt);
        start = clock();
        if (resend_times > MAX_RESEND_TIMES) {
            print_message("Resend times exceed the limit,
there must be something wrong with the network", ERR);
            return false;
        } else {
            print_message("Resend packet with seq 0",
WARNING);
            resend_times++;
        }
    }
    if (isACK(rcvpkt, 1)) {
        print_message("Received ACK1, discard it", DEBUG);
    }
}
return true;
}

```

`rdt_rcv(rcvpkt)` 实现如下:

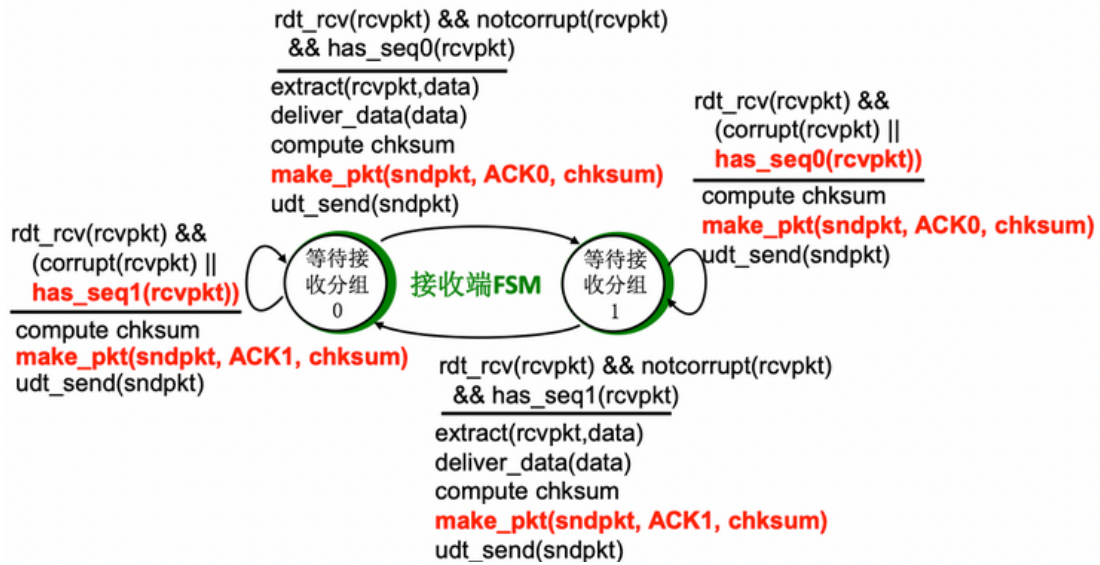
```

bool rdt_rcv(packet &packet1) {
    int len = sizeof(addr_server);
    int ret = recvfrom(socket_sender, (char *) &packet1,
PACKET_SIZE, 0, (SOCKADDR *) &addr_server, &len);
    if (ret == SOCKET_ERROR) {
        return false;
    }
    return ret != 0;
}

```


接收端

接收端状态机



与发送端有所不同，这里的`rdt_rcv(rcvpkt)`是阻塞的，内含一个非阻塞的`recvfrom`进行循环接收。若超时（一分钟）仍未收到消息，认为发送端可能意外退出，跳出接收循环，并随后判断文件是否完整接收，以作退出之前的保存和清理工作。这样设计的原因也是在握手和挥手时不依赖其他条件的需要同样的操作，能够较好的统一起来。

其他部分与状态机中一致。在循环内部需要判断接收到的文件是否已经完全接受，若接受完毕保存文件并退出，准备继续接受下一个文件。

```
while (rdt_rcv(rcvpkt)) {
    if (not_corrupt(rcvpkt)) {
        if (has_seq0(rcvpkt)) {
            if (stage == WAIT0) {
                print_message("Received packet " +
to_string(pkt_no) + ", with seq 0", DEBUG);
                pkt_data_size = rcvpkt.head.data_size;
                memcpy(file_buffer + received_file_len,
rcvpkt.data, pkt_data_size);
                received_file_len += pkt_data_size;
                packet sndpkt = make_pkt(ACK, 0);
                udt_send(sndpkt);
                pkt_no++;
            }
        }
    }
}
```

```

        stage = WAIT1;
    } else {
        print_message("Received a packet with seq 0, but
we are waiting for seq 1", WARNING);
        continue;
    }
} else if (has_seq1(rcvpkt)) {
    if (stage == WAIT1) {
        print_message("Received packet " +
to_string(pkt_no) + ", with seq 1", DEBUG);
        pkt_data_size = rcvpkt.head.data_size;
        memcpy(file_buffer + received_file_len,
rcvpkt.data, pkt_data_size);
        received_file_len += pkt_data_size;
        packet sndpkt = make_pkt(ACK, 1);
        udt_send(sndpkt);
        pkt_no++;
        stage = WAIT0;
    } else {
        print_message("Received a packet with seq 1, but
we are waiting for seq 0", WARNING);
        continue;
    }
}
} else {
    print_message("Received a corrupt packet", DEBUG);
    continue;
}
if (received_file_len == file_size) {
    print_message("Received file successfully", SUC);
    print_message("Time used: " + to_string(clock() -
single_file_start) + "ms", INFO);
    //write the file to disk
    string file_path = get_file_path(file_name);
    ofstream file(file_path, ios::binary);
    if (file.is_open()) {
        file.write(file_buffer, file_size);
    }
}

```

```

        file.close();
        print_message("File saved to " + file_path, SUC);
        new_file_received = true;
    } else {
        print_message("Failed to open file " + file_path,
ERR);
    }
    break;
}
}

```

`rdt_rcv(rcvpkt)` 的实现如下:

```

bool rdt_rcv(packet &packet1) {
    clock_t wait_file_start = clock();
    //non-blocking receive here
    int ret = recvfrom(socket_receiver, (char *) &packet1,
PACKET_SIZE, 0, (SOCKADDR *) &addr_server, &addr_len);
    while (ret == SOCKET_ERROR || ret == 0) {
        //no packet received
        if (wait_file_timeout(wait_file_start)) {
            print_message("Timeout, no packet received", ERR);
            return false;
        }
        ret = recvfrom(socket_receiver, (char *) &packet1,
PACKET_SIZE, 0, (SOCKADDR *) &addr_server, &addr_len);
    }
    return true;
}

```

握手和挥手过程

有了文件传输过程的分析，握手和挥手便很容易理解，因为实际上只是文件传输的特例。当然，由于握手和挥手的代码在传输之前完成，因此在编写代码时这一部分设计比较困难，后面完成传输过程时又对其进行了一些优化。

相比传输过程，握手和挥手主要是需要处理流程上的细节。


```

        wrong_times++;
        continue;
    }
}
}
else {
    //timeout
    return false;
}
}
}

```

挥手

由上面的流程图所示，挥手过程仅应当发生在文件传输的间隔中。每次等待用户传送新文件时，用户有两种选择：传或不传。若传则发送文件信息，不传发送挥手信息。不管如何，这时接收端一定处在等待接收文件信息的阶段。

发送端

用户没有给出文件名或者选择放弃传送，仅以第一种调用情况为例：

```

if (file_path.empty()) {
    //close the connection
    return bye_bye();
}

```

挥手成功退出程序，流程结束。

```

int bye_bye() {
    //send FIN
    packet sndpkt = make_pkt(FIN);
    udt_send(sndpkt);
    if (!wait_FIN_ACK()) {
        print_message("Failed to receive FIN ACK", ERR);
        return 1;
    }
    else

```

```

    {
        print_message("Connection closed elegantly, Task
finished!", SUC);
        return 0;
    }
}

```

接收端

首先需要介绍接收端等待文件信息的逻辑。如一开始的流程图所示，这发生在握手刚完成或文件传输间隙。如果此时无响应，说明发送端异常退出，接收端也应当退出。

```

if (!ready_for_file(file_name, file_size)) {
    print_message("Exit because of no response", INFO);
    return 0;
}

```

`ready_for_file` 的设计:

首先为了方便状态机设计，约定传送文件信息的报文序列是1。成功收到消息返回ACK。若收到握手消息，通过递归调用达到重置计时的作用。

在此过程中若发送方发送挥手消息，给予回应并退出程序，流程结束。

```

bool ready_for_file(string &file_name, int &file_size) {
    packet rcvpkt;
    print_message("Waiting for file info", INFO);
    if (rdt_rcv(rcvpkt)) {
        if (has_seq1(rcvpkt)) {
            print_message("File name: " + string(rcvpkt.data),
DEBUG);

            print_message("File size: " +
to_string(rcvpkt.head.option), DEBUG);
            file_name = string(rcvpkt.data);
            file_size = rcvpkt.head.option;
            string file_path = get_file_path(file_name);

```



```

        print_message("File will be saved to " + file_path,
DEBUG);

        print_message("Ready to receive files", SUC);
        packet sndpkt = make_pkt(ACK, 1);
        udt_send(sndpkt);
        return true;
    } else if (isSYN(rcvpkt)) {
        //if the ack is lost, the sender will resend the SYN
packet
        print_message("Received a SYN packet, reset the
timer", WARNING);
        // wait for the file info again
        return ready_for_file(file_name, file_size);
    } else if (isFIN(rcvpkt)) {
        print_message("Received a FIN packet, close the
connection", SUC);
        packet sndpkt = make_pkt(ACK_FIN);
        udt_send(sndpkt);
        return false;
    } else {
        print_message("Received a wrong packet", ERR);
        return false;
    }
} else {
    print_message("Timeout when waiting for file info", ERR);
    return false;
}
}

```

其他工具类

校验和

由于此次实验并没有要求可变ip和端口号，因此不必加入伪首部的校验。编写代码如下：

```

u_short check_sum(u_short *packet, int packet_len) {

```

```

u_long sum = 0;
// make 16 bit words adjacent
int count = (packet_len + 1) / 2;
auto *temp = new u_short[count + 1];
memset(temp, 0, count + 1);
memcpy(temp, packet, packet_len);
while (count--) {
    sum += *temp++;
    //overflow carry
    if (sum & 0xFFFF0000) {
        sum &= 0xFFFF;
        sum++;
    }
}
//complement
return ~(sum & 0xFFFF);
}

```

校验方法:

```

bool not_corrupt(packet &p) {
    return check_sum((u_short *) &p, HEAD_SIZE + p.head.data_size)
    == 0;
}

```

创建数据包

```

packet make_pkt(u_int flag, u_int seq = 0, u_short data_size = 0,
const char *data = nullptr, u_short window_size = 0,
                u_int option = 0) {
    packet pkt;
    pkt.head.flag = flag;
    pkt.head.seq = seq;
    pkt.head.window_size = window_size;
    pkt.head.data_size = data_size;
    pkt.head.option = option;
}

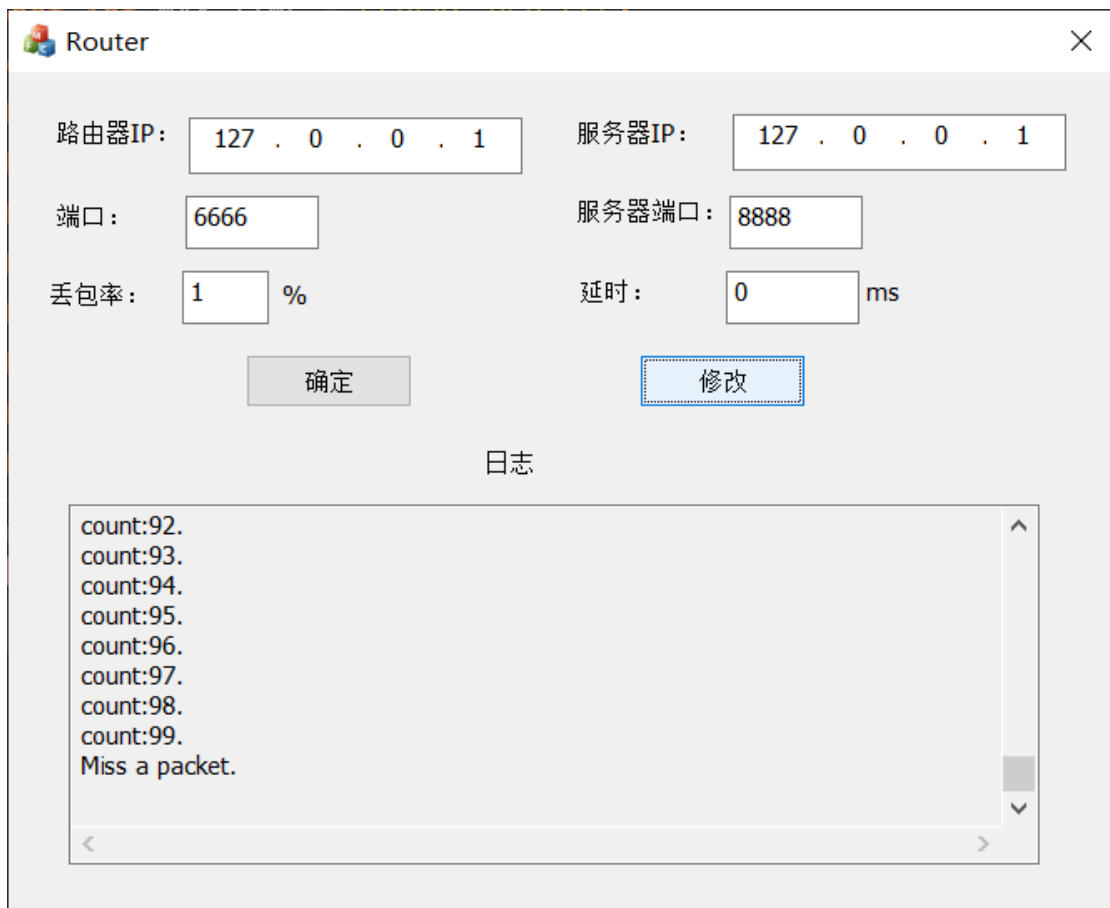
```

```
if (data  $\neq$  nullptr) {  
    memcpy(pkt.data, data, data_size);  
}  
pkt.head.check_sum = check_sum((u_short *) &pkt, PACKET_SIZE);  
return pkt;  
}
```

程序演示

建立连接

路由器设置：



The image shows a 'Router' configuration window with the following fields and buttons:

- 路由器IP: 127 . 0 . 0 . 1
- 服务器IP: 127 . 0 . 0 . 1
- 端口: 6666
- 服务器端口: 8888
- 丢包率: 1 %
- 延时: 0 ms
- Buttons: 确定 (Confirm), 修改 (Modify)
- 日志 (Log) section with a scrollable text area containing:
count:92.
count:93.
count:94.
count:95.
count:96.
count:97.
count:98.
count:99.
Miss a packet.

接收端开启的稍微晚一些，可以看到发送端有一些重发的握手包：

发送端：

```
C:\Users\LENOVO\Desktop\rdt_v1\cmake-build-debug\sender.exe
[TIP] Please input the router's IP address, press ENTER to use default address:

[INFO] Router IP: 127.0.0.1
[WARNING] Resend handshake packet
[WARNING] Resend handshake packet
[SUCCESS] Handshake successfully
[TIP] Please fill the file path, press ENTER to exit
|
```

接收端：

```
C:\Users\LENOVO\Desktop\rdt_v1\cmake-build-debug\receiver.exe
[INFO] Receiver is running on localhost...
[INFO] Waiting for handshake
[SUCCESS] Handshake successfully
[INFO] Waiting for file info
|
```

发送端没有进行文件发送，接收端超时退出：

```
C:\Users\LENOVO\Desktop\rdt_v1\cmake-build-debug\receiver.exe
[INFO] Receiver is running on localhost...
[INFO] Waiting for handshake
[SUCCESS] Handshake successfully
[INFO] Waiting for file info
[ERROR] Timeout, no packet received
[ERROR] Timeout when waiting for file info
[INFO] Exit because of no response

进程已结束,退出代码0
```

异常丢包提示：

```
[DEBUG] Sent packet number 154 with seq 0
[DEBUG] Sent packet number 155 with seq 1
[DEBUG] Sent packet number 156 with seq 0
[DEBUG] Sent packet number 157 with seq 1
[DEBUG] Sent packet number 158 with seq 0
[WARNING] Resend packet with seq 1
[DEBUG] Sent packet number 159 with seq 1
[DEBUG] Sent packet number 160 with seq 0
[DEBUG] Sent packet number 161 with seq 1
[DEBUG] Sent packet number 162 with seq 0
[DEBUG] Sent packet number 163 with seq 1
```

发送端文件发送完毕:

```
[DEBUG] Sent packet number 1509 with seq 1
[DEBUG] Sent packet number 1510 with seq 0
[DEBUG] Sent packet number 1511 with seq 1
[DEBUG] Sent packet number 1512 with seq 0
[DEBUG] Sent packet number 1513 with seq 1
[DEBUG] Sent packet number 1514 with seq 0
[DEBUG] Sent packet number 1515 with seq 1
[SUCCESS] File sent successfully!
[INFO] Time used: 50639ms
[TIP] Do you want to send another file? (Y/N)
```

多文件接收

```
[DEBUG] Received packet 11684, with seq 0
[DEBUG] Received packet 11685, with seq 1
[DEBUG] Received packet 11686, with seq 0
[DEBUG] Received packet 11687, with seq 1
[DEBUG] Received packet 11688, with seq 0
[SUCCESS] Received file successfully
[INFO] Time used: 61865ms
[SUCCESS] File saved to C:\Users\LENOVO\Desktop\rdt_v1\receiver_files\3.jpg
[TIP] Received 2 files till now
[INFO] Waiting for file info
```

github: https://github.com/Lunaticsky-tql/rdt_on_udp

