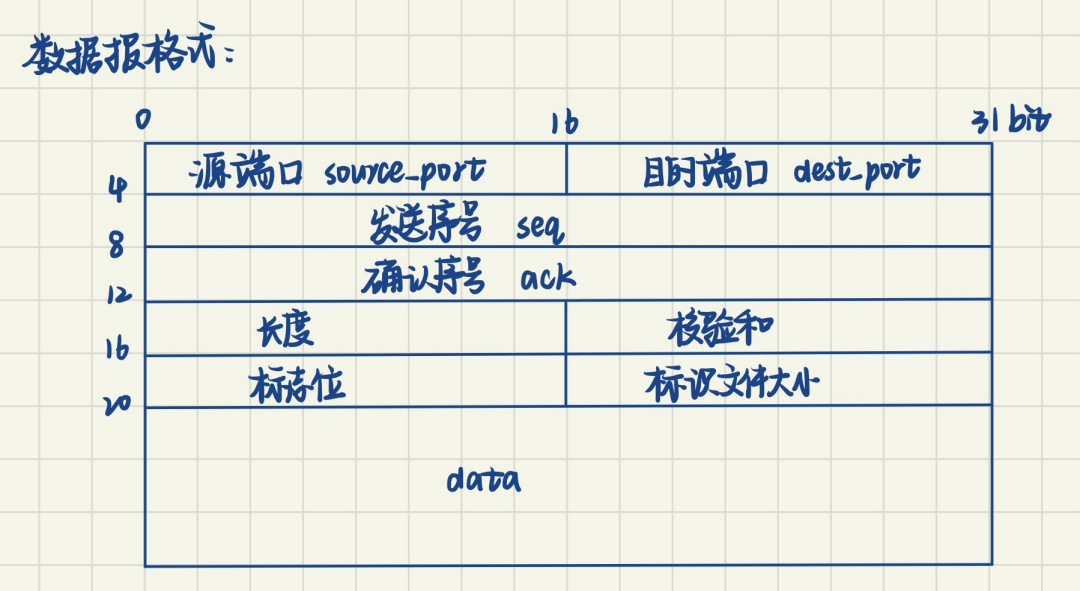
Lab3-2 基于滑动窗口的流量控制机制

2010239 李思凡

一、数据报格式

基于UDP数据报格式来设计本实验中传输的数据报格式，添加了发送序列号、确认序列号、标志位和标识文件大小，如下图所示。



使用结构体来存储数据报的格式，整个数据报占8192字节，data段占8172字节。

```
        // 定义消息格式
        struct Message {
            unsigned short source_port = 0;
            unsigned short dest_port = 0;
            int seq = 0;
            int ack = 0;
            unsigned short lenth = 0;    //data的长度
            unsigned short checksum = 0;
            unsigned short flag = 0;
            unsigned short index = 0;    //传完整个文件需要的数据报个数（1个index=0,2个
index=1...）
            char data[DSIZE] = { 0 };
        };
```

其中flag是一些标志位，从低位到高位分别为FIN，SYN，ACK，REP，SF，EF，分别表示“断开连接信息”，“建立连接信息”，“ACK确认号有效”，“重复发送的信息”，“文件头信息”，“文件的最后一条数据报”。接收数据报后根据标志位判断消息类型，分情况进行处理。

二、协议设计

（一）序列号

本次实验在实验3-1的基础上将停等机制改为基于滑动窗口的流量控制机制。在停等机制下，序列号只需要一位，0/1交替使用即可；但是在流水线发送下，一位序列号不够用，需要增加序列号的位数，使用32位int类型存储序列号seq，具体见上述数据报格式图，序列号依旧循环使用。

在发送端每发送一条数据报，序列号+1；在接收端收到正确的序列号消息，期待的序列号+1。

（二）缓冲区

使用滑动窗口机制时，由于连续发送了多条消息，在未收到相应确认时，需要重传，因此需要有发送缓冲区来保存它们以便重传时能获取到。

在本次实验中，设立一个发送缓冲区数组，将文件信息整合成一个个数据报之后，按顺序放入缓冲区数组。滑动窗口在缓冲区上移动。

（三）滑动窗口GBN

本次实验采用GBN作为流水线协议，进行流量控制、以及超时重传机制。设置窗口大小为10，允许发送10个未得到确认的分组。接收端在确认时，只确认连续正确接收分组的最大序列号。超时重传时，重传所有未确认分组。其中，窗口中是当前可用的序列号，包括已发送但未确认的序列号和还未发送但可用的序列号，确认后窗口向前滑动。

涉及到的量有：窗口的基序号base，下一个序列号nextseqnum，窗口大小N=10，缓冲区最大号datanum，接收端期待收到的序列号expectedseqnum。

1. **发送端**：发送端有三个线程，分别用于发送、接收ACK和超时重传。

   （1）发送线程：发送线程一直等待发送。可以发送的条件是发送包的序列号在当前的窗口内，且序列号不能超过发送缓冲区的最大号（nextseqnum<base+N && nextseqnum<datanum）；如果当前要发送的是窗口最左侧的包（base==nextseqnum），则开启定时器。

   （2）接收线程：接收线程一直工作，若收到的有错，则不做处理，等待超时重传；若收到的正确，则窗口向前滑动，到收到的ack+1的位置（即当前已确认的最大序列号的下一位置）。

   （3）超时重传线程：该线程做计时工作，在未超时时不做处理；若超时，则重新发送从base至nextseqnum-1的所有数据包。

2. **接收端**：

   （1）接收端保存一个希望收到的分组序列号expectedseqnum。若收到的校验和正确且序列号为expectedseqnum，则返回expectedseqnum（即按序正确接收的最高序号）作为ack。同时expectedseqnum++，保留数据包内容。

   （2）若收到失序分组，不保留数据，并发送expectednum-1作为ack。

三、代码细节

发送端：

（一）发送线程

首先设置退出发送的标志位exitFlag，由接收线程确认，标志位为1时发送结束。确认nextseqnum在窗口内且不超过datanum时，从发送缓冲区中获取当前要发送的包，进行发送。若当前base==nextseqnum，启动定时器。每发送一条，nextseqnum++。

```
//发送包，使用滑动窗口
```

```
void sendPackage2() {
    exitFlag = 0;
    while (true) {
        if (exitFlag == 1) {
            break;
        }
        if (nextseqnum < base + N && nextseqnum < datanum) {
            int result = -1;
            Message* message = sendBuf[nextseqnum].message;
            cout << "[SEND] : ";
            printMessage(sendBuf[nextseqnum]);
            if (!isLoss()) {
                result = sendto(clientSocket, (char*)message, sizeof(struct
Message), 0, (struct sockaddr*)&serverAddrIn, sizeof(SOCKADDR));
            }
            if (base == nextseqnum) {
                timer.start();
            }
            nextseqnum++;
            cout << "[WINDOW] : base = " << base << " nextseqnum = " <<
nextseqnum << endl;
        }
    }
}
```

（二）接收线程

接收线程一直循环接收消息，若未收到或校验和有误，都不做处理，等待超时重传。若校验和正确，则更新base=recv.ack+1；若base==nextseqnum且nextseqnum>=datanum，说明发送缓冲区中的数据包都发送完了，exitFlag置为1，指示发送线程可以结束发送。

```
void beginRecv() {
    std::thread recvThread([&]() {
        while (true) {
            Message recvMesg;
            int serverLength = sizeof(SOCKADDR);
            int recvLength = recvfrom(clientSocket, (char*)&recvMesg,
sizeof(struct Message), 0, (struct sockaddr*)&serverAddrIn, &serverLength);
            if (recvLength > 0) {
                Message_C recv_C(&recvMesg);
                cout << "[RECV] : ";
                printMessage(recv_C);
                //检查校验和
                if (recv_C.isCk(&recvPheader)) {
                    if (recv_C.isSYN() && recv_C.isACK()) {
                        cout << "[SUCC] : Connection Success! " << endl;
                    }
                    else if (recv_C.isFIN() && recv_C.isACK()) {
                        closesocket(clientSocket);
                        cout << "[SUCC] : Connection Destroyed! " << endl;
                    }
                    else {
                        cout << "[ACK] : Package (SEQ:" << recvMesg.ack << ")
send success! " << endl;
                    }
                    if (recv_C.isFIN()) {
                        timer.stop();
```

```
                        exitFlag = 1;
                        return;
                    }
                    //更新base
                    base = recvMesg.ack + 1 ;
                    cout << "[WINDOW] : base = " << base << " nextseqnum = " <<
nextseqnum << endl;
                    if (base == nextseqnum) {
                        timer.stop();
                        if (nextseqnum >= datanum) {
                            exitFlag = 1;
                        }

                    }
                    else {
                        timer.start();
                    }
                }
                else {
                    cout << "[FAILED] : Checksum failed. Wait for timeout to
resend" << endl;
                }
            }
            else {
                cout << "[FAILED] : Client received failed. Wait for timeout to
resend" << endl;
            }
        }
        });
    recvThread.detach();
}
```

（三）超时重传线程

该线程承担计时工作，在还未超时时，阻塞在while循环里，什么都不做；超时时便将base-nextseqnum-1的数据包全部重新发送。

```
void beginTimeout() {
    std::thread resendThread([&]() {
        while (true) {
            while(!timer.isTimeout()){}
            int i = base;
            do {
                Message* message = sendBuf[i].message;
                cout << "[RESEND] : ";
                printMessage(sendBuf[i]);
                int result = -1;
                if (!isLoss()) {
                    result = sendto(clientSocket, (char*)message, sizeof(struct
Message), 0, (struct sockaddr*)&serverAddrIn, sizeof(SOCKADDR));
                }
            } while (++i < nextseqnum);
        }
        });
    resendThread.detach();
}
```

（四）丢包设置

通过设置丢包代码完成丢包。采用随机数，有1/500的概率会丢包。在每个sendto函数前调用isLoss()函数，判断是否需要丢弃。

```
/定义随机丢包
bool isLoss() {
    if (rand() % 500 < 1) {
        return true;
    }
    return false;
}
```

接收端：

接收端的代码和3-1相比无较大改动。主要改动点在于设置expectedseqnum，每次收到正确的seq就对其++。若收到错误的，则传回的ack=expectedseqnum-1。

```
//校验和错误或序列号错误
else {
    if (recv_C.isSYN()) {
        sendSYNACK(expectedseqnum-1);
    }
    else if(recv_C.isFIN()){
        sendFINACK(expectedseqnum - 1);
    }
    else {
        sendACK(expectedseqnum - 1);
    }
}
```

四、演示

以发送图片2为例

（一）建立连接

发送端：

可以看到，发送端发送一条SYN=1的消息，序列号为seq=0；此时base=0，nextseqnum由于发送了一条，变为1。

收到了接收端返回的SYN=1，ACK=1的消息，ack=0表示是对该条的确认。连接建立成功。base=ack+1=1。此时base=nextseqnum，结束发送。

```
Wait for connection.
[SEND] : [Package]: (FIN: 0), (SYN: 1), (ACK: 0), (SF: 0), (EF: 0), (seq: 0), (ack: 0), (len: 0), (cks: 33336)
[WINDOW] : base = 0 nextseqnum = 1
[RECV] : [Package]: (FIN: 0), (SYN: 1), (ACK: 1), (SF: 0), (EF: 0), (seq: 1), (ack: 0), (len: 0), (cks: 33331)
[SUCC] : Connection Success!
[WINDOW] : base = 1 nextseqnum = 1
```

接收端：

可以看到，接收端正确收到了发送端的消息，并返回SYN=1, ACK=1的消息，返回的ack=0。

```
Receive Message:
[Package]: (FIN: 0), (SYN: 1), (ACK: 0), (SF: 0), (EF: 0), (seq: 0), (ack: 0), (len: 0), (cks: 33336)
Send SYN_ACK:
[Package]: (FIN: 0), (SYN: 1), (ACK: 1), (SF: 0), (EF: 0), (seq: 1), (ack: 0), (len: 0), (cks: 33331)
Send SYN_ACK Success!
```

（二）传输文件（正确时）

发送端：

可以看到，当发送了seq=80的数据包后，nextseqnum变为81；窗口长度为10，此时nextseqnum已经超出窗口，不能继续发送了，要等待接收使得base前移。

接着，收到了接收端传来的对ack=71的确认消息，表示数据包71已经发送成功，base向前移动，变为72。此时nextseqnum=81在窗口内，继续发送seq=81的数据包，发完后nextseqnum变为82。

```
[SEND] : [Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 80), (ack: 0), (len: 8172), (cks: 25082)
[WINDOW] : base = 71 nextseqnum = 81
[RECV] : [Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 72), (ack: 71), (len: 0), (cks: 33191)
[ACK] : Package (SEQ:71) send success!
[WINDOW] : base = 72 nextseqnum = 81
[SEND] : [Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 81), (ack: 0), (len: 8172), (cks: 25081)
[WINDOW] : base = 72 nextseqnum = 82
```

接收端：

接收端接收到seq=71的数据包，回复ack=71、seq=72的数据包。

```
Receive Message:
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 71), (ack: 0), (len: 8172), (cks: 25091)
recv.seq=71
ack=71
Send ACK:
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 72), (ack: 71), (len: 0), (cks: 33191)
Send ACK Success!
```

（三）丢包处理和超时重传

丢包：

接收端：

可以看到，接收端连续正确收到的最大分组序号为77，并返给发送端ack=77的数据包，自身的expectedseqnum变为78。但接下来收到的却是seq=79、80的数据包，不是期待得到的序列号。因此接收端仍然返回ack=77的数据包。

```
Receive Message:
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 77), (ack: 0), (len: 8172), (cks: 25085)
recv.seq=77
ack=77
Send ACK:
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 78), (ack: 77), (len: 0), (cks: 33179)
Send ACK Success!
Receive Message:
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 79), (ack: 0), (len: 8172), (cks: 25083)
ack=77
Send ACK:
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 78), (ack: 77), (len: 0), (cks: 33179)
Send ACK Success!
Receive Message:
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 80), (ack: 0), (len: 8172), (cks: 25082)
ack=77
Send ACK:
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 78), (ack: 77), (len: 0), (cks: 33179)
Send ACK Success!
```

发送端：

发送端一直收到的都是ack=77的确认包，base一直为78，nextseqnum=88，等待超时重传。

```
[RECV] : [Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 78), (ack: 77), (len: 0), (cks: 33179)
[ACK] : Package (SEQ:77) send success!
[WINDOW] : base = 78 nextseqnum = 88
[RECV] : [Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 78), (ack: 77), (len: 0), (cks: 33179)
[ACK] : Package (SEQ:77) send success!
[WINDOW] : base = 78 nextseqnum = 88
[RECV] : [Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 78), (ack: 77), (len: 0), (cks: 33179)
[ACK] : Package (SEQ:77) send success!
[WINDOW] : base = 78 nextseqnum = 88
[FAILED] : Client received failed. Wait for timeout to resend
[RECV] : [Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 78), (ack: 77), (len: 0), (cks: 33179)
[ACK] : Package (SEQ:77) send success!
[WINDOW] : base = 78 nextseqnum = 88
[RECV] : [Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 78), (ack: 77), (len: 0), (cks: 33179)
[ACK] : Package (SEQ:77) send success!
[WINDOW] : base = 78 nextseqnum = 88
[RECV] : [Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 78), (ack: 77), (len: 0), (cks: 33179)
[ACK] : Package (SEQ:77) send success!
[WINDOW] : base = 78 nextseqnum = 88
[RECV] : [Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 78), (ack: 77), (len: 0), (cks: 33179)
[ACK] : Package (SEQ:77) send success!
[WINDOW] : base = 78 nextseqnum = 88
```

重传：

发送端：

在等待指定时间后，发送端开始超时重传，seq=78至87的数据包。

```
[RESEND] : [Package]: [FAILED] : Client received failed. Wait for timeout to resend
(FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 78), (ack: 0), (len: 8172), (cks: 25084)
[RESEND] : [Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 79), (ack: 0), (len: 8172), (cks: 25083)
[RESEND] : [Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 80), (ack: 0), (len: 8172), (cks: 25082)
[RESEND] : [Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: [FAILED] : Client received failed. Wait for timeout to resend
0), [RECV] : [Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 79), (ack: 78), (len: 0), (cks: 33177)
[ACK] : Package (SEQ:78) send success!
[WINDOW] : base = 79 nextseqnum = 88
```

接收端：

接收端收到seq=78的数据包后，开始回复ack=78。

```
Receive Message:
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 78), (ack: 0), (len: 8172), (cks: 25084)
recv.seq=78
ack=78
Send ACK:
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 79), (ack: 78), (len: 0), (cks: 33177)
Send ACK Success!
Receive Message:
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 79), (ack: 0), (len: 8172), (cks: 25083)
recv.seq=79
ack=79
Send ACK:
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 80), (ack: 79), (len: 0), (cks: 33175)
Send ACK Success!
Receive Message:
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 80), (ack: 0), (len: 8172), (cks: 25082)
recv.seq=80
ack=80
Send ACK:
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 81), (ack: 80), (len: 0), (cks: 33173)
Send ACK Success!
```

（四）断开连接

发送端：

base=nextseqnum=723时，图片2的文件传输完毕。开始发送断开连接请求。可以看出，发送端发送了FIN=1的断连消息，并收到相应的ack，关闭连接。

```
[WINDOW] : base = 723 nextseqnum = 723
Send file: ./test/2.jpg size: 5898505 in 35679ms with throughoutput in 165.321KB/s !
[FAILED] : Client received failed. Wait for timeout to resend
[SEND] : [Package]: (FIN: 1), (SYN: 0), (ACK: 0), (SF: 0), (EF: 0), (seq: 0), (ack: 0), (len: 0), (cks: 33337)
[WINDOW] : base = 0 nextseqnum = 1
[FAILED] : Client received failed. Wait for timeout to resend
[RECV] : [Package]: (FIN: 1), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 1), (ack: 0), (len: 0), (cks: 33332)
[SUCC] : Connection Destroyed!
```

接收端：

接收端收到FIN=1的消息，并返回FIN=1，ACK=1的确认消息。

```
[SUCC] : File ./test/2.jpgReceive Success!
ack=722
Send ACK:
[Package]: (FIN: 0), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 723), (ack: 722), (len: 0), (cks: 31889)
Send ACK Success!
Receive Message:
[Package]: (FIN: 1), (SYN: 0), (ACK: 0), (SF: 0), (EF: 0), (seq: 0), (ack: 0), (len: 0), (cks: 33337)
Send FIN_ACK:
[Package]: (FIN: 1), (SYN: 0), (ACK: 1), (SF: 0), (EF: 0), (seq: 1), (ack: 0), (len: 0), (cks: 33332)
Send FIN_ACK Success!
```