华中科技大学计算机学院
《计算机通信与网络》实验报告

班级 CS2005 姓名 董玲晶 学号 U202090063

项目	Socket 编程（40%）	数据可靠传输协议设计（20%）	CPT 组网（20%）	平时成绩（20%）	总分
得分					

教师评语：

教师签名：

给分日期：

目　　录

实验二 数据可靠传输协议设计实验

1.1 环境

1.1.1 开发平台

本机：Intel(R) i5-10210U CPU + Windows 11 家庭中文版 + Vscode2022

1.1.2 运行平台

（1） Linux 多核 GPU 服务器 + Vscode2022

（2） 模拟网络环境 API，模拟的网络环境实现了以下功能：

- 应用层的数据向下递交给发送方运输层 Rdt 协议；
- 接收方运输层 Rdt 协议收到差错检测无误的报文后向上层应用层递交；
- 将发送方运输层 Rdt 协议准备好的报文通过网络层递交给接收方，在递交过程中按一定的概率会产生丢包、报文损坏；
- 定时器的启动、关闭、定时器 Timeout 后通知发送方运输层 Rdt 协议。

1.2 系统功能需求

（1） 实现基于 GBN 的可靠传输协议；

（2） 实现基于 SR 的可靠传输协议；

（3） 实现 GBN 协议的基础上，根据 TCP 的可靠数据传输机制实现一个简化版的 TCP 协议，要求如下：

- 报文段格式、接收方缓冲区大小和 GBN 协议一样保持不变；
- 报文段序号按照报文段为单位进行编号；
- 单一的超时计时器，不需要估算 RTT 动态调整定时器 Timeout 参数；
- 支持快速重传和超时重传，重传时只重传最早发送且没被确认的报文段；
- 确认号为收到的最后一个报文段序号；
- 不考虑流量控制、拥塞控制；

（4） 实现可靠的运输层协议且只考虑单向传输，即：只有发送方发送数据报文，接收方仅仅接收报文并给出确认；

（5） 要求实现具体协议时，指定编码报文序号的二进制位数（例如 3 位二进制编码报文 序

号）以及窗口大小（例如大小为 4），报文段序号必须按照指定的二进制位数进行编码。

1.3 系统设计

（1） 基于 GBN 的可靠传输协议

① GBN 发送方

Ⅰ.上层调用

- 上层要发送数据时,发送方先检查发送窗口是否已满。
- 如果未满,则产生一个帧并将其发送;如果窗口已满,发送方只需将数据返回给上层,暗示窗口已满.

Ⅱ.收到 ACK

- GBN 协议中,对 n 号帧的确认采用累计确认,表明接收方已经收到 n 号帧和它之前的全部帧

Ⅲ.超时事件

- 若出现丢失和时延过长帧时,发送方将**重传所有已发送但未收到确认的帧**

② GBN 接收方

Ⅰ.若正确收到 n 号帧,并且按序,那么接收方为 n 帧返回一个 ACK,并将该帧中的数据部分交给上层

Ⅱ.其余情况都丢弃帧,并为最近按序接收的帧重新发送 ACK.接收方无需缓存任何失序帧,只需要维护一个信息: expectedseqnum(下一个按序接收的帧序号)

③ 方案设计

- GBNRdtSender 和 GBNRdtReceiver 接口的实现，分别在对应的头文件里；
- GBNRdtReceiver 类的函数具体实现：GBNRdtReceiver()对各接口数据进行初始化或更新、~GBNRdtReceiver()、receiver()接收报文并检查校验和、处理；
- GBNRdtSender 类的函数具体实现：GBNRdtSender()对各接口数据进行初始化或更新、~GBNRdtSender()、send()发送报文。**除此之外，由于在 GBN 协议中，当发送方的发送窗口满了时应该返回 RdtSender 处于等待状态，即 true，所以需要增加 getWaitingState()函数；同时还应有超时处理函数。**
- GBN.cpp 文件，main 函数启动模拟网络环境，实现最上层的调用。

（2） 基于 SR 的可靠传输协议

① SR 发送方

Ⅰ.上层的调用

从上层收到数据后，SR 发送方检查下一个可用于该帧的序号.如果序号位于发送窗口内，则发送该数据帧.否则要么将数据缓存,要么返回给上层之后再传输

Ⅱ.收到 ACK

SR 发送方将这个被确认的帧标记为已接收.如果该帧序号是窗口的下界,则窗口向前移动到最小序号的未被确认帧处.如果窗口移动了且有序号在窗口内的未发送帧,则发送这些帧.

Ⅲ.超时事件

只重传超时帧

② SR 接收方

SR 接收方将确认一个正确接收的帧而**不管其是否按序,失序的帧将被缓存,并返回给发送方一个该帧的确认帧,直到所有帧(序号更小的帧)都被收到为止,这时将一批帧顺序交付上层**,然后向前移动滑动窗口。

序号在 rcv_base 至 rcv_base + N - 1 内的分组被正确接受。如果该分组不是期望的分组，那么缓存，如果是，那么给应用层并且看缓存里面有没有后续，有就直接一起给应用层。序号在 rcv_base - N 至 rcv_base - 1 内的分组被正确接受。返回一个确认 ACK。表示我已经收到了。

③ 方案设计

- SRRdtSender 和 SRRdtReceiver 接口的实现，分别在对应的头文件里；
- SRRdtReceiver 类的函数具体实现：SRRdtReceiver()对各接口数据进行初始化或更新、~SRRdtReceiver()、receiver()检查校验和并处理；**由于要保证接收到到的都是落在当前窗口里的，所以需要增加一个 isinwindow()函数来判断。**
- SRRdtSender 类的函数具体实现：SRRdtSender()对各接口数据进行初始化或更新、~SRRdtSender()、send()发送报文。除此之外，由于在 SR 协议中，当发送方的发送窗口满了时应该返回 RdtSender 处于等待状态,即 true,所以需要增加 getWaitingState()函数；
- SRcpp 文件，main 函数启动模拟网络环境，实现最上层的调用。

（3） 基于 GBN 的简化版 TCP 协议

由于是在 GBN 的基础上添加功能，因此基础功能和实现都与 GBN 协议相同，但根据实验手册要求，仍需要增加一些新的功能。1.支持快速重传，重传时只重传最早发送且没被确认的报文段；2.确认号为收到的最后一个报文段序号；

实现方法设计：对于新增的快速重传功能，只需增加一个计数器 redundack，对收到的 ACKn 进行计数，当达到 3 时，重置计时器并重传最早发送且没被确认的报文段。而对于确认号的设

置，只需要标记最后一次确认的报文序号，在检验到校验和正确但不是当前期待收到的报文序号时重传此标记的序号即可。

1.4 系统实现

1.4.1 GBN

1. GBNRdtSender 和 GBNRdtReceiver 接口实现

（1）GBNRdtReceiver 类，继承自 RdtReceiver

表 1.4.1-1 GBNRdtReceiver 类

属性作用域	类型	变量	注释
Private	Int	expectSequenceNumberRcvd	期待收到的下一个报文序号
	Int	seqsize	使用的序号长度
	Int	winsizeN	窗口大小
	Packet	lastAckPkt	上次发送的确认报文
Public	Func	GBNRdtReceiver()	参数更新
	Func	receive()	接收报文
	Func	printSlideWindow()	打印结果

（2）GBNRdtSender 类，继承自 RdtSender

表 1.4.1-2 GBNRdtSender 类

属性作用域	类型	变量	注释
Private	Int	expectSequenceNumberSend	期待收到的下一个报文序号
	Int	Seqsize	使用的序号长度
	Int	winsizeN	窗口大小
	Packet	Allpacket[seqsize]	所有待发送的报文
	Int	nextseqnum	下一个序号
	Int	base	基序号
	Bool	waitingState	是否处于等待 ACK 状态
Public	Func	GBNRdtSender()	参数更新
	Func	send()	发送报文
	Func	printSlideWindow()	打印结果
	Func	getWaitingState()	获取等待状态
	Func	receive()	接收确认 ACK
	Func	timeoutHandler()	超时处理

2. GBNRdtReceiver 类函数具体实现

（1）GBNRdtReceiver()

设置序列号长度 seqsize 为 8，窗口大小为 4，初始状态下将上次发送的确认包的确认序号为-1，使得当第一个接受的数据包出错时该确认报文的确认号为-1。加载包后计算校验和。

（2） receive(const Packet &packet)

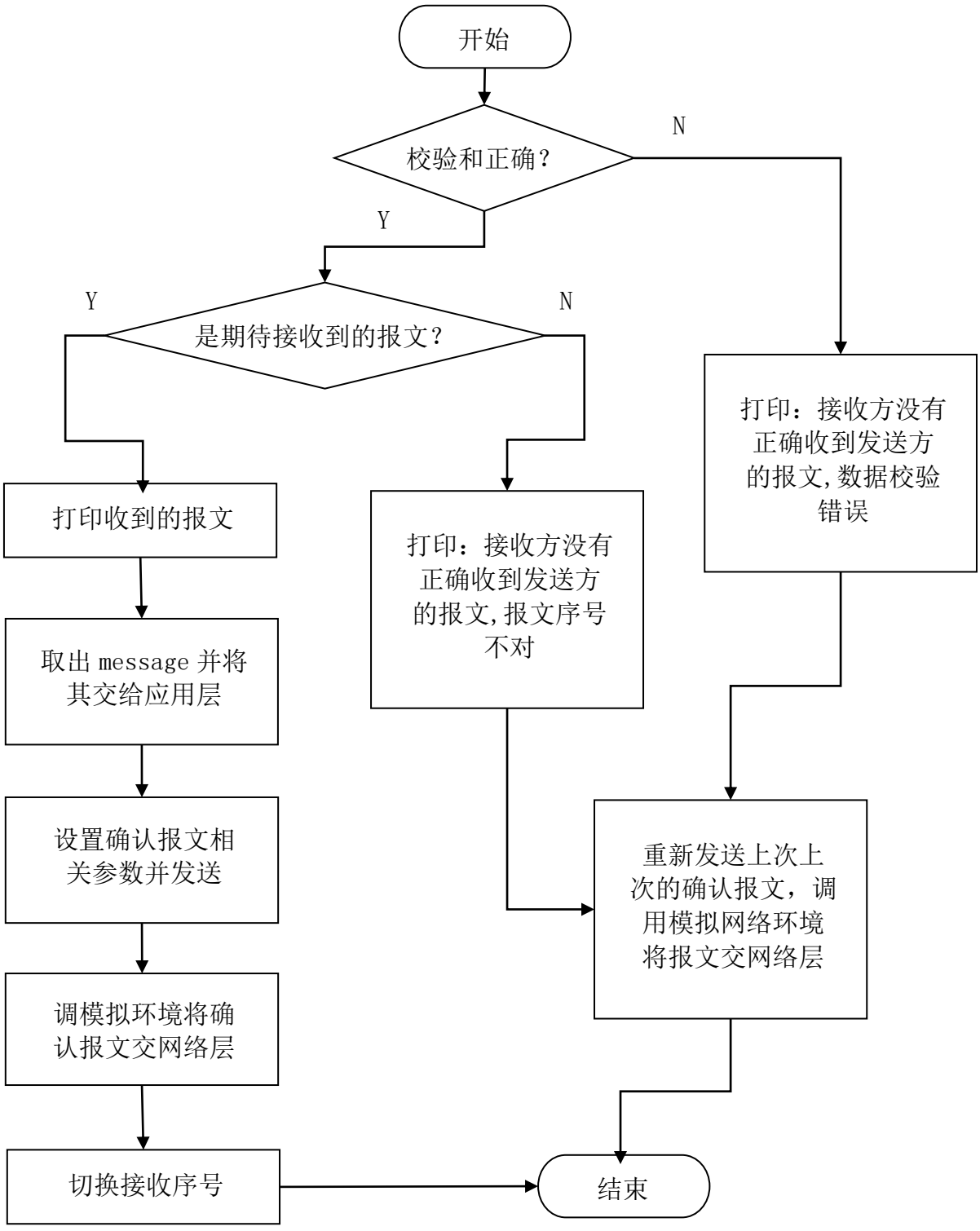

图 1.4.1-1 GBNRdtReceiver 接收报文类函数 receive()的实现

3. GBNRdtSender 类函数具体实现

（1）GBNRdtSender()

初始化时基序号为 0，下一个序号也为 0；窗口大小和序列长度的设置遵循发送方和接收方统一的原则，分别为 4 和 8；初始化一个 Packet 参数。

（2） getWaitingState() => 公式：(base + winsizeN) % seqsize == nextseqnum % seqsize。

（3） send(const Message &message)

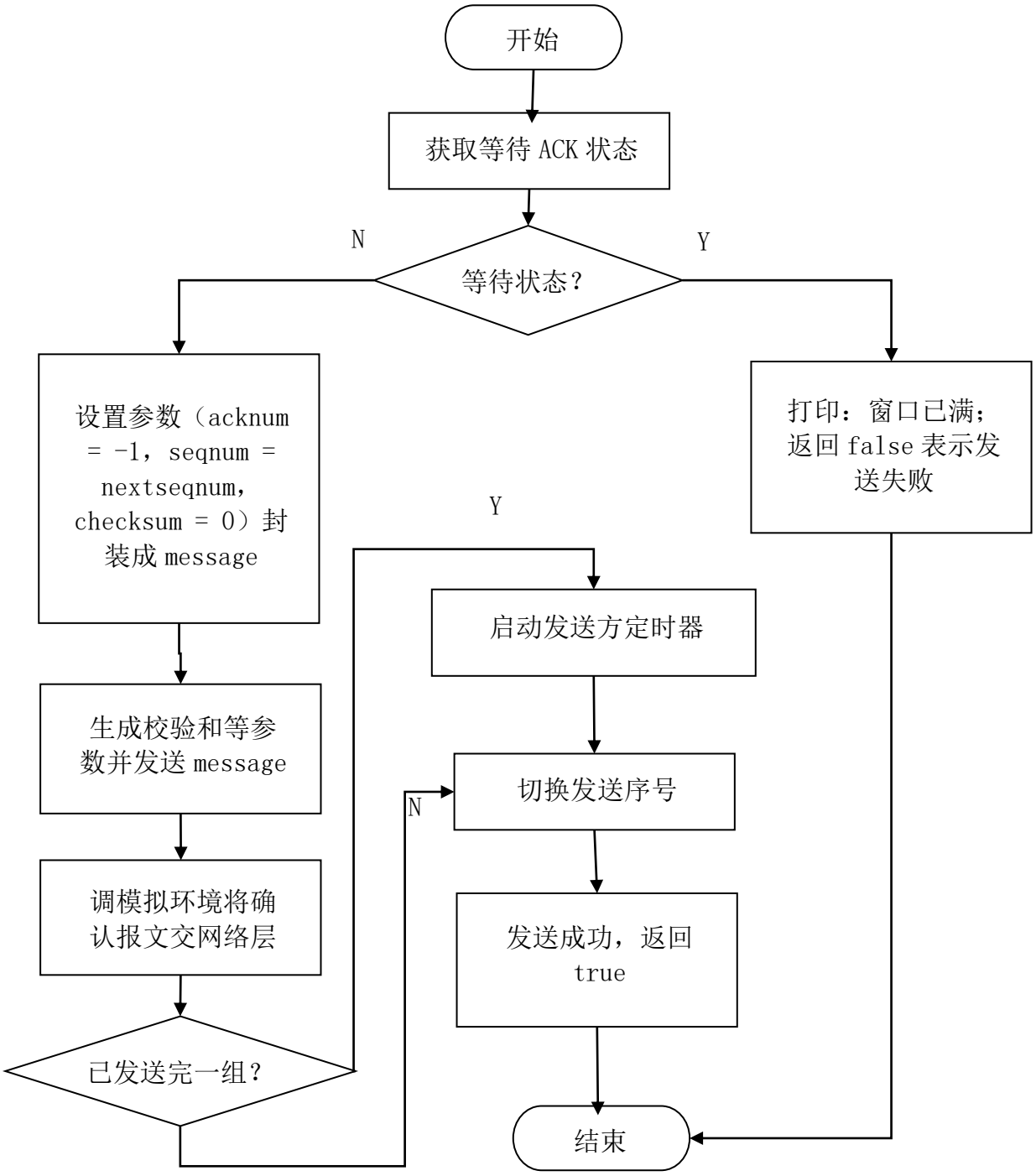

图 1.4.1-2 GBNRdtSender 发送报文类函数 sender()的实现

（4） receive(const Packet &ackPkt)

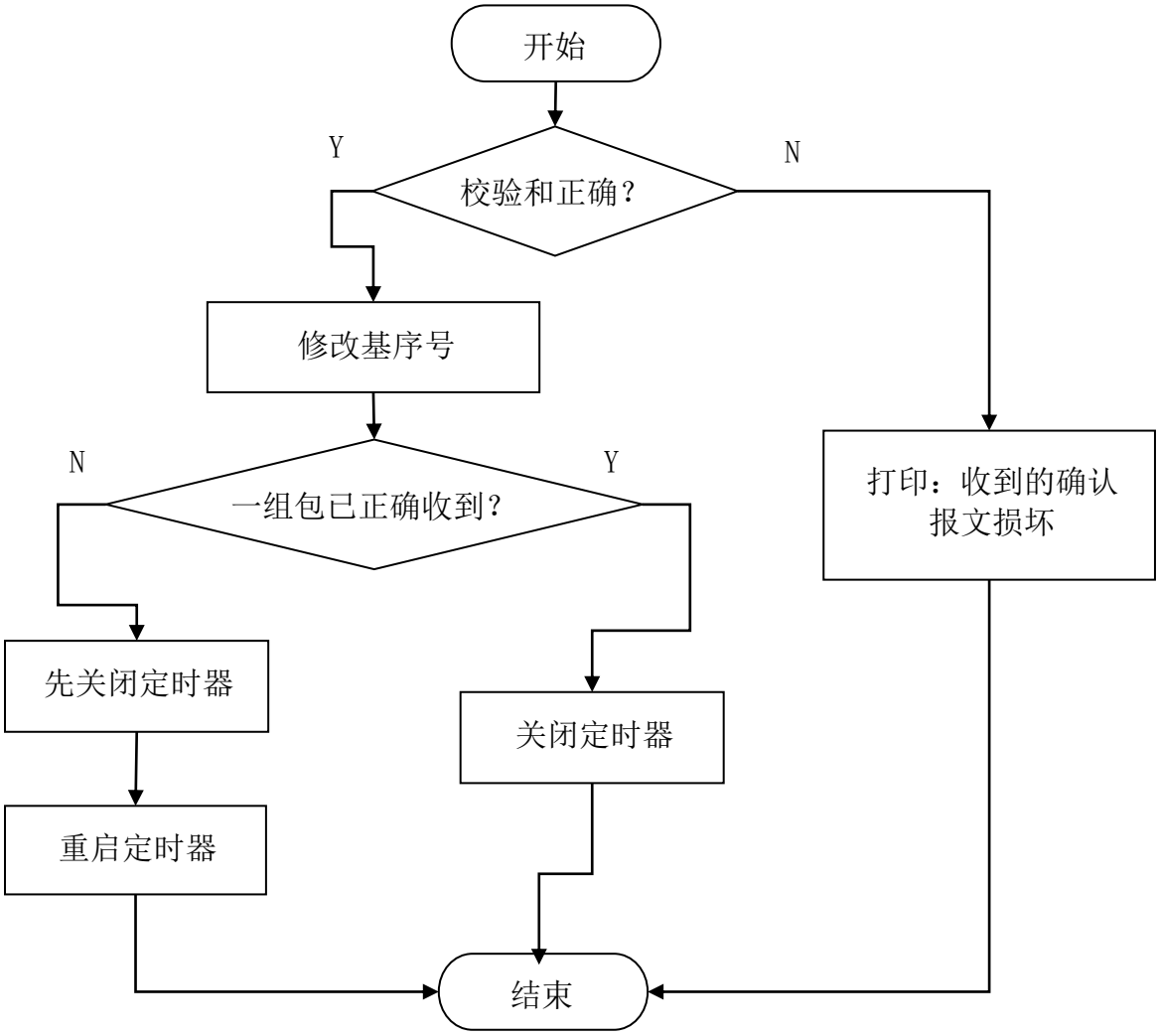

图 1.4.1-3 GBNRdtSender 接收确认报文函数 receive()的实现

（5） timeoutHandler(int seqNum)

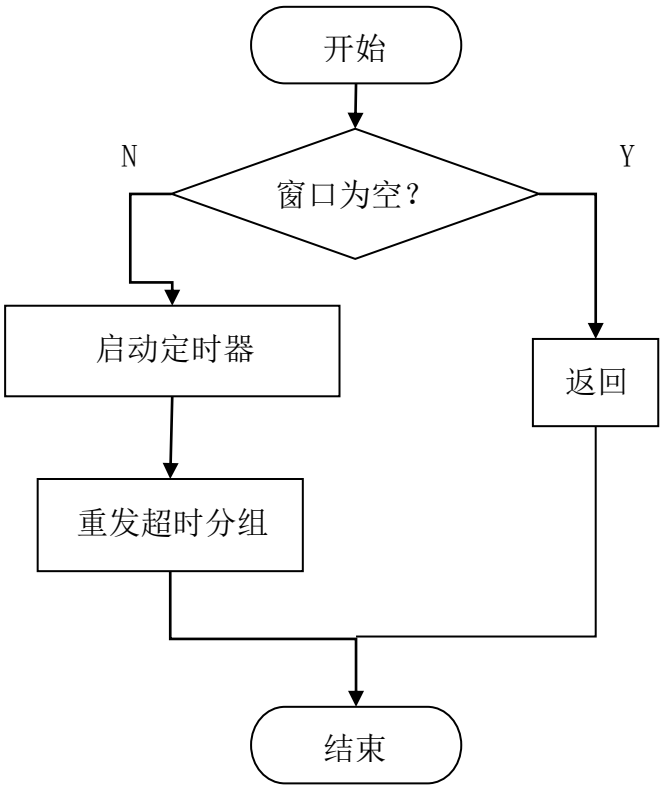

图 1.4.1-4 GBNRdtSender 分组超时函数 timeoutHandler()的实现

（6） printSlideWindow() => 在控制台输出滑动窗口的内容

算法：模板为（1，2，[3]，4，5，6，7）。Seqsize 大小的循环里，遇到基序号打印左括号，遇到窗口大小-1 的序号打印右括号。遇到下一个期待收到的序号时打印方括号。

```
int i;
for (i = 0; i < seqsize; i++)
{
        if (i == base)
                std::cout << "(";
        if (i == this->nextseqnum)
                std::cout << "[" << i <<"]";
        else
        {
                std::cout << i;
        }
        if (i == (base + this->winsizeN - 1) % seqsize)
                std::cout << ")";
        std::cout << "  ";
}
std::cout << std::endl;
```

图 1.4.1-5 GBNRdtSender. printSlideWindow()关键代码

1.4.2 SR

1. SRRdtSender 和 SRRdtReceiver 接口实现

（1）SRRdtReceiver 类，继承自 RdtReceiver

表 1.4.2-1 SRRdtReceiver 类

属性作用域	类型	变量	注释
Private	Int	expectSequenceNumberRcvd	期待收到的下一个报文序号
	Int	seqsize	使用的序号长度
	Int	winsizeN	窗口大小
	Int	tempPacketrcv[8]	
	Packet	lastAckPkt	上次发送的确认报文
	Packet	temppacket[8]	缓存
Public	Func	SRRdtReceiver()	参数更新
	Virtual	~ SRRdtReceiver()	/
	Func	receive()	接收报文
	Func	insinwindow()	是否在窗口内

（2）SRRdtSender 类，继承自 RdtSender

表 1.4.2-2 SRRdtSender 类

属性作用域	类型	变量	注释
Private	Int	expectSequenceNumberSend	期待收到的下一个报文序号
	Int	Seqsize	使用的序号长度
	Int	winsizeN	窗口大小
	Packet	Allpacket[seqsize]	所有待发送的报文
	Int	nextseqnum	下一个序号
	Int	base	基序号
	Int	rcvstatus[8]	接收状态判断
	Bool	waitingState	是否处于等待 ACK 状态
Public	Func	SRRdtSender()	参数更新
	Virtual	~ SRRdtSender()	/
	Func	send()	发送报文
	Func	printSlideWindow()	打印结果
	Func	getWaitingState()	获取等待状态
	Func	receive()	接收确认 ACK
	Func	Isinwindow()	是否错序判断
	Func	timeoutHandler()	定时器设置

2. SRRdtReceiver 类函数具体实现

（1）SRRdtReceiver()

同 GBNRdtReceiver()，详情可见 1.4.1-2 节。

（2）isinwindow(int seqNum) => 判断是否是错序报文，供 receive()调用

（3）receive(const Packet &packet)

逻辑部分与 GBNRdtReceiver.receive()相同。先检查校验和是否正确，若正确，再判断收到报文的序号是否等于接收方期待收到的报文序号一致。若一致，取出 message，使期待收到的下一个报文序号 expectSequenceNumberRcvd+1，同时将数据递交到上层并回传 ACK，接着查询是否已有缓存，若下一个分组已经收到就一并交给应用层并且滑动窗口；若不一致则只是缓存分组并回传 ACK 确认报文。而若校验和错误或者序号错误，则单独回传上一个 ACK，表示当前分组并没有收到。

```
if (this->expectSequenceNumberRcvd == packet.seqnum)
{
        pUtils->printPacket("接收方正确收到发送方的报文", packet);
        Message msg;
        memcpy(msg.data, packet.payload, sizeof(packet.payload));
        pns->delivertoAppLayer(RECEIVER, msg);
        tempPacketrcv[this->expectSequenceNumberRcvd] = 0;
        this->expectSequenceNumberRcvd = (this->expectSequenceNumberRcvd + 1) % seqsize;
        int tmp = this->expectSequenceNumberRcvd;
        for (int i = tmp; i != (tmp + winsizeN - 1) % seqsize; i = (i + 1) % seqsize)
        {
                if (tempPacketrcv[i] == 1)
                {
                        Message msg;
                        memcpy(msg.data, temppacket[i].payload, sizeof(temppacket[i].payload));
                        pns->delivertoAppLayer(RECEIVER, msg);
                        pUtils->printPacket("数据递交到上层", temppacket[i]);
                        tempPacketrcv[i] = 0;
                        this->expectSequenceNumberRcvd = (this->expectSequenceNumberRcvd + 1) %

                }
                else
                        break;
        }
}
```

图 1.4.2-1 SRRdtReceiver.receive()关键代码

3. SRRdtSender 类函数具体实现

（1）INIT() => 供 SRRdtSender()调用

初始化 ACK 接收状态，全设置为 0

（2）SRRdtSender()

同 1.4.1 节中的 GBNRdtSender()，对基序号等参数进行初始化，并调用 INIT()初始化 ACK 接收状态。

（3）getWaitingState()

同 1.4.1 节中的 getWaitingState()相同，公式(base + winsizeN) % seqsize == nextseqnum。

（4）printSlideWindow()

同 1.4.1 节中的 printSlideWindow()相同

（5）isinwindow(int seqNum) => 判断是否是错序 ACK，供接收 ACK 函数 receive()调用

```
if ((base + winsizeN) % seqsize > base)
        return (seqNum >= base) && (seqNum < (base + winsizeN) % seqsize);
else if ((base + winsizeN) % seqsize < base)
        return (seqNum >= base) || (seqNum < (base + winsizeN) % seqsize);
else
{
        return false;
}
```

图 1.4.2-2 SRRdtSender.isinwindow()实现代码

（6）receive(const Packet &ackPkt)

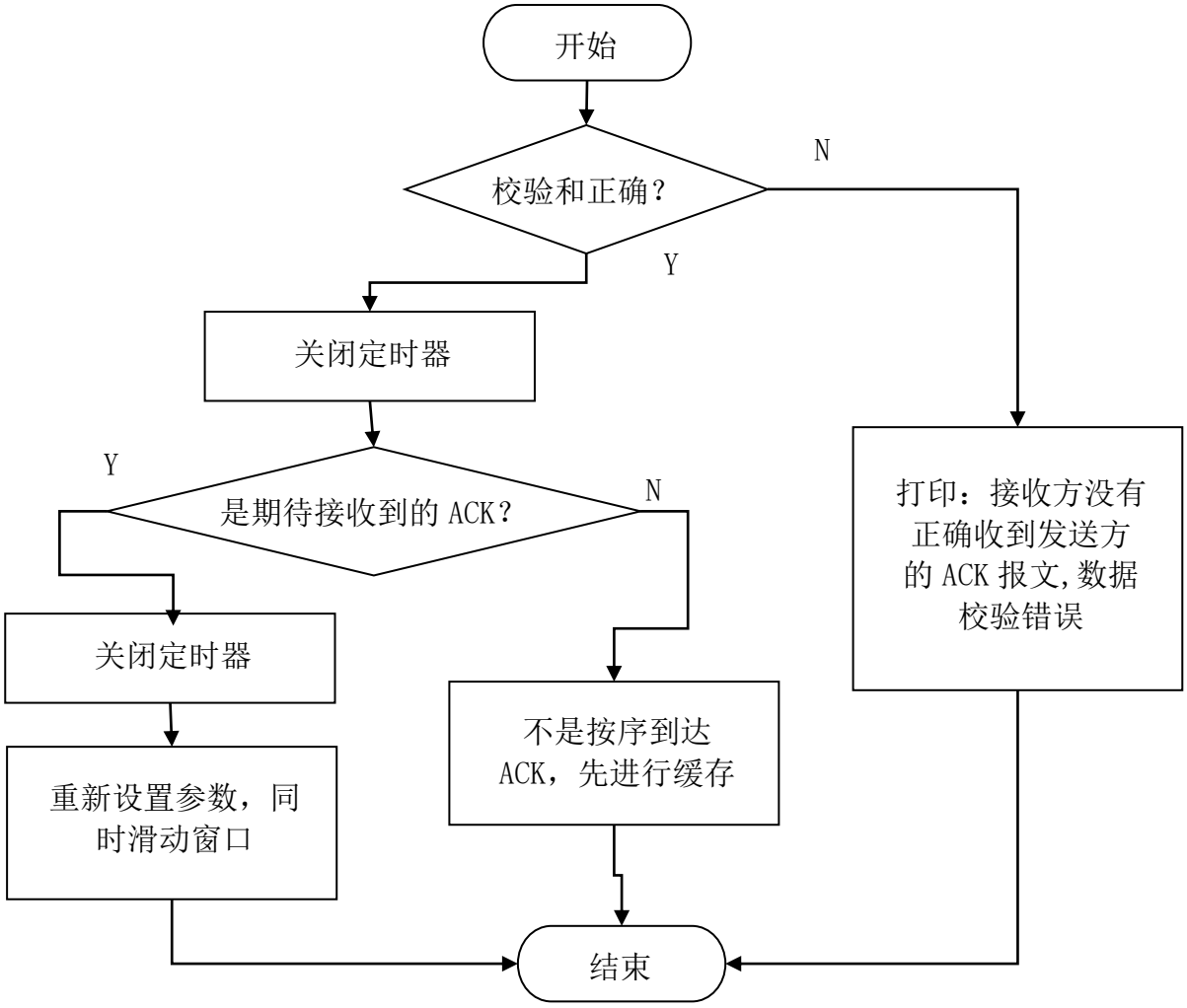

图 1.4.2-3 SRRdtSender.receive()流程图

（7）send(const Message &message)

和 1.4.1 节中的 SRRdtSender.send()类似，详情可见图 1.4.1-2.

```
this->rcvstatus[this->base] = 0;
this->base = (this->base + 1) % seqsize;
int tmp = this->base;
for (int i = tmp; i != (tmp + this->winsizeN - 1) % seqsize; i = (i + 1) % seqsize)
{
        if (this->rcvstatus[i] == 1)
        {
                this->rcvstatus[i] = 0;
                this->base = (this->base + 1) % seqsize;
        }
```

图 1.4.2-4 SRRdtSender.send() 校验和正确且收到期待数据包时的情况

（8）timeoutHandler(int seqNum)

只需启动定时器 -> 调用模拟网络 -> 重发超时分组

1.4.3 TCP

1. TCPRdtSender 和 TCPRdtReceiver 接口实现

（1）TCPRdtReceiver 类，继承自 RdtReceiver

所有接口与 GBNRdtReceiver 类相同

（2）TCPRdtSender 类，继承自 RdtSender（***只列举比 GBN 多增加的接口**）

表 1.4.3-1 TCPRdtSender 类新增接口

属性作用域	类型	变量	注释
Private	Int	redundack	冗余 ACK 计数器

2. TCPRdtReceiver 类函数具体实现

和 GBNRdtReceiver 类函数完全相同，没有任何区别，复用即可。详情流程图和实现内容可见 1.4.1 节 GBNRdtReceiver 类函数的实现。

3. TCPRdtSender 类函数具体实现

（1）TCPRdtSender()

大致和 GBNRdtSender()相同，增加了 redundack 的初始化。

（2）getWaitingState()

同 1.4.1 节中的 getWaitingState()相同，公式(base + winsizeN) % seqsize == nextseqnum。

（3）printSlideWindow()

同 1.4.1 节中的 printSlideWindow()相同.

（4）isinwindow(int seqNum)

判断是否是错序 ACK,供接收 ACK 函数 receive()调用,代码与 SRRdtSender.isinwindow()代码完全相同，原理也相同，不再重复阐述。

（5）send(const Message &message)

和 GBNRdtSender.send()逻辑、代码完全相同，详情可见图 1.4.1-2

（6）receive(const Packet &ackPkt)

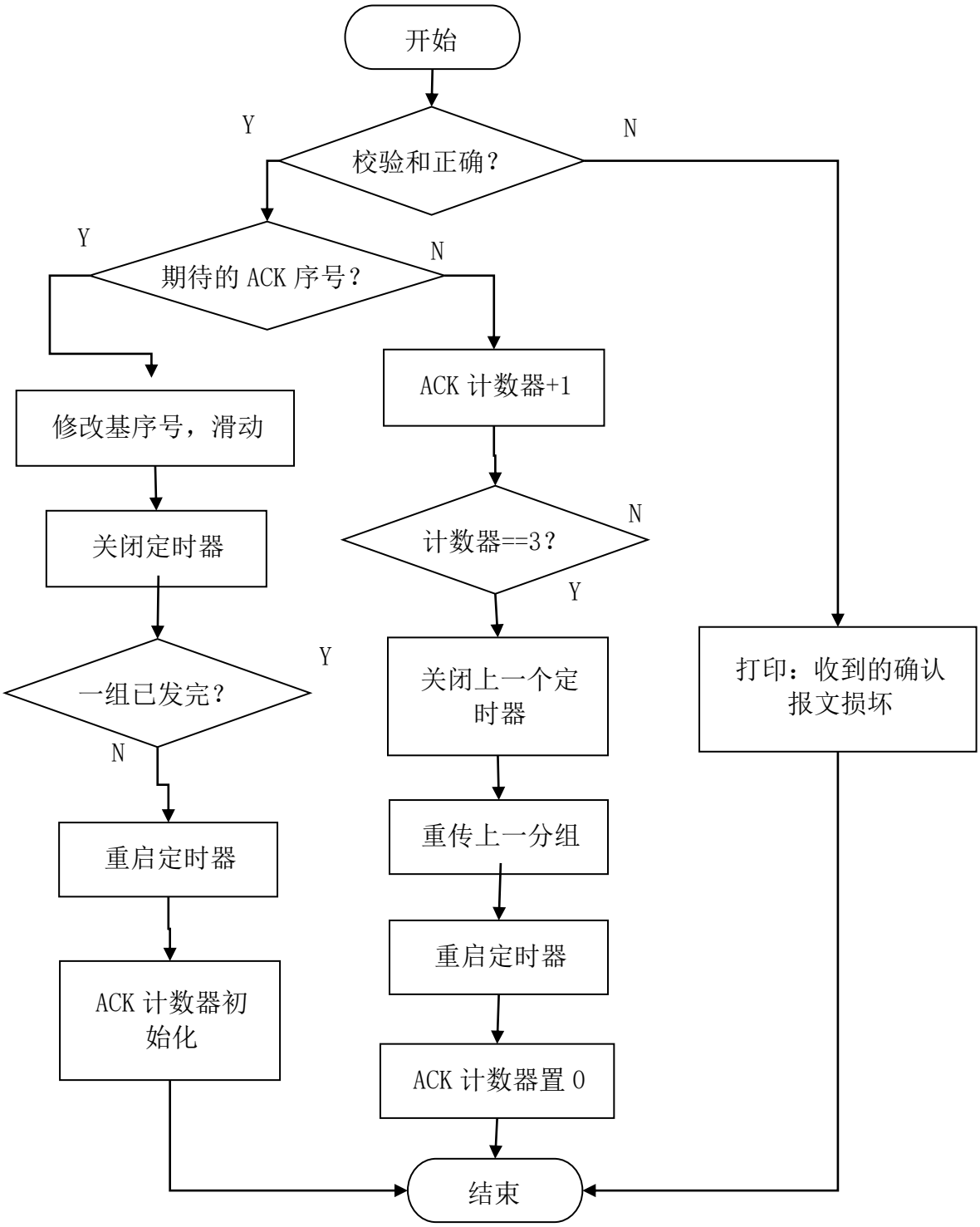

图 1.4.3-1 TCPRdtSender.receive()流程图

（7）timeoutHandler(int seqNum)

当遇到特殊情况，即窗口满时不做任何处理。否则重启定时器并重传超时分组。相对应的代码如下。

```
if (this->base == this->nextseqnum) //窗口为空的特殊情况
        return;
else
{
        pns->startTimer(SENDER, Configuration::TIME_OUT, this->base);
        pns->sendToNetworkLayer(RECEIVER, this->Allpacket[this->base]);
        pUtils->printPacket("重发超时分组", this->Allpacket[this->base]);
}
```

图 1.4.3-2 TCPRdtSender.timeoutHandler()流程图

1.5 系统测试及结果说明

（1）GBN

运行 GBN.cpp 文件，结果如图所示。由于设置的窗口大小是 8，即 0~7.观察滑动窗口的改变，[4|567]完即是[|5670]。从 0 开始复用。对应显示的 seqnum 变量的值也正确，随滑动窗口的改变而改变。

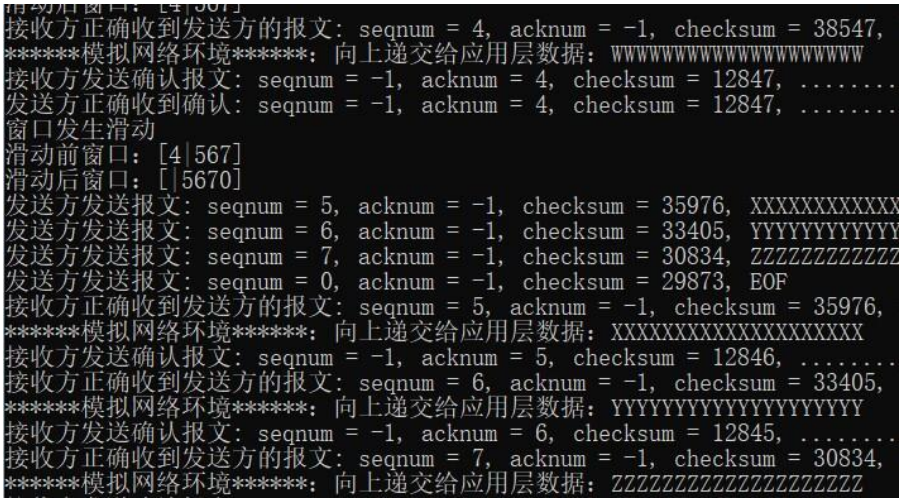

图 1.5-1 GBN 运行结果图

（2）SR

运行 SR.cpp 文件，可观察滑动窗口的变化。滑动前是[5n, 6n, 7n,0n]，滑动后则是[6n, 7n, 0n, 1n]，滑动窗口功能正确。根据模拟网络环境的输出信息，可看到也正确收到了接收方回传的确认 ACK 包。观察定时器的开启和关闭，对应报文的发送和接收，也是相对应的；以及定时器的超时事件，在超时后重传上一个，观察 seqnum 的值，也是正确无误。

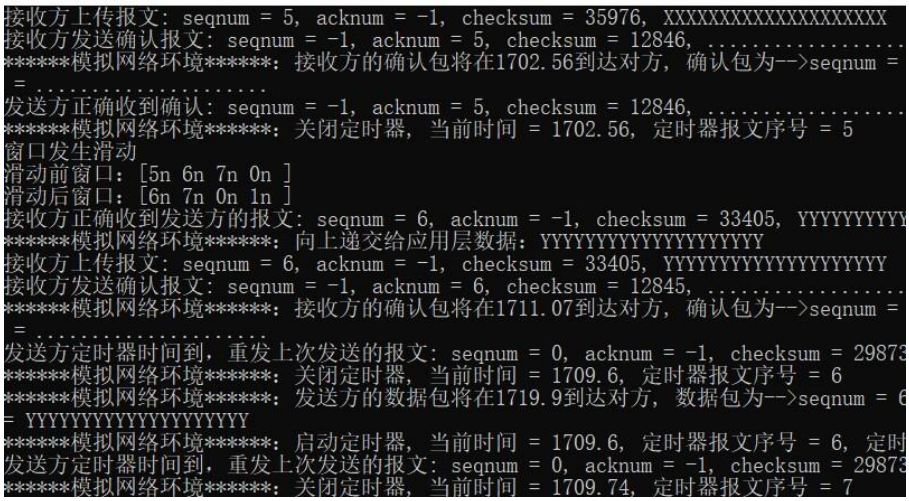

图 1.5-2 SR 运行结果图

（3）TCP

同前面 GBN 和 SR 所述，窗口大小为 4，观察滑动窗口的变化。随着确认 ACK1 的到达，滑动窗口向前滑动，从[1234|]变成[234|5]，5 号为还未发送，234 为已发送但还未收到确认 ACK，待后续收到确认滑动窗口则继续向前滑动。

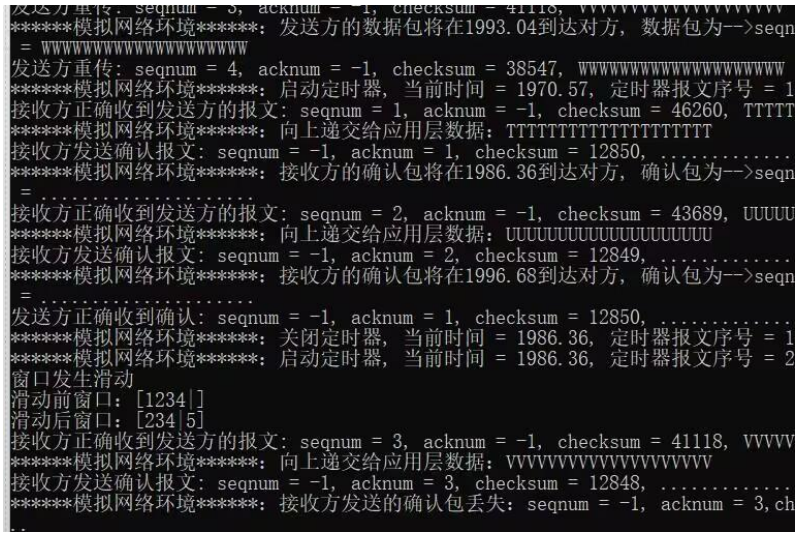

图 1.5-3 TCP 运行结果图

如图 1.5-4 所示。当收到三个冗余 ACK 时，启动快速重传功能，发送方重传，并启动对应的定时器。

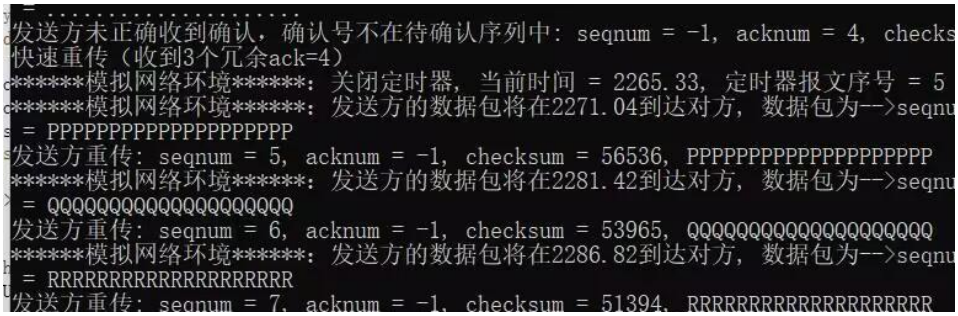

图 1.5-4 TCP 收到三个冗余 ACK 快速重传功能

1.6 参考文献

[1] Computer Networking：A Top-Down Approach，Seventh Edition. James，F.Kurose，Keith，W.Ross

心得体会与建议

2.1 心得体会

本次实验我在实验手册 StopWait 的示例下，完成了基于 GBN、SR 的可靠传输协议，同时基于 GBN 完成了一个简化版 TCP 协议。在完成的过程中，我对各个协议的传输原理和过程理解更加深刻，尤其是对滑动窗口的认识和理解。同时我也更好地掌握了每个协议之间的联系与不同之处。

因为之前没有学过 C++，所以在刚开始做计网的实验（socket 实验）时感觉十分得痛苦，再加上计网实验所写的代码和之前接触的不太相同，对第一次接触的我来说感觉十分的困难。根据老师给的例子完成了第一次实验后，再接触第二次实验我的心态好了很多。由于第二次实验实验手册给出了很优秀的示范代码，也有很详细的模拟网络 API 调用示范，在刚入手时通过观察和分析示例代码和实验手册，解决了我很多的困惑和难题，在此对编写实验手册的老师/助教表示由衷的感谢。个人感觉最有趣的实验是第三次实验，可能是因为是仿真实验而且不用写代码，总体也比较简单和轻松，但加深了我对子网划分、VLAN 设置、IP 设置等方法的掌握。

总体来说，在三次实验里我收获颇多，虽然过程很痛苦，但做完了之后还是很有成就感的。感恩过去，展望未来。

2.2 建议

1. 实验一真的很难，幸好辜老师写了个示范代码，不然可能真的做不出来。但是据我所知好像不是所有的班级都有那份代码。或许可以考虑下次作为实验材料发给同学们，降低一下难度。

2. 实验课的课程安排可以提早一些，不然最后排在期末还有实验课，还要写实验报告，期末还有许多的考试和其他课的结课项目和作业，总体会很匆忙、时间很紧张。