華中科技大學

HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

LAB REPORT

计算机通信与网络实验报告

姓名 王怡彬

学号 U201914858

班级 计卓 2001

时间 2022 年 12 月 26 日

目录

1 实验二 数据可靠传输协议设计实验

1.1 环境

(1) 操作系统: Windows 11 家庭中文版
(2) 处理器: 11th Gen Intel(R) Core(TM) i5-1155G7 @ 2.50GHz
(3) 虚拟内存: 16.0 GB
(4) 集成开发环境: Visual Studio 2022
(5) 编程语言: C++
(6) 依赖库: netsimlib.lib。

1.2 系统功能需求

• 可靠运输层协议实验只考虑单向传输，即：只有发送方发生数据报文，接收方仅仅接收报文并给出确认报文。

• 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。

• 代码实现不需要基于 Socket API，不需要利用多线程，不需要任何 UI 界面。

本实验包括三个级别的内容，具体包括：

（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）在实现 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：

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。

1.3 系统设计

系统主体分为模拟网络环境和可靠数据传输协议层两部分，一下分别对二者的设计进行描述。

1.3.1 模拟网络架构设计

图1.1给出了模拟网络环境架构和学生实现的 Rdt 协议的之间的关系。二者之间需要协同工作。蓝色背景部分为模拟网络环境，橙色背景部分为 Rdt 协议的发送方

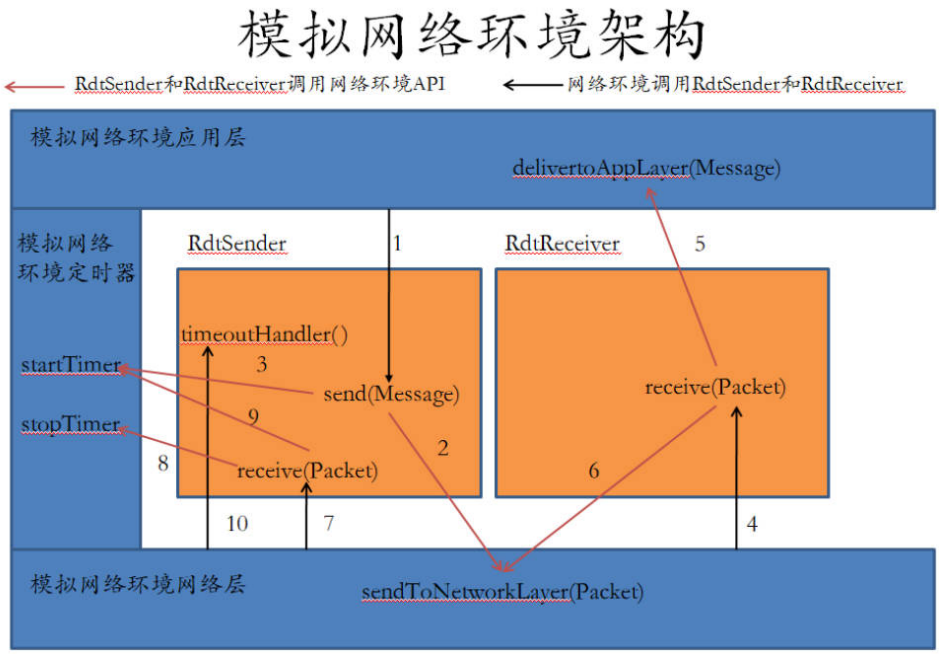

图 1.1: 模拟网络环境架构

(RdtSender) 和《计算机通信与网络》实验指导手册（可靠数据传输协议设计），接收方 (RdtReceiver)。红色箭头表示 RdtSender 和 RdtReceiver 调用模拟网络环境的函数，黑色箭头表示模拟环境调用 RdtSender 和 RdtReceiver 的函数。

1.3.2 可靠数据传输协议层设计

实验共设计并实现了 GBN、SR、简化 TCP 三种传输层协议的发送方和接收方运行代码，并为三种不同协议的接收方与发送方统一设计了抽象父类 RdtReceiver 和 RdtSender，为发送方设计了 send，timeoutHandler，receive 和 getWaitingState 函数，为接收方设计了 receive 函数。

- send(Message) 将模拟网络环境产生的应用层数据进行报文封装并发送给网络层，同时根据不同协议内容进行其他事件处理。
- timeoutHandler() 超时事件发生时，根据不同协议内容对超时事件做出处理。
- receive(Packet) 接收并处理从发送方/接收方收到的报文。
- getWaitingState() 根据当前窗口状态等信息判断发送方是否处于等待确认状态。

1.3.3 模拟网络环境与可靠数据传输协议层之间的调用关系

模拟网络环境模拟实现了应用层和网络层，而需要和学生实现的运输层 Rdt 协议协同工作完成数据的可靠传输。从应用层有数据到来开始，它们之间的调用关系如下所

述：

- 模拟网络环境模拟产生应用层数据，调用 Rtdsender 的 Send(Message) 方法；
- RtdSender 的 Send(Message) 方法调用模拟网络环境的 sendToNetworkLayer(Packet) 方法，将数据发送到模拟网络环境的网络层；
- RtdSender 的 Send(Message) 方法调用模拟网络环境的 startTimer( ) 方法启动定时器；
- 模拟网络环境调用 RdtReceiver 的 receive(Packet) 方法将数据交给 RdtReceiver；
- 如果校验正确，RdtReceiver 调用模拟网络环境的 delivertoAppLayer(Message) 方法将数据向上递交给应用层；
- RdtReceiver 调用模拟网络环境的 sendToNetworkLayer(Packet) 方法发送确认；
- 模拟网络环境调用 RtdSender 的 receive(Packet) 方法递交确认给 RtdSender；
- 如果确认正确，RtdSender 调用模拟网络环境的 stopTimer 方法关闭定时器；
- 如果确认不正确，RtdSender 调用模拟网络环境的 startTimer 方法重启定时器；
- 如果定时器超时，模拟网络环境调用 RtdSender 的 timeoutHandler( ) 方法。

1.4 系统实现

1.4.1 GBN 协议

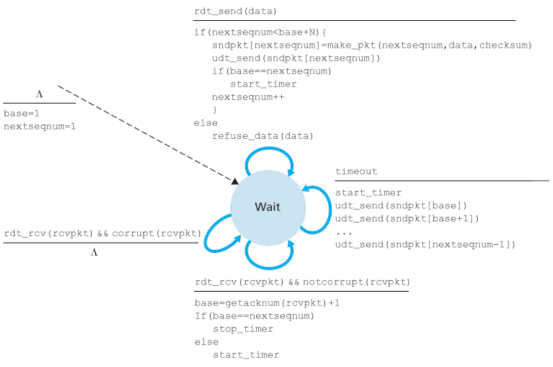

图 1.2: GBN 协议发送方 FSM 图

对象设计 GBN 协议的发送方 FSM 示意图如图1.2所示，接收方 FSM 示意图如图1.3所示。将 GBN 协议的发送方和接收方设计并实现为两个对象 GBNRdtSender 和 GBN-RdtReceiver。

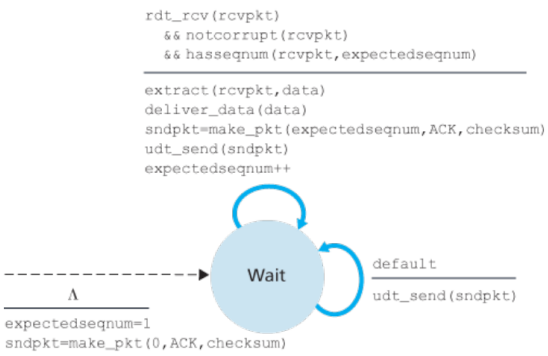

图 1.3: GBN 协议接收方 FSM 图

GBNRdtSender 中定义了静态常量 wndsize 以及 seqsize，用于规定协议的滑动窗口长度为 4，序号长度为 8。同时还定义了两个成员变量：最早未确认分组的序号 base 和下一个待发分组序号 nextSeqNum。sendBuf 为 Packet 类的变长数组，用于作为发送方缓冲区的实现，保存已发送的报文并用于重传。根据 GBN 协议内容，对构造、析构函数和 RdtSender 父类定义的函数进行了实现。

GBNRdtReceiver 中定义了静态常量 seqsize 用于规定协议的序号长度，与发送方保持一致，设置为 8，同时定义了成员变量 expectSequenceNumberRcvd 用于存储期待的下一个分组序号，定义了 Packet 类的成员变量用于存储上一次发送的确认报文。根据 GBN 协议内容，对构造、析构函数和 RdtReceiver 父类定义的函数进行了实现。

GBN 发送方与接收方数据结构定义

```
class GBNRdtSender :
public RdtSender
{
private:
        int base; //最早的未确认分组的序号
        int nextSeqnum; //下一个待发分组的序号
        const int wndsize; //滑动窗口长度，实验建议为4
        const int seqsize; //序号长度，实验建议为8
        Packet* const sendBuf; //发送缓冲区
};
class GBNRdtReceiver :
public RdtReceiver
{
private:
        int expectSequenceNumberRcvd; // 期待的下一个分组序号
        Packet lastAckPkt; //上次发送的确认报文
```

```
17          const int seqsize; //序号长度，实验建议为8
18  };
```

关键函数实现

1. GBNRdtSender::send(const Message& message)

   接收应用层的数据 message 并将其封装为传输层数据包 Packet；由于 GBN 中采取累计确认机制，不管哪个分组超时，都会重传所有未确认的分组，因此只需要一个计时器；当最早的未确认分组序号等于下一个待发分组序号时，即说明接收方接收到了所有的数据包，此时再次发送数据包时应重新启动定时器；调用 sendToNetworkLayer 发送数据包；发送完毕后，将滑动窗口循环右移并打印当前窗口状况。函数接收应用层的数据 message，返回 bool 变量，代表发送是否成功。函数流程图如图1.4所示。

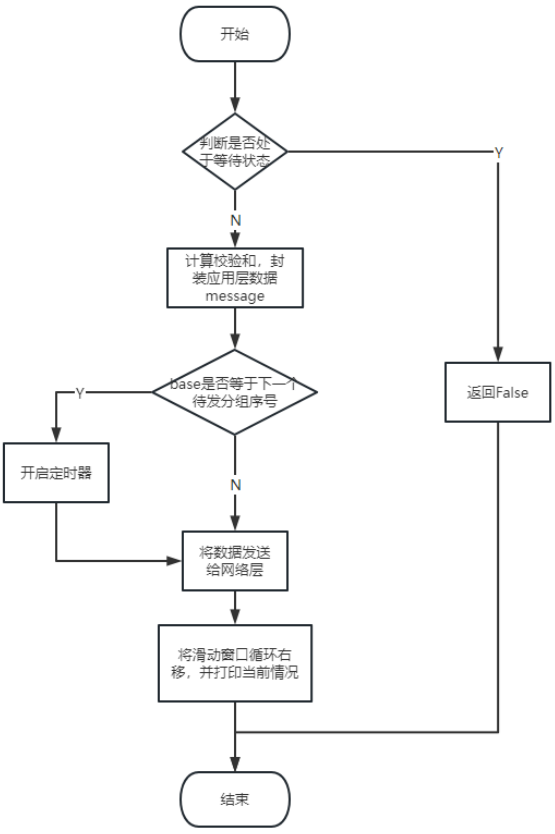

图 1.4: GBNRdtSender::send 函数流程图

2. GBNRdtSender::receive(const Packet& ackPkt)

   接收来自接收方的 ACK 报文 ackPkt，通过计算校验和检查发来的 ack 是否损坏，若损坏则不做处理；更新滑动窗口状态，由于采取累计确认机制，确认 ack 后则将

base 序号直接设置为 (ack+1)% seqsize，若此时 base 序号与下一个待确认分组序号相同，表示接收方已成功接收了所有数据包，则停止计时器，否则重启计时器。函数接收来自接收方的 ACK 报文 ackPkt，没有返回值。函数流程图如图1.5所示

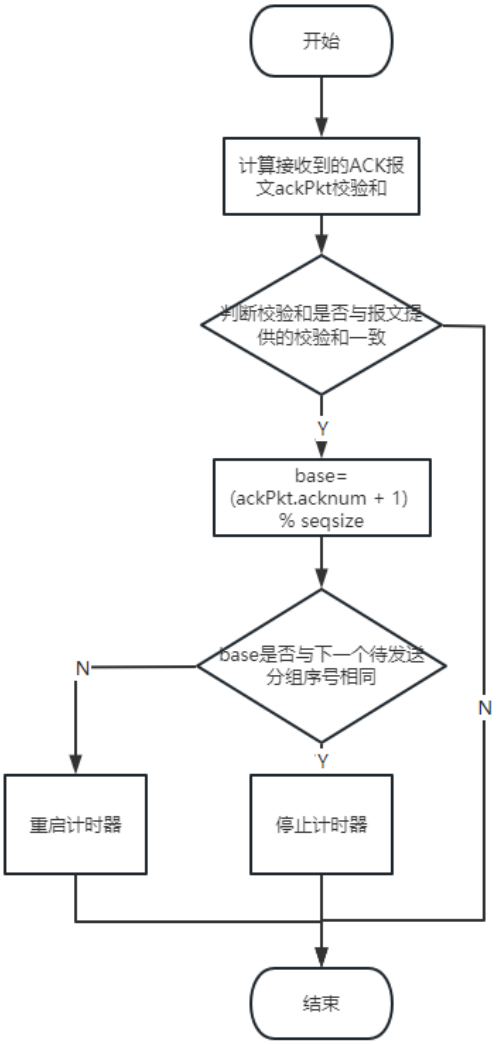

图 1.5: GBNRdtSender::receive 函数流程图

3. GBNRdtSender::timeoutHandler(int seqNum)

   判断下一个待确认分组序号与 base 序号是否相同，若相同则不做处理，否则重启计时器，并将从 base 序号到下一个待确认分组序号前的所有数据包重传。

4. GBNRdtSender::getWaitingState()

   由于 GBN 协议中滑动窗口满则无法接收上层应用数据，故判断等待状态时，计算从 base 序号到下一个待确认分组序号前的距离是否超出滑动窗口大小，并返回判断结果，若窗口满则判断处于等待态，否则判断不处于等待态。

5. GBNRdtReceiver::receive(const Packet& packet) 接收来自发送方的数据报 packer，计算其校验和并判断是否正确，若不正确则重发上次报文；同时检查数据报序号是否是所期待接收的序号，若不是则重发上次报文；若二者均正确则取出数据并递交给应用层，通过网络层发送确认报文给对方。

1.4.2 SR 协议实现

对象设计 将 SR 协议的发送方和接收方设计并实现为两个对象 SRRdtSender 和 SRRdtReceiver。

SRRdtSender 中 seqsize，wndsize，base，nextSeqnum 均与 GBNRdtSender 相同，此处不再赘述。Packet 数组 sendBuf 用于保存已发送且正等待确认的报文；bool 类型数组 bufStatus 用于保存对应序号报文的确认状态。

SRRdtReceiver 中 recvBuf 用于缓存收到且未上传应用层的失序报文，bufStatus 用于保存对应序号报文的确认状态，base 表示接收方窗口的基序号，其他均与 GBNRdtReceiver 相同。

SR 发送方与接收方数据结构定义

```
class SRRdtSender :
public RdtSender
{
private:
        const int seqsize; //序号大小
        const int wndsize; //滑动窗口大小
        Packet* const sendBuf; //发送缓冲区
        bool* const bufStatus;
        int base, nextSeqnum;

};
class SRRdtReceiver :
public RdtReceiver
{
private:
        const int wndsize;
        const int seqsize;
        Packet lastAckPkt;
        Message* const recvBuf;
        bool* const bufStatus;
```

```
21          int base;
22  };
```

关键函数实现

1. bool SRRdtSender::send(const Message& message)

   接收应用层的数据 message，判断缓冲区是否已满，若已满则发送失败，返回 false，否则将其与当前滑动窗口序号、校验和一并封装为传输层数据包 Packet 并发送给网络层；同时由于 SR 协议需要选择重传，故每一个数据包均需为其设置独立的计时器；在发送完成后，更新滑动窗口状态并打印滑动窗口信息，返回 true。

2. void SRRdtSender::receive(const Packet& ackPkt)

   接收来自接收方的 ACK 报文 ackPkt，通过计算校验和检查发来的 ack 是否损坏，若损坏则不做处理；由于采取选择重传机制，确认 ack 后则将 base 序号设置为最后一个未确认的数据包序号，同时更新滑动窗口状态并打印。

3. void SRRdtSender::timeoutHandler(int seqnum)

   由于 SR 协议的选择重传机制，超时时只需重传超时的当前数据包，同时重启其计时器。

4. bool SRRdtSender::getWaitingState()

   判断等待状态时，计算从 base 序号到下一个待确认分组序号前的距离是否超出滑动窗口大小，并返回判断结果，若窗口满则判断处于等待态，否则判断不处于等待态。

5. void SRRdtReceiver::receive(const Packet& packet)

   接收来自发送方的数据报 packer，计算其校验和并判断是否正确，若不正确则重发上次报文；同时检查数据报序号是否是所期待接收的序号（是否在接收方滑动窗口内），若不是则重发上次报文；若二者均正确则将接收缓冲区中所有数据包取出数据并递交给应用层，通过网络层发送确认报文给对方。

1.4.3 TCP 协议实现

对象设计 将 TCP 协议的发送方设计并实现为对象 TCPRdtSender，根据实验要求，TCP 协议的接收方无额外要求，故将其接收方复用为 GBN 协议的接收方。TCPRdtSender 的 int 型成员变量 dupAckNum 用于记录收到冗余 ACK 的基础，满 3 次时即接收到三个冗余 ACK，根据 TCP 的快速重传机制进行快速重传；其他成员变量与 GBNRdtSender 一致。

SR 发送方与接收方数据结构定义

```
class TcpRdtSender :
        public RdtSender
{
private:
        int base; //基序号
        int nextSeqnum; //下一个待发分组序号
        const int wndsize; //滑动窗口大小
        const int seqsize; //序号大小
        Packet* const sendBuf; //发送缓冲区
        int dupAckNum; //收到3个冗余ack快速重传
};
```

关键函数实现

1. void TcpRdtSender::receive(const Packet& ackPkt)

   接收来自接收方的 ACK 报文 ackPkt，通过计算校验和检查发来的 ack 是否损坏，若损坏则不做处理；由于采取快速重传机制，当接收到窗口外的 ACK（冗余 ACK）时，将冗余 ack 计数变量 dupAckNum 更新，若检测到接收到连续三个冗余 ack 则进行快速重传；否则由于采取累计确认机制，确认 ack 后则将 base 序号直接设置为 (ack+1)% seqsize，若此时 base 序号与下一个待确认分组序号相同，表示接收方已成功接收了所有数据包，则停止计时器，否则重启计时器。

2. void TcpRdtSender::timeoutHandler(int seqNum)

   由于 TCP 协议使用快速重传机制，故超时时只需重传最早的数据包并重启计时器。

   根据实验要求，TCP 协议只需额外实现快速重传机制，故 TCPRdtSender 的其余函数内容均与 GBNRdtSender 一致，此处不再赘述。

1.5 系统测试及结果说明

系统测试的硬件环境见1.1，使用实验提供的测试脚本 check_win.bat 对实验结果的正确性进行测试。

1.5.1 GBN 协议测试

通过测试脚本将规定文本内容在模拟网络环境中进行十次随机收发测试并对比发送数据与接收数据的差异，在十次随机测试中收发数据均保持一致，找不到差异，如

图1.6所示，证明了 GBN 协议数据传输的正确性。

```
Test "GBN.exe" 4:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "GBN.exe" 5:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "GBN.exe" 6:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "GBN.exe" 7:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "GBN.exe" 8:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "GBN.exe" 9:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "GBN.exe" 10:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异
```

图 1.6: GBN 脚本测试结果

在控制台输出信息中，GBN 协议的超时重传机制实现结果如图1.7所示；滑动窗口机制实现结果如图1.8所示，可见 GBN 协议正确实现了超时重传机制与滑动窗口机制。

```
发送方正确收到确认: seqnum = -1, acknum = 6, checksum = 12845, ....................
滑动窗口状态: 0 1* 2 ]  3 4 5 6  [ 7

超时重传:: seqnum = 7, acknum = -1, checksum = 56534, PPPPPPPPPPPPPPPPPPP
超时重传:: seqnum = 0, acknum = -1, checksum = 53971, QQQQQQQQQQQQQQQQQQQ
接收方正确收到发送方的报文: seqnum = 7, acknum = -1, checksum = 56534, PPPPPPPPPPPPPPPPPPP
******模拟网络环境******: 向上递交给应用层数据: PPPPPPPPPPPPPPPPPPP
接收方发送确认报文: seqnum = -1, acknum = 7, checksum = 12844, ....................
发送方正确收到确认: seqnum = -1, acknum = 7, checksum = 12844, ....................
滑动窗口状态:  [ 0 1* 2 3 ]  4 5 6 7
```

图 1.7: GBN 协议超时重传展示

1.5.2 SR 协议测试

通过测试脚本将规定文本内容在模拟网络环境中进行十次随机收发测试并对比发送数据与接收数据的差异，在十次随机测试中收发数据均保持一致，找不到差异，如图1.9所示，证明了 SR 协议数据传输的正确性。

```
******模拟网络环境******: 向上递交给应用层数据: QQQQQQQQQQQQQQQQQQQQ
接收方发送确认报文: seqnum = -1, acknum = 0, checksum = 12851, ....................
发送方没有正确收到确认: seqnum = -1, acknum = -999999, checksum = 12851, ....................
超时重传:: seqnum = 0, acknum = -1, checksum = 53971, QQQQQQQQQQQQQQQQQQQQ
接收方没有正确收到发送方的报文,数据校验错误: seqnum = 0, acknum = -1, checksum = 53971, RQQQQQQQQQQQQQQQQQQQ
接收方重新发送上次的确认报文: seqnum = -1, acknum = 0, checksum = 12851, ....................
发送方正确收到确认: seqnum = -1, acknum = 0, checksum = 12851, ....................
滑动窗口状态: 0  [ 1* 2 3 4 ]  5 6 7

发送方发送报文: seqnum = 1, acknum = -1, checksum = 51400, RRRRRRRRRRRRRRRRRRR
发送方发送后窗口: 0  [ 1 2* 3 4 ]  5 6 7

发送方发送报文: seqnum = 2, acknum = -1, checksum = 48829, SSSSSSSSSSSSSSSSSSS
发送方发送后窗口: 0  [ 1 2 3* 4 ]  5 6 7

发送方发送报文: seqnum = 3, acknum = -1, checksum = 46258, TTTTTTTTTTTTTTTTTTT
发送方发送后窗口: 0  [ 1 2 3 4* ]  5 6 7

发送方发送报文: seqnum = 4, acknum = -1, checksum = 43687, UUUUUUUUUUUUUUUUUUU
发送方发送后窗口: 0  [ 1 2 3 4 ]  5* 6 7
```

图 1.8: GBN 协议滑动窗口展示

在控制台输出信息中,SR 协议的超时重传机制与滑动窗口机制实现结果如图1.10所示，可见 SR 协议正确实现了超时选择重传机制与滑动窗口机制。

1.5.3 TCP 协议测试

通过测试脚本将规定文本内容在模拟网络环境中进行十次随机收发测试并对比发送数据与接收数据的差异，在十次随机测试中收发数据均保持一致，找不到差异，如图1.11所示，证明了 TCP 协议数据传输的正确性。

在控制台输出信息中,TCP 协议的超时重传机制与滑动窗口机制实现结果如图1.12所示，可见 TCP 协议正确实现了根据冗余 ACK 快速重传机制与滑动窗口机制。

1.6 其他需要说明的问题

根据实验要求，TCP 协议只需实现快速重传机制，故实验在 TCP 协议中复用了大部分 GBN 协议的代码，仅在发送方的 receive 函数和超时处理函数按 TCP 协议快速重传机制对其进行了实现，其余内容均与 GBN 协议保持一致。

```
Test "SR.exe" 4:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "SR.exe" 5:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "SR.exe" 6:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "SR.exe" 7:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "SR.exe" 8:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "SR.exe" 9:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异

Test "SR.exe" 10:
正在比较文件 input.txt 和 OUTPUT.TXT
FC: 找不到差异
```

图 1.9: SR 协议脚本测试结果

```
接收方窗口移动: 0 1 2  [ 3 4 5 6  ] 7

接收方发送确认报文: seqnum = -1, acknum = 2, checksum = 12849, .....................
发送方正确收到确认: seqnum = -1, acknum = 1, checksum = 12850, .....................
发送方滑动窗口状态: 0 1  [ 2 3 4 |5  ] 6 7

超时重传: seqnum = 2, acknum = -1, checksum = 24414, CCCCCCCCCCCCCCCCCCCC
超时重传: seqnum = 3, acknum = -1, checksum = 21843, DDDDDDDDDDDDDDDDDDDD
超时重传: seqnum = 4, acknum = -1, checksum = 19272, EEEEEEEEEEEEEEEEEEEE
******模拟网络环境******: 向上递交给应用层数据: DDDDDDDDDDDDDDDDDDDD

接收方窗口移动:  ] 0 1 2 3  [ 4 5 6 7

接收方发送确认报文: seqnum = -1, acknum = 3, checksum = 12848, .....................
发送方正确收到确认: seqnum = -1, acknum = 3, checksum = 12848, .....................
发送方滑动窗口状态: 0 1  [ 2 3* 4 |5  ] 6 7

******模拟网络环境******: 向上递交给应用层数据: EEEEEEEEEEEEEEEEEEEE

接收方窗口移动: 0  ] 1 2 3 4  [ 5 6 7

接收方发送确认报文: seqnum = -1, acknum = 4, checksum = 12847, .....................
不是窗口内的分组, 忽略: seqnum = 2, acknum = -1, checksum = 24414, CCCCCCCCCCCCCCCCCCCC
发送方正确收到确认: seqnum = -1, acknum = 4, checksum = 12847, .....................
发送方滑动窗口状态: 0 1  [ 2 3* 4* |5  ] 6 7

发送方正确收到确认: seqnum = -1, acknum = 2, checksum = 12849, .....................
发送方滑动窗口状态: 0  ] 1 2 3 4  [ |5 6 7
```

图 1.10: SR 协议选择重传与滑动窗口展示

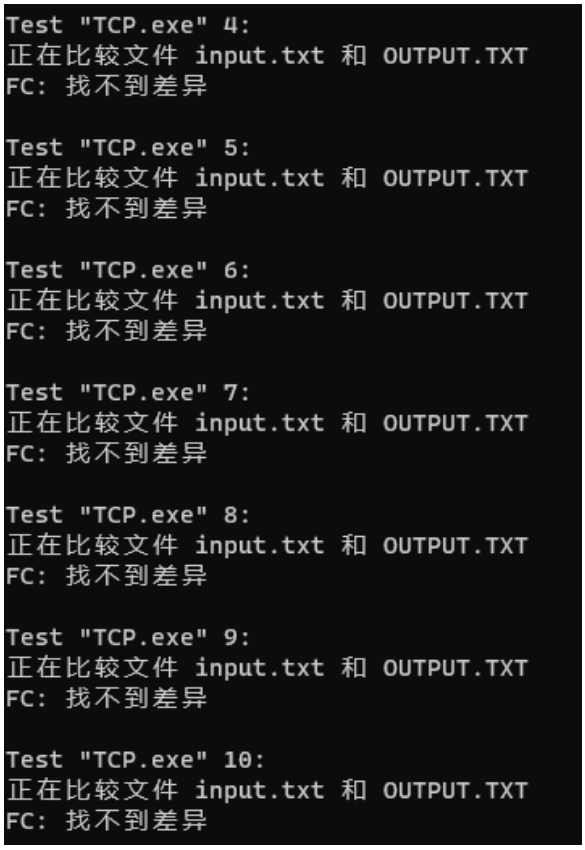

图 1.11: TCP 协议脚本测试结果

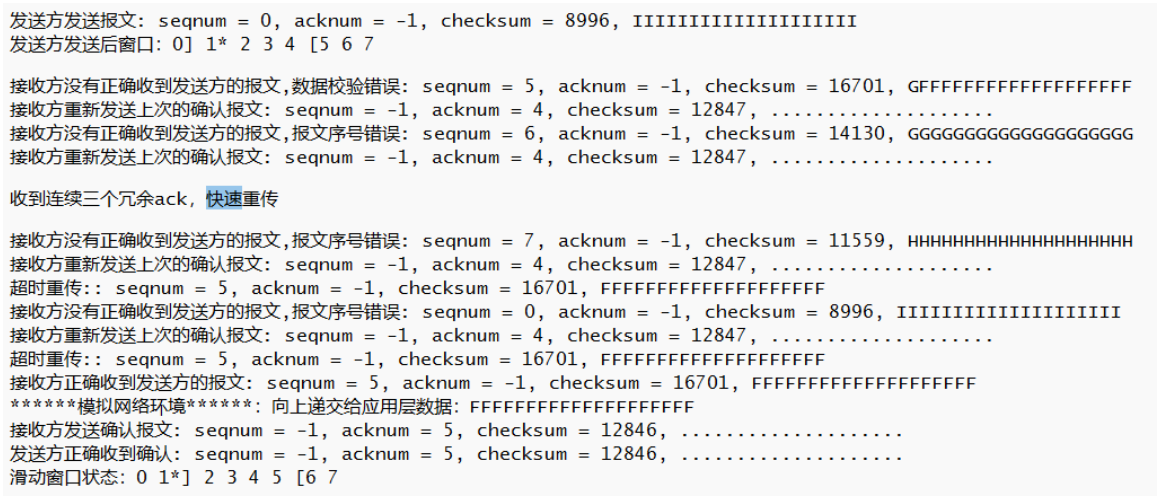

图 1.12: TCP 协议快速重传与滑动窗口展示

2 心得体会与建议

2.1 心得体会

本次实验，我按照实验要求依次实现了 GBN，SR，TCP 三种不同的可靠数据传输协议，并将其与模拟网络环境进行接合，实现了文本数据的模拟收发与检查。

这三种不同的 RDT 协议内容复杂度与实现难度依次递增，我通过将它们一步步实现、调试和完善的过程，发现了许多在理论课程中未曾注意过的问题（如 SR 协议中接收方对数据包的缓存细节），同时通过查阅资料、重新阅读教科书和咨询同学解决了这些问题，深化了对三种协议运行流程的理解。GBN 协议实现较为简单，但它的拥塞控制功能较弱；SR 协议实现相对较为复杂，但它在保证可靠性的同时提供了较好的流量控制功能。而 TCP 协议的快速重传机制在 GBN 协议的基础上进一步提高了拥塞控制能力。本次实验使我深刻地理解了不同的可靠数据传输协议的运行原理，了解了不同机制在数据收发中带来的收发效率与速度上的变化。

这里也感谢计算机网络实验课程组提供的详细的实验说明文档和样例代码，帮助我较为容易地上手了实验，同时提供的模拟网络环境与检测脚本文件也使我十分方便地完成了文本数据的模拟收发与检查。

2.2 建议

希望实验指导书能够更加简明扼要，针对性更强。最好能够首先清晰地描述实验的内容，然后再提供一些额外的参考文档。这样的话，同学们就能够更加轻松地上手实验，而不会觉得一开始就不知道从何做起。至于一些其他的补充内容或者教程文档，最好能够单独写在一个文档中，以便于同学们更好地查阅。本人觉得实验一和实验二的指导文档在这方面做得比较好，但实验三的指导文档就比较难以理解和实际上手实验。

参考文献

[1] James F. Kurose, Keith W. Ross. 计算机网络: 自顶向下方法 (第 7 版) [M]. 机械工业出版社, 2018.