

---

# 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 CS2011      姓名 徐锦慧      学号 U202011675

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
1.1 环境 .....	1
1.2 系统功能需求.....	1
1.3 系统设计 .....	2
1.4 系统实现.....	4
1.5 系统测试及结果说明 .....	8
1.6 其他需要说明的问题.....	15
1.7 参考文献.....	15
心得体会与建议 .....	16
2.1 心得体会 .....	16
2.2 建议 .....	16

---

## 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

处理器：Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz 1.19 GHz

操作系统：Windows 10 专业版

机带 RAM：8.00 GB (7.75 GB 可用)

第三方组件：模拟网络环境 API

开发平台：Visual Studio 2019

开发语言：C/C++

#### 1.1.2 运行平台

处理器：Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz 1.19 GHz

操作系统：Windows 10 专业版

第三方组件：模拟网络环境 API

运行平台：Visual Studio 2019

### 1.2 系统功能需求

1. 实现基于 GBN 的可靠传输协议，分值为 50%。
2. 实现基于 SR 的可靠传输协议，分值为 30%。
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，分值为 20%，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

#### 1.3.1 系统架构

系统分为 GBN、SR、TCP 三部分，每部分均基于模拟网络环境来实现。模拟网络环境模拟了真实的丢包、报文损坏的情况，实现了应用层和网络层，还需实现运输层的 Rdt 协议来协同工作完成数据的可靠传输。模拟网络环境架构和 Rdt 协议之间的关系如图 1.1 所示：

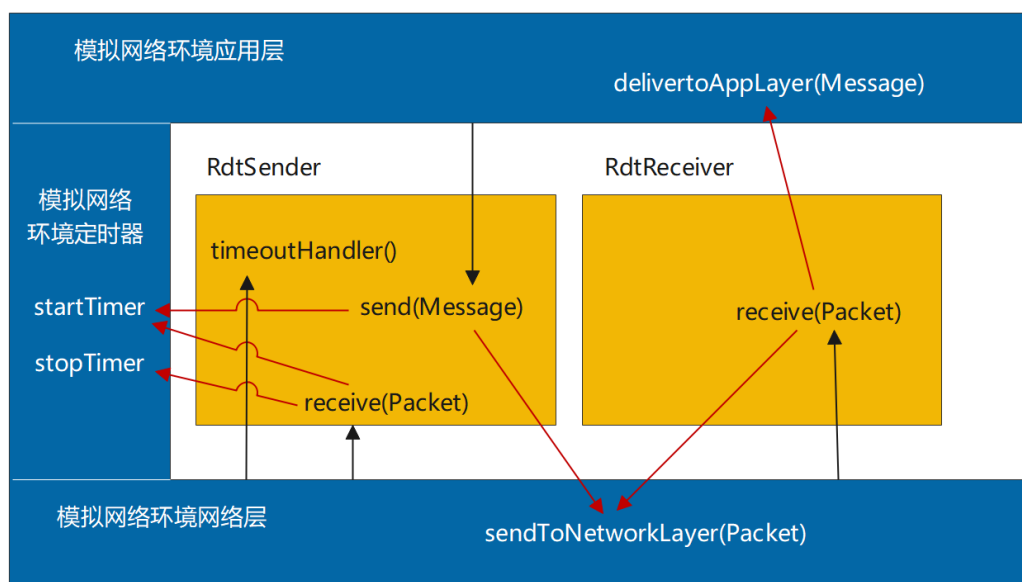


图 1.1 模拟网络环境架构

其中，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（RdtSender）和接收方（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。

模拟网络环境实现了以下函数以供 Rdt 协议调用：deliverToAppLayer(Message)、sendToNetworkLayer(Packet)、startTimer(int seqNum)、stopTimer(int seqNum)。

Rdt 协议的发送方 RdtSender 和接收方 RdtReceiver 则是实验中需要自己实现的功能。其中 RdtSender 必须实现的函数为：send(Message)、receive(Packet)、timeoutHandler(int seqNum)；RdtReceiver 必须实现的函数为：receive(Packet)。

---

### 1.3.2 数据结构设计

系统在 DataStructure.h 头文件里定义了 Message 和 Packet 类，分别表示应用层数据和运输层数据。Message 的结构如下：

```
struct Message {
    char data[Configuration::PAYLOAD_SIZE]; //payload
    Message();
    Message(const Message &msg);
    Message& operator=(const Message &msg);
    ~Message();
    void print();
};
```

Packet 的结构如下：

```
struct Packet {
    int seqnum; //序号
    int acknum; //确认号
    int checksum; //校验和
    char payload[Configuration::PAYLOAD_SIZE]; //payload
    Packet();
    Packet(const Packet &pkt);
    Packet &operator=(const Packet &pkt);
    bool operator==(const Packet &pkt) const;
    ~Packet();
    void print();
};
```

### 1.3.3 函数设计

- bool RdtSender::getWaitingState()

输入：无。

输出：RdtSender 是否处于等待状态。

功能：判断 RdtSender 是否处于等待状态。如果发送方正等待确认或者发送窗口已满，则处于等待状态并返回 true，否则返回 false。

- bool RdtSender::send(const Message& message)

输入：由应用层下发的 message。

输出：是否成功发送 message。

功能：向网络层发送 message。如果发送方处于等待确认状态或发送窗口已满，则拒绝发送 Message 并返回 false；否则先计算校验和，将待发送的包加入窗口队列，再调用模拟网络

---

环境的函数 `sendToNetworkLayer()` 发送 `message` 并更新下一个序号，若发送成功则返回 `true`。

- `void RdtSender::receive(const Packet& ackPkt)`

输入：应用层报文段。

输出：无。

功能：确认接收 **ACK**。首先检查校验和是否正确，若不正确则输出报错信息，否则关闭对应的定时器并滑动窗口，更新基序号。

- `void RdtSender::timeoutHandler(int seqNum)`

输入：`seqNum`，即和该定时器关联的 `Packet` 的序号

输出：无。

功能：处理数据传输时的超时状况。首先关闭定时器，接着重新启动定时器并重新发送窗口的所有待确认数据包。

- `void RdtReceiver::receive(const Packet& packet)`

输入：应用层报文段。

输出：无。

功能：接收发送方送来的报文段。首先检查校验和是否正确，若不正确则输出相应报错信息并重发，否则调用模拟网络环境的 `sendToNetworkLayer`，通过网络层发送确认报文，并更新下一个期待收到的报文序号。

由于 **GBN**、**SR**、**TCP** 协议的原理不同，以上几个函数的实现也有所区别，尤其是 **GBN** 协议和 **SR** 协议，二者的对比如表 1.1 所示：

表 1.1 GBN 协议和 SR 协议对比表

	GBN	SR
确认方式	累积确认	单个确认
定时器	对所有已发送但未确认的分组统一设置一个定时器	对所有已发送但未确认的分组分别设置定时器
超时(n)	重传分组 <b>n</b> 和窗口中所有序号大于 <b>n</b> 的分组	仅重传分组 <b>n</b>
失序分组	丢弃（不缓存）→接收方无缓存	缓存
	重发按序到达的最高序号分组的 <b>ACK</b>	对失序分组进行选择性的确认

## 1.4 系统实现

### 1. GBN 协议实现

在 GBN 协议中，允许发送方发送多个分组而不需等待确认，并且发送方只使用一个定时器，它被当作是最早的已发送但未确认的分组。如果收到一个 ACK，但仍有已发送但未确认的分组，则定时器被重新启动，否则停止该定时器。

我们将基序号（base）定义为最早未确认分组的序号，将下一个序号（nextseqnum）定义为最小的未使用序号，则可以将序号范围分为 4 段：在 $[0, \text{base}-1]$ 段内的序号对应于已发送并被确认的分组； $[\text{base}, \text{nextseqnum}-1]$ 段内对应已经发送但未被确认的分组； $[\text{nextseqnum}, \text{base}+N-1]$ 段内对应那些即将被发送的分组；大于或等于  $\text{base}+N$  的序号则不能使用，直到当前流水线中未被确认的分组得到确认为止。GBN 协议的序号范围如图 1.2 所示：

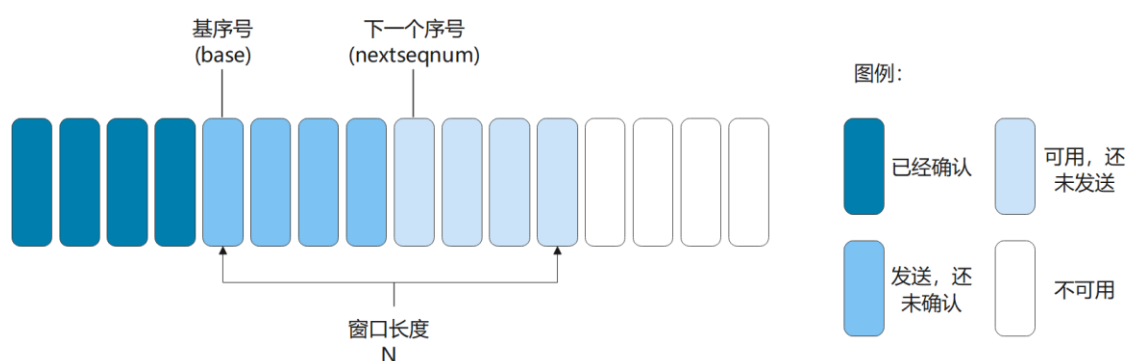


图 1.2 GBN 发送方窗口

在 GBN 协议中，发送方响应的事件有 3 种：

①上层的调用，即 `send()`函数的实现。当上层调用该函数时，发送方首先检查发送窗口是否已满，即是否有  $N$  个已发送但未确认的分组。若窗口已满则拒绝发送，否则产生一个分组并将其加入窗口队列，如果该分组是窗口里的第一个分组就启动计时器，接着通过网络层发送分组并更新下一个序号。

②接收 ACK，即 `receive()`函数的实现。首先检查校验和是否正确，若不正确则输出相应的报错信息并退出，否则对收到的分组采取累计确认：首先以 `base` 为基准关闭定时器，接着滑动窗口，将已成功接收的 ACK 及之前的退出窗口队列，并更新基序号，最后以 `base` 为基准开启定时器。

③超时事件，即 `timeHandler()`函数的实现。当出现超时事件时，发送方重新启动定时器，并重传所有已发送但还未被确认的分组。

GBN 协议发送方的扩展 FSM 描述如图 1.3 所示：

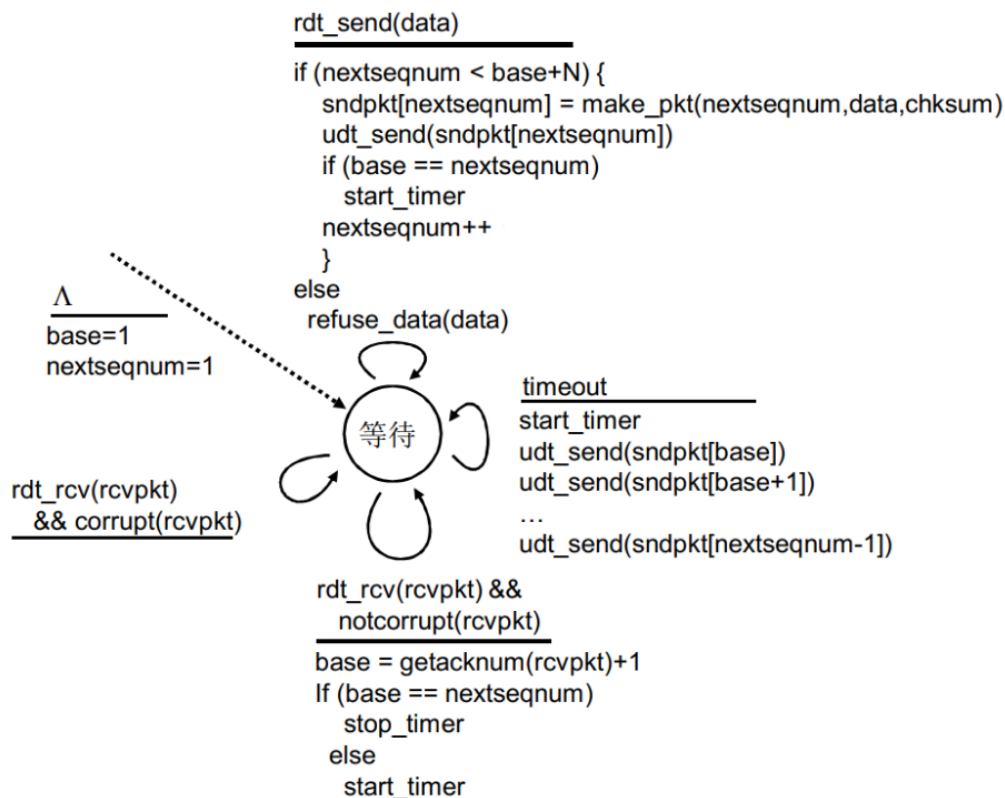


图 1.3 GBN 发送方 FSM

接收方的动作则相对简单。如果一个序号为  $n$  的分组被正确收到且顺序正确，则接收方为分组  $n$  发送 ACK 并将数据交付上层，否则接收方丢弃该分组，并为最近按序接收的分组重新发送 ACK。GBN 协议接收方的扩展 FSM 描述如图 1.4 所示：

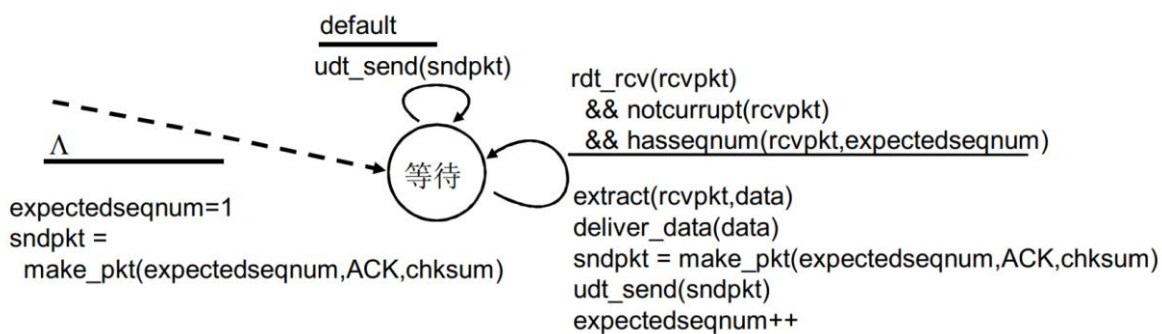


图 1.4 GBN 接收方 FSM

## 2. SR 协议实现

SR 协议为每一个分组都设置了定时器，发送方在发送时仅重传错误的分组，接收方则确认所有正确接收的分组而不管其是否按序，失序的分组将被缓存直到所有丢失分组都被收到。SR 发送方与接收方看到的序号控件如图 1.5 所示：



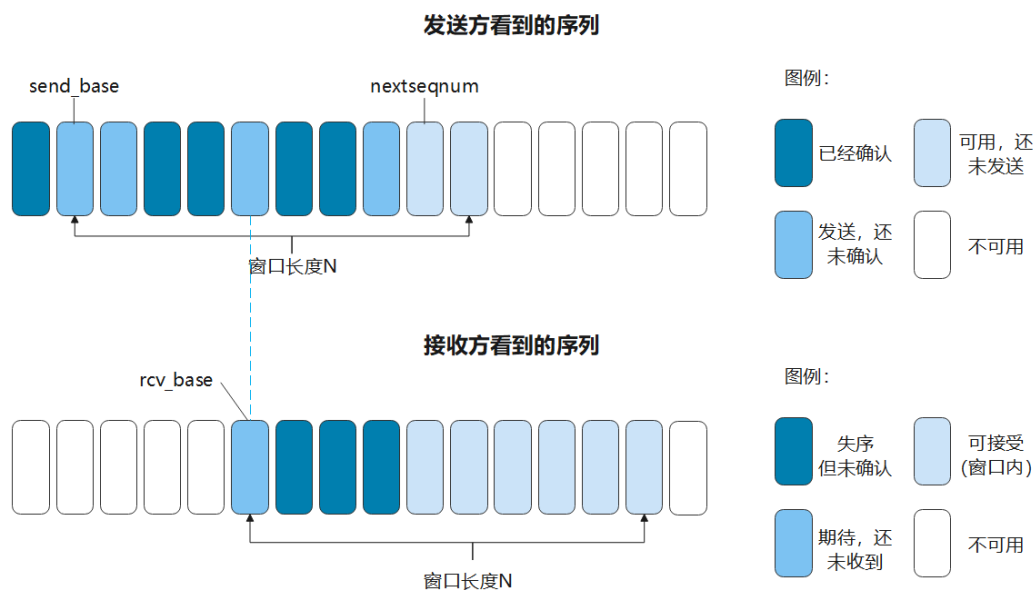


图 1.5 SR 窗口序列

与 GBN 类似, SR 协议发送方响应的事件也有 3 种:

①上层的调用, 即 `send()`函数的实现。从上层接收到数据后, SR 发送方检查下一个可用于该分组的序号, 如果序号位于发送方窗口内, 则将数据打包发送; 否则返回 `false` 并退出。注意此处与 GBN 协议的实现有所区别, 在发送时需要为每一个分组都设置定时器。

②接收 ACK, 即 `receive()`函数的实现。如果收到 ACK 并且分组序号在窗口内, 则 SR 发送方将其标记为已接收; 如果该分组的序号等于 `send_base`, 则窗口基序号向前移动到具有最小序号的未确认分组处。如果窗口移动了并且有序号落在窗口内的未发送分组, 则发送分组。

③超时事件, 即 `timeHandler()`函数的实现。当发生超时事件时, SR 协议重启定时器, 但是仅重传错误分组, 这也与 GBN 的实现不同。

SR 协议的接收方则根据序号负责接收分组。序号在 $[rcv\_base, rcv\_base+N-1]$ 内且被正确接收时, 即收到的分组落在接收方窗口内, 如果该分组以前没收到过则缓存该分组, 如果该分组的序号等于接收窗口的基序号 (`rcv_base`), 则将该分组及以前缓存的序号连续的分组交付给上层; 序号在 $[rcv\_base-N, rcv\_base-1]$ 内且被正确接收时, 则必须产生一个 ACK, 即使该分组是接收方以前已经确认过的分组; 序号为其他情况时则忽略分组。

### 3. TCP 协议实现

TCP 协议基于 GBN 协议实现, 其中接收方的实现和发送方 `send()`函数的实现均与 GBN 协议一致, 且均只设置一个定时器, 只有发送方的 `receive()`函数和 `timehandler()`函数有所不同:

在发送方的 `receive()`函数中, 除了计算检查, 还需要对冗余 ACK 计数, 当收到 3 个冗

余 ACK 时，发送方需要启动快速重传机制，重传最早发送且未被确认的报文段。超时重传部分的伪代码如下：

```
event: ACK received, with ACK field value of y
    if (y > SendBase) {
        SendBase = y
        if (there are currently not-yet-acknowledged segments)
            start timer
    }
    else {
        increment count of dup ACKs received for y
        if (count of dup ACKs received for y = 3)
            resend segment with sequence number y
    }
```

在发送方的 `timeoutHandler()` 函数中，当发生超时事件时，只需要重启定时器并重传最早发送且未被确认的报文段，而不是像 GBN 协议那样重传窗口内的所有待确认数据报。TCP 协议发送方的扩展 FSM 如图 1.6 所示：

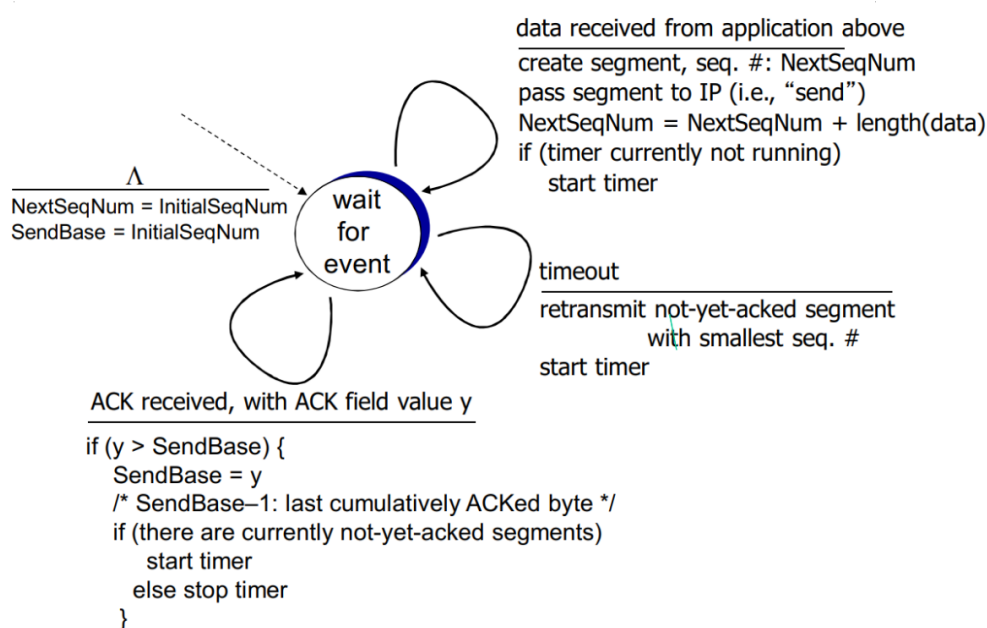


图 1.6 TCP 发送方 FSM

## 1.5 系统测试及结果说明

### 1.5.1 测试环境

处理器：Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz 1.19 GHz

操作系统：Windows 10 专业版

第三方组件：模拟网络环境 API

运行平台：Visual Studio 2019

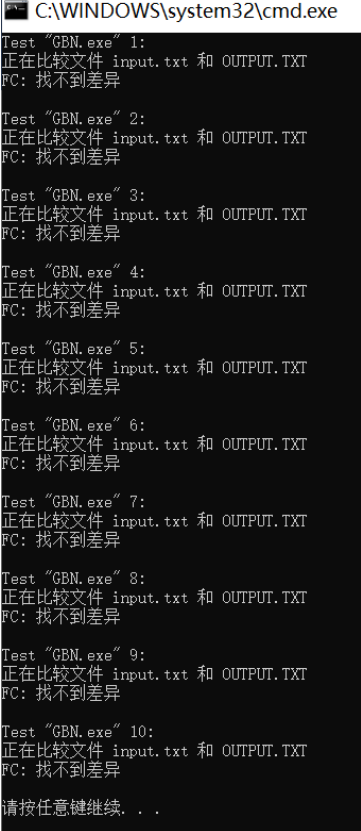
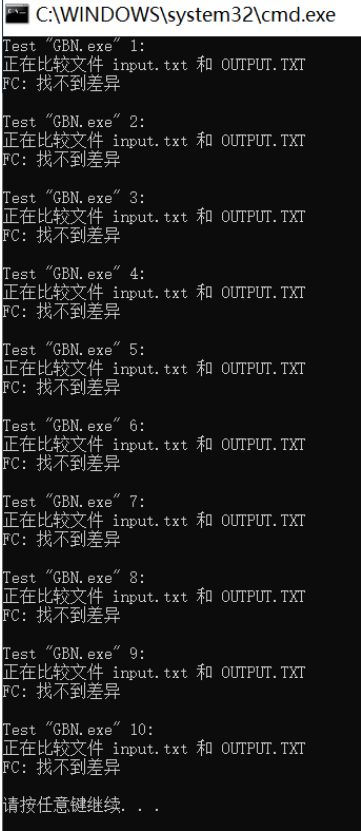
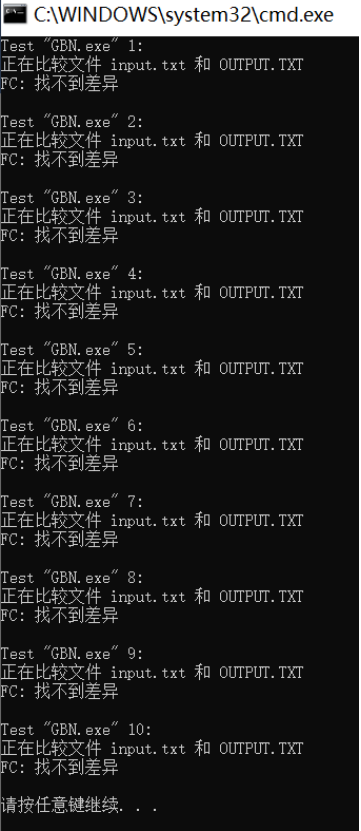
### 1.5.1 测试结果及分析

#### 1. GBN 协议

##### 1) GBN 协议实现的正确性

使用 3 个测试样例，通过检查脚本连续运行 10 次程序检查输入文件 input.txt 和输出文件 output.txt 的内容一致性，测试结果如表 1.2 所示：

表 1.2 GBN 脚本测试结果

用例序号	用例 1	用例 2	用例 3
测试截图			

通过以上多个实例测试结果可以得出，多次运行程序的输入文件、输出文件内容相同，即 GBN 协议能够正确实现。

##### 2) 滑动窗口移动的正确性

在本次实验每次移动滑动窗口时在控制台输出滑动窗口的内容，并通过检查脚本将控制台的输出重定向到文件 result.txt 中，以此检查滑动窗口的移动是否正确。其中 rcvdAckNum 表示

收到的 ACK 确认序号，baseSeqNum 表示窗口的基序号，nextSeqNum 表示下一个期望收到的序号，frontSlidingWindow 表示滑动前的窗口队列，rearSlidingWindow 表示滑动后的窗口序号。

运行测试用例后观察 result.txt 文件，可以看到，当 rcvdAckNum=baseSeqNum=4 时，窗口的基序号为 4 且发送方收到的 ACK 序号也为 4，窗口从[4 5 6 7]变成[5 6 7 0]，符合测试要求。即当收到的 ACK 序号为窗口基序号时，窗口能正确向前滑动一格，如图 1.7 所示：

```
=====
发送方定时器时间到，重发窗口报文: seqnum = 4, acknum = -1, checksum = 19272, EEEEEEEEEEEEEEEEEEE
接收方正确收到发送方的报文: seqnum = 4, acknum = -1, checksum = 19272, EEEEEEEEEEEEEEEEEEE
*****模拟网络环境*****: 向上递交给应用层数据: EEEEEEEEEEEEEEEEEEE
接收方发送确认报文: seqnum = -1, acknum = 4, checksum = 12847, .....
rcvdAckNum: 4
baseSeqNum: 4
nextSeqNum: 5
frontSlidingWindow: [ 4 5 6 7 ]
发送方正确收到确认: seqnum = -1, acknum = 4, checksum = 12847, .....
rearSlidingWindow: [ 5 6 7 0 ]
=====
```

图 1.7 GBN 窗口移动结果 a

当 rcvdAckNum=3 而 baseSeqNum=2 时，窗口基序号为 2，发送方收到的 ACK 序号为 4，窗口从[2 3 4 5]变成[4 5 6 7]，符合测试要求。即当收到的 ACK 序号和窗口基序号不一致时，窗口也能正确向前滑动，如图 1.8 所示：

```
=====
接收方正确收到发送方的报文: seqnum = 3, acknum = -1, checksum = 21843, DDDDDDDDDDDDDDDDDDDDD
*****模拟网络环境*****: 向上递交给应用层数据: DDDDDDDDDDDDDDDDDDDDD
接收方发送确认报文: seqnum = -1, acknum = 3, checksum = 12848, .....
发送方没有正确收到该报文确认,数据校验错误: seqnum = -1, acknum = 2, checksum = 12849, /.....
接收方没有正确收到发送方的报文,数据校验错误: seqnum = 4, acknum = -1, checksum = 19272, FEEEEEEEEEEEEEEEEEE
接收方重新发送上次的确认报文: seqnum = -1, acknum = 3, checksum = 12848, .....
rcvdAckNum: 3
baseSeqNum: 2
nextSeqNum: 5
frontSlidingWindow: [ 2 3 4 5 ]
发送方正确收到确认: seqnum = -1, acknum = 3, checksum = 12848, .....
rearSlidingWindow: [ 4 5 6 7 ]
=====
```

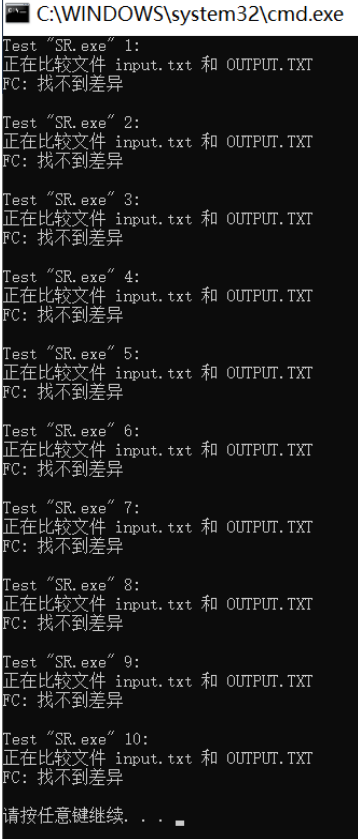
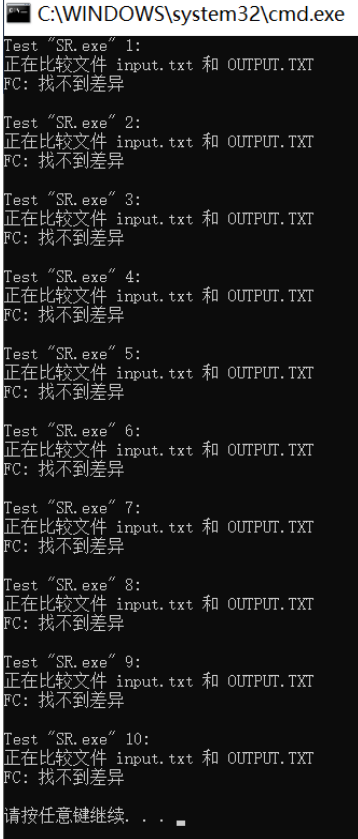
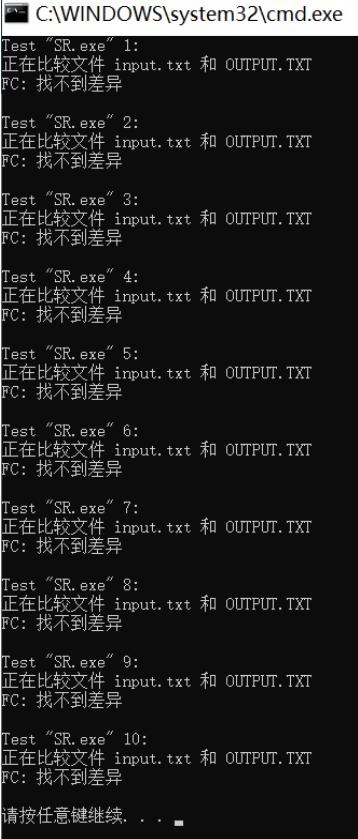
图 1.8 GBN 窗口移动结果 b

## 2. SR 协议

### 1) SR 协议实现的正确性

和 GBN 相似，使用 3 个测试样例，通过检查脚本连续运行 10 次程序检查输入文件 input.txt 和输出文件 output.txt 的内容一致性，测试结果如表 1.3 所示：

表 1.3 SR 脚本测试结果

用例序号	用例 1	用例 2	用例 3
测试 截图			

通过以上多个实例测试结果可以得出，多次运行程序的输入文件、输出文件内容相同，即 SR 协议能够正确实现。

## 2) 滑动窗口移动的正确性

与 GBN 协议类似,实验通过检查脚本将控制台输出的滑动窗口内容重定向到文件 result.txt 中，以此检查滑动窗口的移动是否正确。其中 rcvdAckNum 表示收到的 ACK 确认序号，baseSeqNum 表示窗口的基序号，nextSeqNum 表示下一个期望收到的序号，frontSlidingWindow 表示滑动前的窗口队列，rearSlidingWindow 表示滑动后的窗口序号。序号后的“N”表示还未收到该分组的正确信息，“Y”表示已正确收到该分组的信息。

运行测试用例后观察 result.txt 文件，可以看到，当 rcvdAckNum=baseSeqNum=0 时，发送方窗口的基序号为 0 且收到的 ACK 序号也为 0，发送方窗口从[0Y 1 2 3]变成[1 2 3 4]，符合测试要求；当系统继续运行，接收方正确收到序号为 2 的窗口后，窗口序列仍为[1N 2Y 3N 4N]，意味着接收方将缓存乱序到来的分组，并保持窗口队列不移动，符合测试要求。即当收到的 ACK 序号为窗口基序号时，窗口能正确向前移动，如图 1.9 和图 1.10 所示：

```

=====
接收方发送确认报文: seqnum = -1, acknum = 0, checksum = 12851, .....
发送方窗口详情:
rcvdAckNum: 0
baseSeqNum: 0
nextSeqNum: 1
frontSlidingWindow: [ 0Y 1 2 3 ]
发送方正确收到确认: seqnum = -1, acknum = 0, checksum = 12851, .....
rearSlidingWindow: [ 1 2 3 4 ]
=====

```

图 1.9 SR 发送方窗口移动结果 a

```

=====
发送方发送报文: seqnum = 1, acknum = -1, checksum = 26985, BBBBBBBBBBBBBBBBBBBB
发送方发送报文: seqnum = 2, acknum = -1, checksum = 24414, CCCCCCCCCCCCCCCCCCCC
发送方发送报文: seqnum = 3, acknum = -1, checksum = 21843, DDDDDDDDDDDDDDDDDDDDDD
发送方发送报文: seqnum = 4, acknum = -1, checksum = 19272, EEEEEEEEEEEEEEEEEEEE
接收方没有正确收到发送方的报文,数据校验错误: seqnum = 1, acknum = -1, checksum = 26985, CBBBBBBBBBBBBBBBBBBBBB
接收方窗口详情:
frontSlidingWindow: [ 1N 2Y 3N 4N ]
接收方正确收到发送方的报文: seqnum = 2, acknum = -1, checksum = 24414, CCCCCCCCCCCCCCCCCCCC
rearSlidingWindow: [ 1N 2Y 3N 4N ]
=====

```

图 1.10 SR 接收方窗口移动结果 a

当 rcvdAckNum=4 而 baseSeqNum=1 时，发送方窗口基序号为 1，收到的 ACK 序号为 4，说明 1 号分组的 ACK 未收到而 4 号分组的 ACK 已收到，发送方窗口仍为[1N 2N 3Y 4Y]，符合测试要求，如图 1.11 所示：

```

=====
接收方发送确认报文: seqnum = -1, acknum = 4, checksum = 12847, .....
发送方窗口详情:
rcvdAckNum: 4
baseSeqNum: 1
nextSeqNum: 5
frontSlidingWindow: [ 1N 2N 3Y 4Y ]
发送方正确收到确认: seqnum = -1, acknum = 4, checksum = 12847, .....
rearSlidingWindow: [ 1N 2N 3Y 4Y ]
=====

```

图 1.11 SR 发送方窗口移动结果 b

```

=====
发送方定时器时间到，重发上次发送的报文: seqnum = 1, acknum = -1, checksum = 26985, BBBBBBBBBBBBBBBBBBBB
发送方定时器时间到，重发上次发送的报文: seqnum = 2, acknum = -1, checksum = 24414, CCCCCCCCCCCCCCCCCCCC
接收方窗口详情:
frontSlidingWindow: [ 1Y 2Y 3Y 4Y ]
接收方正确收到发送方的报文: seqnum = 1, acknum = -1, checksum = 26985, BBBBBBBBBBBBBBBBBBBB
*****模拟网络环境*****: 向上递交给应用层数据: BBBBBBBBBBBBBBBBBBBB
*****模拟网络环境*****: 向上递交给应用层数据: CCCCCCCCCCCCCCCCCCCC
*****模拟网络环境*****: 向上递交给应用层数据: DDDDDDDDDDDDDDDDDDDDDD
*****模拟网络环境*****: 向上递交给应用层数据: EEEEEEEEEEEEEEEEEEEE
rearSlidingWindow: [ 5N 6N 7N 0N ]
=====

```

图 1.12 SR 接收方窗口移动结果 b



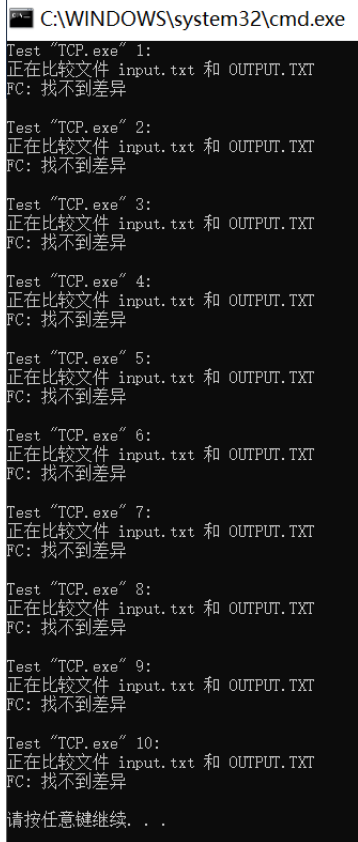
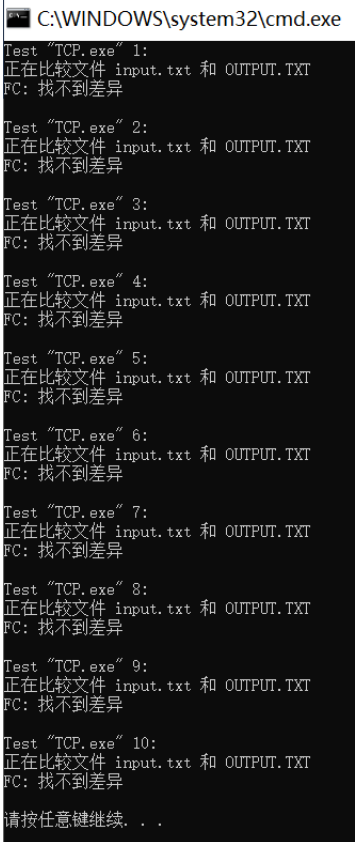
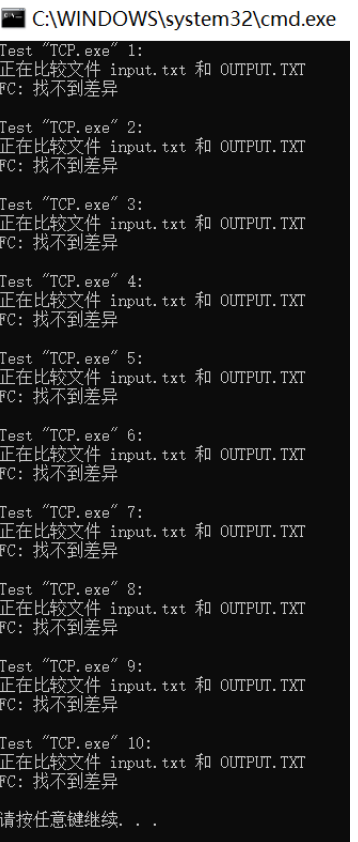
系统继续运行，当接收方正确收到 1 号分组和 2 号分组后，接收方的窗口变为[1Y 2Y 3Y 4Y]，接着接收方正确向上层递交这些分组，窗口向前移动变为[5N 6N 7N 0N]，符合测试要求。即当收到的 ACK 序号和窗口基序号不一致时，窗口也能正确判断队列内序号状态并向前移动，如图 1.12 所示。

### 3. TCP 协议

#### 1) TCP 协议实现的正确性

和 GBN 相似，使用 3 个测试样例，通过检查脚本连续运行 10 次程序检查输入文件 input.txt 和输出文件 output.txt 的内容一致性，测试结果如表 1.4 所示：

表 1.4 TCP 脚本测试结果

用例序号	用例 1	用例 2	用例 3
测试截图			

通过以上多个实例测试结果可以得出，多次运行程序的输入文件、输出文件内容相同，即 TCP 协议能够正确实现。

#### 2) 滑动窗口移动的正确性

与 GBN 协议类似，实验通过检查脚本将控制台输出的滑动窗口内容重定向到文件 result.txt 中，以此检查滑动窗口的移动是否正确。其中 rcvdAckNum 表示收到的 ACK 确认序号，

baseSeqNum 表示窗口的基序号, nextSeqNum 表示下一个期望收到的序号, frontSlidingWindow 表示滑动前的窗口队列, rearSlidingWindow 表示滑动后的窗口序号。

运行测试用例后观察 result.txt 文件, 可以看到, 当 rcvdAckNum=baseSeqNum=3 时, 发送方窗口的基序号为 3 且收到的 ACK 序号也为 3, 窗口从[3 4 5 6]变成[4 5 6 7], 符合测试要求。即当收到的 ACK 序号为窗口基序号时, 窗口能正确向前滑动一格, 如图 1.13 所示:

```
=====
发送方定时器时间到, 重发最早发送且未被确认的报文段: seqnum = 3, acknum = -1, checksum = 21843, DDDDDDDDDDDDDDDDDDDDDDD
接收方正确收到发送方的报文: seqnum = 3, acknum = -1, checksum = 21843, DDDDDDDDDDDDDDDDDDDDDDD
*****模拟网络环境*****: 向上递交给应用层数据: DDDDDDDDDDDDDDDDDDDDDDD
接收方发送确认报文: seqnum = -1, acknum = 3, checksum = 12848, .....
rcvdAckNum: 3
baseSeqNum: 3
nextSeqNum: 5
frontSlidingWindow: [ 3 4 5 6 ]
发送方正确收到确认: seqnum = -1, acknum = 3, checksum = 12848, .....
rearSlidingWindow: [ 4 5 6 7 ]
=====
```

图 1.13 TCP 窗口移动结果 a

当 rcvdAckNum=0 而 baseSeqNum=7 时, 窗口基序号为 7, 发送方收到的 ACK 序号为 0 (这里的 0 序号指下一循环中的第一个序号), 窗口从[7 0 1 2]变成[1 2 3 4], 即实行累积确认, 符合测试要求。即当收到的 ACK 序号和窗口基序号不一致时, 窗口也能正确向前滑动, 如图 1.14 所示:

```
=====
接收方正确收到发送方的报文: seqnum = 7, acknum = -1, checksum = 35974, XXXXXXXXXXXXXXXXXXXXXXXX
*****模拟网络环境*****: 向上递交给应用层数据: XXXXXXXXXXXXXXXXXXXXXXXX
接收方发送确认报文: seqnum = -1, acknum = 7, checksum = 12844, .....
发送方没有正确收到该报文确认, 数据校验错误: seqnum = -1, acknum = 7, checksum = 12844, /.....
接收方正确收到发送方的报文: seqnum = 0, acknum = -1, checksum = 33411, YYYYYYYYYYYYYYYYYYYY
*****模拟网络环境*****: 向上递交给应用层数据: YYYYYYYYYYYYYYYYYYYY
接收方发送确认报文: seqnum = -1, acknum = 0, checksum = 12851, .....
rcvdAckNum: 0
baseSeqNum: 7
nextSeqNum: 1
frontSlidingWindow: [ 7 0 1 2 ]
发送方正确收到确认: seqnum = -1, acknum = 0, checksum = 12851, .....
rearSlidingWindow: [ 1 2 3 4 ]
=====
```

图 1.14 TCP 窗口移动结果 b

### 3) 快速重传的正确性

当发送方收到 3 次冗余的 acknum 时, 将启动快速重传机制, 并重传最早发送且未被确认的报文段。如图 1.15 所示, 发送方最早发送且未被确认的报文段序号为 1, 当发送方连续收到 3 个冗余的序号为 0 的 acknum 时, 便会启动快速重传机制, 重传序号为 1 的报文段, 即 TCP



协议快速重传的功能可以正确实现。

```
接收方没有正确收到发送方的报文,报文序号不对: seqnum = 2, acknum = -1, checksum = 33409, YYYYYYYYYYYYYYYYYY
接收方重新发送上次的确认报文: seqnum = -1, acknum = 0, checksum = 12851, .....
接收方没有正确收到发送方的报文,报文序号不对: seqnum = 3, acknum = -1, checksum = 30838, ZZZZZZZZZZZZZZZZZZZ
接收方重新发送上次的确认报文: seqnum = -1, acknum = 0, checksum = 12851, .....
发送方已正确收到过该报文确认: seqnum = -1, acknum = 0, checksum = 12851, .....
发送方已正确收到过该报文确认: seqnum = -1, acknum = 0, checksum = 12851, .....
接收方没有正确收到发送方的报文,报文序号不对: seqnum = 4, acknum = -1, checksum = 29552, AAAAAAAAAAAAAAAAAA
接收方重新发送上次的确认报文: seqnum = -1, acknum = 0, checksum = 12851, .....
发送方定时器时间到, 重发最早发送且没被确认的报文段: seqnum = 1, acknum = -1, checksum = 35980, XXXXXXXXXXXXXXXXXXXX
发送方已正确收到过该报文确认: seqnum = -1, acknum = 0, checksum = 12851, .....
发送方启动快速重传机制, 重传最早发送且没被确认的报文段: seqnum = 1, acknum = -1, checksum = 35980, XXXXXXXXXXXXXXXXXXXX
接收方正确收到发送方的报文: seqnum = 1, acknum = -1, checksum = 35980, XXXXXXXXXXXXXXXXXXXX
*****模拟网络环境*****: 向上递交给应用层数据: XXXXXXXXXXXXXXXXXXXX
接收方发送确认报文: seqnum = -1, acknum = 1, checksum = 12850, .....
```

图 1.15 TCP 快速重传结果 b

## 1.6 其他需要说明的问题

在实现具体的可靠数据传输协议时, 需要考虑序号空间和窗口大小的问题。需要注意窗口大小和编码报文序号的二进制位数之间的关系, 在判断滑动窗口的位置和收到的确认报文序号关系时, 不能简单地直接比较大小关系。本次实验中, 编码报文序号的二进制位数设为 3, 窗口大小设为 4。

另外需要注意的是定时器的問題, 关闭和启动定时器时一定要注意序号; 重新启动一个定时器前一定要先关闭该定时器。在本次实验中, 大部分报错原因都是计时器设置有误。

最后, 由于在模拟网络环境中丢包和报文损坏都是利用随机事件产生的, 需要多次运行, 每次运行后比较二个文件内容是否相同, 这样才能确保协议的正确性。因此, 系统测试中采用了检查脚本, 并运行了多个测试用例来保证结果的正确性。

## 1.7 参考文献

[1]模块二 可靠数据传输协议设计(修订版 V2020).pdf

[2]James F. Kurose, Keith W. Ross, 库罗斯, 等. 计算机网络: 自顶向下方法(第四版影印版)[M]. 高等教育出版社, 2009.

[3]库罗斯, 罗斯. 计算机网络: 自顶向下方法(原书第 7 版)[M]. 机械工业出版社, 2018.

---

## 心得体会与建议

### 2.1 心得体会

在计算机网络实验课中，我认识了应用层、运输层、网络层等的基本概念和常用协议，并在实验中理解了它们的工作原理。通过这门课程，我对计算机网络有了更深入的理解，并且也培养了自己的实验能力。

在实验一中，通过实现一个完整的 socket 编程流程，我对客户端浏览器服务模式、对整个应用层都有了更好的掌握。由于实验一与日常生活中的使用场景类似，也让我体验到了将理论应用于实践的乐趣。然而，实验过程中也遇到了一些困难，由于初次接触 socket 编程而看不懂任务书，导致刚做实验时一头雾水。好在查阅了大量资料、请教老师同学后终于明白了 socket 编程的原理。

实验二则是我印象最为深刻的实验。通过实现 GBN、SR、TCP 协议，我对可靠数据传输协议有了更深层的认识，并且也反向巩固了课本上的理论知识。通过查看不同协议的滑动窗口输出，我对它们的工作特点和相互间的区别更加记忆深刻，也更能体会到 SR 协议选择重传的优点和 TCP 协议快速重传的便捷之处。在实验中也遇到了一些难题，因为定时器的设置不当而调试了很久，这也让我对定时器这部分的知识印象格外深刻。

在实验三中，我学会了 Cisco Packet Tracer 6.0 仿真软件的使用，通过为下属区域分配 IP，深刻地理解了网络层的 IP 协议运作原理，由于设计虚拟局域网的设置，同样也加深了链路层方面的知识。并且在实验中我们作为组网者切实参与到网络的搭建中，也更能体会到网关、虚拟局域网、路由器等在现实生活中的作用。

总的来说，这门课程内容丰富，涉及计算机网络的多个层次；生动有趣，与实际生活紧密相关；指导详细，参考资料丰富全面。在此也很感谢辛苦编写指导手册、积极答疑解惑的课程组老师们！

### 2.2 建议

课程组给出的指导资料很详细，希望老师们可以在实验二指导手册中再添加一下检查脚本

---

的使用说明，并适当规范滑动窗口的输出结果，这样可以让同学们更清楚检查的形式，也可以加快检查进度。

另外，希望实验三的指导手册中可以添加无线上网功能、选路协议配置等的示例说明，实验时可以更容易上手。