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 CS2009 姓名 raindaydream 学号

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

1. 操作系统：Windows10；
2. 处理器：Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz；
3. 内存：8.00 GB (7.83 GB 可用)；
4. 开发平台：Microsoft Visual Studio Community 2019；
5. 依赖库：老师所提供的模拟网络环境 netsimlib.lib。

1.1.2 运行平台

1. 操作系统：Windows10；
2. 处理器：Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz；
3. 内存：8.00 GB (7.83 GB 可用)；
4. 运行平台：Microsoft Visual Studio Community 2019；
5. 依赖库：老师所提供模拟网络环境 netsimlib.lib。

1.2 系统功能需求

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，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 20%，要求：
   （1）报文段格式、接收方缓冲区大小和 GBN 协议一样保持不变；
   （2）报文段序号按照报文段为单位进行编号；
   （3）单一的超时计时器，不需要估算 RTT 动态调整定时器 Timeout 参数；
   （4）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；
   （5）确认号为收到的最后一个报文段序号；
   （6）不考虑流量控制、拥塞控制。

1.3 系统设计

1.3.1 GBN 协议

GBN 协议按序发送报文序列，每一个发送的报文都有一个序列号，发送的报文最大数量即为滑动窗口大小，如果超过窗口大小将进入等待状态；在接收方，接收方并未设置报文缓存，而是采用累计确认的方式，只接受当前应该接收的报文（采用报文序列号确认），如果报文序列号不正确，以此序号开始的报文将全部重新发送。因此需要定义序列号范围、滑动窗口大小和发送方与接收方的数据结构。

因此数据结构定义为：

①全局信息（定义在 DataStructure.h 中）：

```
#define GBN_seqnum 8//序列号范围
#define GBN_Winsize 4//滑动窗口大小
```

②发送方 sender

```
class GBNRdtSender :public RdtSender
{
private:
    int nextSeqNum;// 下一个发送序号
    int base; //基序号
    bool waitingState; //是否处于等待Ack的状态
    Packet packetWaitingAck[GBN_seqnum]; //报文序号为0~7，储存所有以发送待确认报文
public:
    bool getWaitingState();
    bool send(const Message& message); //发送应用层下来的Message，由NetworkServiceSimulator调用,如果发送方成功地将Message发送到网络层，返回true;如果因为发送方处于等待正确确认状态而拒绝发送Message，则返回false
    void receive(const Packet& ackPkt); //接受确认Ack，将被NetworkServiceSimulator调用
    void timeoutHandler(int seqNum);//Timeout handler，将被NetworkServiceSimulator调用
public:
    GBNRdtSender();
    virtual ~GBNRdtSender();
};
```

③接收方 receiver

```
class GBNRdtReceiver :public RdtReceiver
{
private:
    int expectSequenceNumberRcvd;  // 期待收到的下一个报文序号
    Packet lastAckPkt;                    //上次发送的确认报文
public:
    GBNRdtReceiver();
```

```
    virtual ~GBNRdtReceiver();
public:
    void receive(const Packet& packet);//接收报文，将被 NetworkService 调用
};
```

1.3.2 SR 协议

SR 协议相比于 GBN 协议来说，只重传在接收方出错的分组而避免了不必要的重传。在发送方，SR 协议与 GBN 协议相同，按照报文需要依次发送报文序列；在接收方，接收方会设置一个数组，用来缓存已经发送成功的报文序列，这些报文序列会一直缓存直到向上层传送的报文序号更新为该报文的报文序号。

因此所需要的数据结构为：

①全局信息（定义在 DataStructure.h 中）：

```
#define SR_seqnum 8//序列号范围
#define SR_Winsize 4//滑动窗口大小
```

②发送方 sender

```
class SRRdtSender :public RdtSender
{
private:
    bool waitingstate;
    int sendbase;
    int nextseqnum;  // 下一个发送序号
    Packet packetWaitingAck[SR_seqnum]; //已发送并等待 Ack 的数据包
    bool ACK[SR_seqnum];   //储存已收到的 ACK 序号，true 表示对应报文确认信号已收到
public:
    bool getWaitingState();
    bool send(const Message& message); // 发送应用层下来的 Message，由 NetworkServiceSimulator 调用,如果发送方成功地将 Message 发送到网络层，返回 true;如果因为发送方处于等待正确确认状态而拒绝发送 Message，则返回 false
    void receive(const Packet& ackPkt); //接受确认 Ack，将被 NetworkServiceSimulator 调用
    void timeoutHandler(int seqNum);//Timeout handler，将被 NetworkServiceSimulator 调用
public:
    SRRdtSender();
    virtual ~SRRdtSender();
};
```

③接收方 receiver

```
class SRRdtReceiver :public RdtReceiver
```

```
{
private:
    int rcvbase; //接收方窗口的基序号
    bool rcvseq[SR_seqnum];   //记录是否已缓存对于下标的报文
    Packet lastAckPkt;                  //发送给发送方的确认报文
    Message rcvmsg[SR_seqnum];                  // 本地缓存的报文
public:
    SRRdtReceiver();
    virtual ~SRRdtReceiver();
public:
    void receive(const Packet& packet);    //接收报文，将被 NetworkService 调用
};
```

1.3.3 TCP 协议

本实验要求 TCP 在 GBN 的基础上实现，而 TCP 需要增加的功能为快速重传和超时重传，在这里需要增加一个冗余序号记录需要重传的序号，本实验中其他方面与 GBN 基本相同。

因此数据结构定义为：

①全局信息（定义在 DataStructure.h 中）：

```
#define TCP_Winsize 8//滑动窗口大小
#define TCP_seqnum 16//序列号范围
```

②发送方 sender

```
class TCPRdtSender :public RdtSender
{
private:
    int sendbase;// 当前还未确认的最小的segment序号
    int nextseqnum;     //已发送到的序号
    int y;   //收到的冗余ACK序号
    int num;     //y收到的次数
    bool waitingState;                  // 是否处于等待Ack的状态
    Packet packetWaitingAck[TCP_seqnum];        //发送方缓存
public:
    bool getWaitingState();
    bool send(const Message& message);  //发送应用层下来的Message，由NetworkServiceSimulator调用,如果发送方成功地将Message发送到网络层，返回true;如果因为发送方处于等待正确确认状态而拒绝发送Message，则返回false
    void receive(const Packet& ackPkt); //接受确认Ack，将被NetworkServiceSimulator调用
    void timeoutHandler(int seqNum);//Timeout handler，将被NetworkServiceSimulator调用
public:
    TCPRdtSender();
    virtual ~TCPRdtSender();
```

```
};
```
③接收方 receiver
```
class TCPRdtReceiver :public RdtReceiver
{
private:
    int expectSequenceNumberRcvd;  // 期待收到的下一个报文序号
    Packet lastAckPkt;                    //上次发送的确认报文
public:
    TCPRdtReceiver();
    virtual ~TCPRdtReceiver();
public:
    void receive(const Packet& packet);    //接收报文，将被NetworkService调用
};
```

1.4 系统实现

1.4.1 GBN 协议

1. 发送方 sender

（1）send 函数

该函数将报文序列发送给接收方：

①首先确认当前的等待状态，如果等待状态为 true，表明当前的报文发送队列已满，返回 false，如果当前等待状态为 false，表明当前的报文发送序列还有位置，可以继续发送；

②初始化报文序列：报文序列的序列号赋值为下一个空闲的序列号，该序列号也是发送队列中该报文的位置；将应用层发送的 message 的数据复制给该报文的数据段中，通过检验和计算函数更新检验和；

③将该报文发送给网络层，调用模拟环境中的发送到网络层的函数；

④判断基序列号是否与当前发送的报文的序列号相等，如果相等，就启用计时器计时发送队列发送的第一个报文的时间；

⑤更新下一个空闲的序列号；

⑥更新等待状态：如果当前的发送队列已经满了，就将等待状态设置为 true；判断状态的操作为计算发送队列占用的长度与滑动窗口的长度进行比较，如果更小就表示没满。

（2）receiver 函数

该函数接收从接收方发送的 ACK 包，通过该包判断基序列报文是否发送成功，根据判断结果进行下一步操作：

①计算 ACK 包的检验和，如果检验和与发送过来的包的检验和相同表明数据传送没有问题，如果不相同就执行重传操作；

②判断 ACK 包的序列号与基序列号是否相同，如果不同表明接收到的包出现问题，将执行

重传操作；

③如果没有①②的问题，更新当前发送方的相关信息：关闭当前基序列对应的计时器；基序列号更新为下一个报文序列号；开启新的基序列的计时器；并且此时空出来了一个新的报文位置，需要将等待状态置为 false；

④如果有问题，将重启该基序列的计时器并重新向网络层重新传送该基序列的报文；同时更新下一个传送的报文序列为该基序列的下一个，重传所有的报文，此时队列空闲，将等待状态置为 false。

（3）timeoutHandler 超时处理函数

计时器记录的是基序列的发送时间，因此如果超时：

①关闭计时器；

②将下一个发送的报文序列号置为当前的基序列号，表明从基序列开始重传；

③将等待状态置为 false；

④之后等待网络层调用 send 函数重新传送报文。

（4）其他函数：

GBNRdtSender 函数为将数据结构初始化的函数，这里需要将 nextSeqNum 初始化为 0, base 初始化为 0, waitingState 初始化为 false。

getWaitingState 函数为获取等待状态的函数，直接将 waitingState 的值返回即可，这里修改等待状态的代码在 send 函数中。

~GBNRdtSender 函数为析构函数，这里可以为空。

2. 接收方 receiver

（1）GBNRdtReceiver 函数

该函数初始化了 ACK 包，对期待序列号初始化为 0，对确认序号初始化为-1，对确认报文初始化为“receive success”并确定检验和。模拟网络环境定义了一个 lastAckPkt 的包来保存相关信息。

（2）receive 函数

①判断收到的数据包的检验和与本身传送过来的检验和是否相等，以及传送的数据包的序列号与期待序列号是否相同；

②如果都相等，表明传送成功：将该数据报文的数据包发送给应用层的 message；更新确认序号为当前收到的报文序号，并更新 ack 报文和检验和、发送报文给网络层；更新期待报文序号；

③如果不相等，表明传送报文失败，将 ack 报文的数据更新为“receive error”并更新检验和将其发送给发送方。

1.4.2 SR 协议

1. 发送方 sender

（1）发送函数 send

发送函数将报文传递给网络层，通过模拟网络环境传递给接收方：

①判断当前等待状态，如果处于等待状态将直接返回 false；

②对于将要发送的报文序列，其序列号为 nextseqnum：将 ack 序列中该序号的 ack 置为 false；

③将当前的报文的序号置为 nextseqnum；

④将报文 message 的数据复制给报文并计算检验和；

⑤对该序号的报文启动一个定时器并将该报文传送给网络层；

⑥更新下一个报文序号。

（2）接收函数 receive

①判断当前收到的 ack 包的报文序号是否合法，如果不合法直接 return；

②判断当前的 ack 包的检验和与计算的检验和是否相等以及报文序号是否处于滑动窗口中，如果不满足，表明传送失败，将等待超时重传；

③如果满足，表明该包传送成功，关闭该报文对应的计时器，将该报文对应的 ack 状态置为 true；

④判断当前收到的报文序号是否为基序列号，如果不是，表示报文传送失序，等待报文基序列超时重传；

⑤如果是基序列号，更新滑动窗口：从滑动窗口的第一个报文开始找到第一个 ack 状态为 false 的报文，将基序列号置为该报文的序号，在遍历的过程中，将 ack 状态为 true 的报文状态置为 false；

⑥将等待状态置为 false。

（3）其他函数

超时处理函数 timeoutHandler 将根据报文序号重启计时器再重传。

SRRdtSender 函数对数据进行初始化：nextseqnum 初始化为 0, sendbase 初始化为 0, waitingstate 初始化为 false， ACK 数组整个初始化为 false。

getWaitingState 函数直接返回等待状态 waitingstate。

2. 接收方 receiver

（1）SRRdtReceiver 函数

①对缓存队列的状态都初始化为 false；

②对报文期待的基序列号初始化为 0；

③对报文数据初始化为“receive success”并计算检验和。

（2）receive 函数

①判断收到的报文的序列号是否处于缓存队列的期待序号中，如果不在，发送的序列号出错，将 ack 报文更新为“receive error”并计算检验和发送给发送方；

②判断报文的检验和与计算的检验和是否相等，如果不相等表明报文受损，等待发送方超时重传；

③如果检验和正确，判断该报文的序号与期待的报文基序列号是否相同；

④如果不相同，需要将该报文缓存在缓存队列并将缓存状态置为 true；

⑤如果相同，需要将该报文传送给应用层的 message，并更新缓存队列的期待序列号：从基序列号的下一个开始找到第一个缓存状态为 false 的报文序号，将其设置为期待的基序列号，并且在遍历过程中将遍历的缓存状态为 true 的报文传递给应用层；

⑥发送 ack 报文给发送方来确认。

1.4.3 TCP 协议

1. 发送方 sender

（1）发送函数 send

发送函数将报文传递给网络层，通过模拟网络环境传递给接收方：

①判断当前等待状态，如果处于等待状态将直接返回 false；

②对于将要发送的报文序列，将当前的报文的序号置为 nextseqnum；

③将报文 message 的数据复制给报文并计算检验和并将该报文传送给网络层；

⑤如果传送的这个报文就是基序列好，对该序号的报文启动一个定时器；

⑥更新下一个报文序号。

（2）接收函数 receive

①判断当前的 ack 包的检验和与计算的检验和是否相等，如果不满足，表明传送失败，将等待超时重传；

②判断 ack 报文序号是否处于滑动窗口中，如果不满足，表明是期待的冗余报文，将其记录了下来并记录传递过来的次数，来实现快速重传；

③如果当前的冗余报文累计连续传递的次数达到三次，将调用模拟网络环境中的函数将该序号对应的报文快速的传递给网络层；

④如果满足，表明该包传送成功，关闭该报文对应的计时器，更新基序列号的值并重新启动一个计时器；

⑤将等待状态置为 false。

（3）其他函数

超时处理函数 timeoutHandler 将根据报文序号重启计时器再重传。

SRRdtSender 函数对数据进行初始化：nextseqnum 初始化为 0, sendbase 初始化为 0, waitingstate 初始化为 false，y 序号初始化为-1，num 初始化为 0。

getWaitingState 函数直接返回等待状态 waitingstate。

2. 接收方 receiver

（1）TCPRdtReceiver 函数

①初始化期待报文序列号为 0；

②初始化 ack 报文：将报文数据置为“receive success”，计算检验和；将序列号和确认号先初始化为-1。

（2）receive 函数

①判断接收到的报文序号是否为期待报文序列号以及检验和是否与计算的检验和相等；

②如果不相等，表明传送失败，将 ack 报文数据更新为“receive error”并重新计算检验和；

③如果都相等，发送报文成功，将数据传递给应用层的 message；

④更新 ack 报文中的确认号 acknum，将该报文传递给网络层发送给发送方；

⑤更新期待收到的序列号。

1.5 系统测试及结果说明

1.5.1 测试环境

1. 操作系统：Windows10；
2. 处理器：Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz；
3. 测试程序：提供的脚本文件 check_win.bat。

1.5.2 测试结果及分析

对三个协议通过 VS2019 运行，可以得到模拟网络环境的相关输出信息和自己编写的程序中的输出信息，这里包括了滑动窗口、接收方和发送方的信息传递信息等。

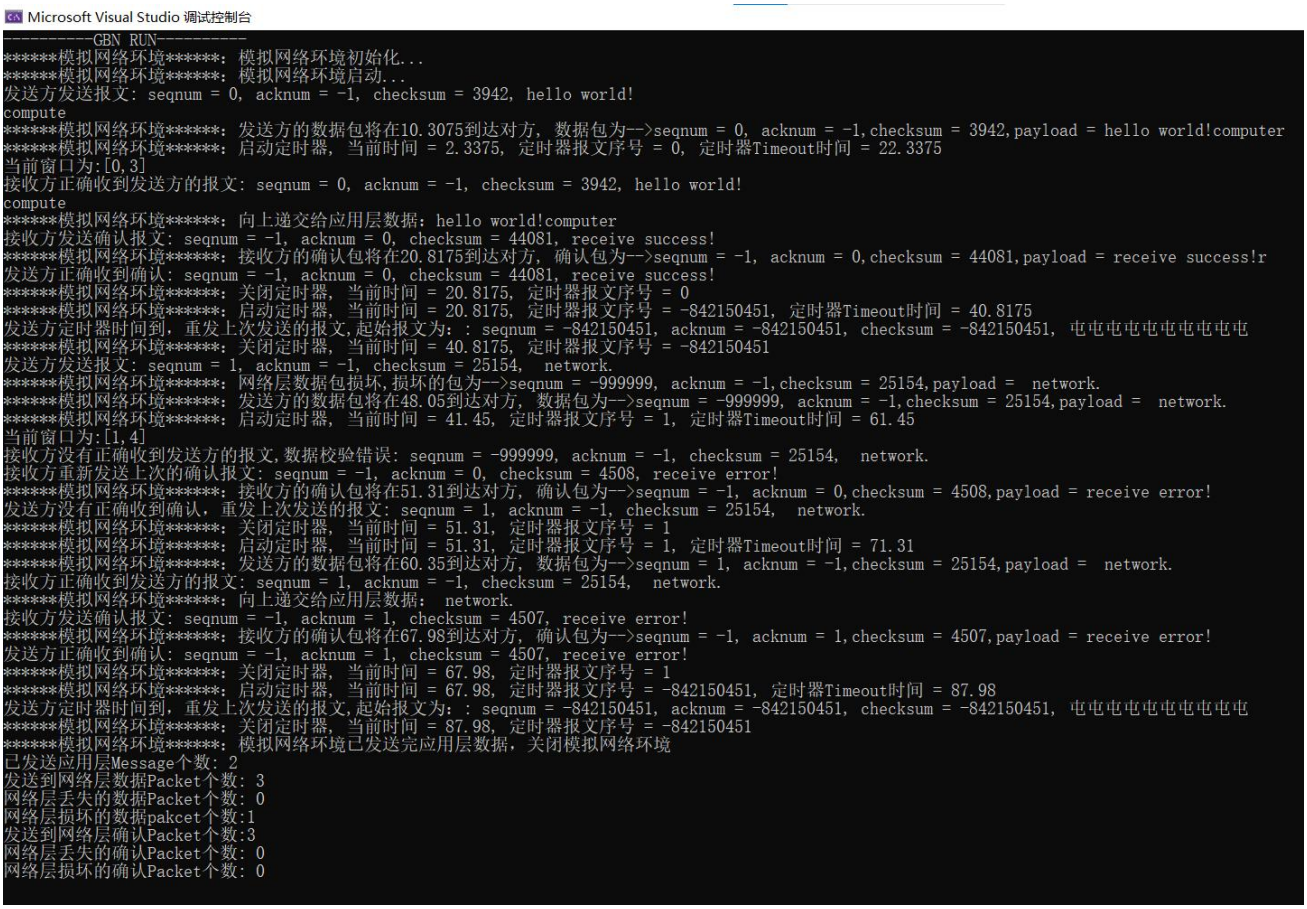

图 1 GBN 运行结果图

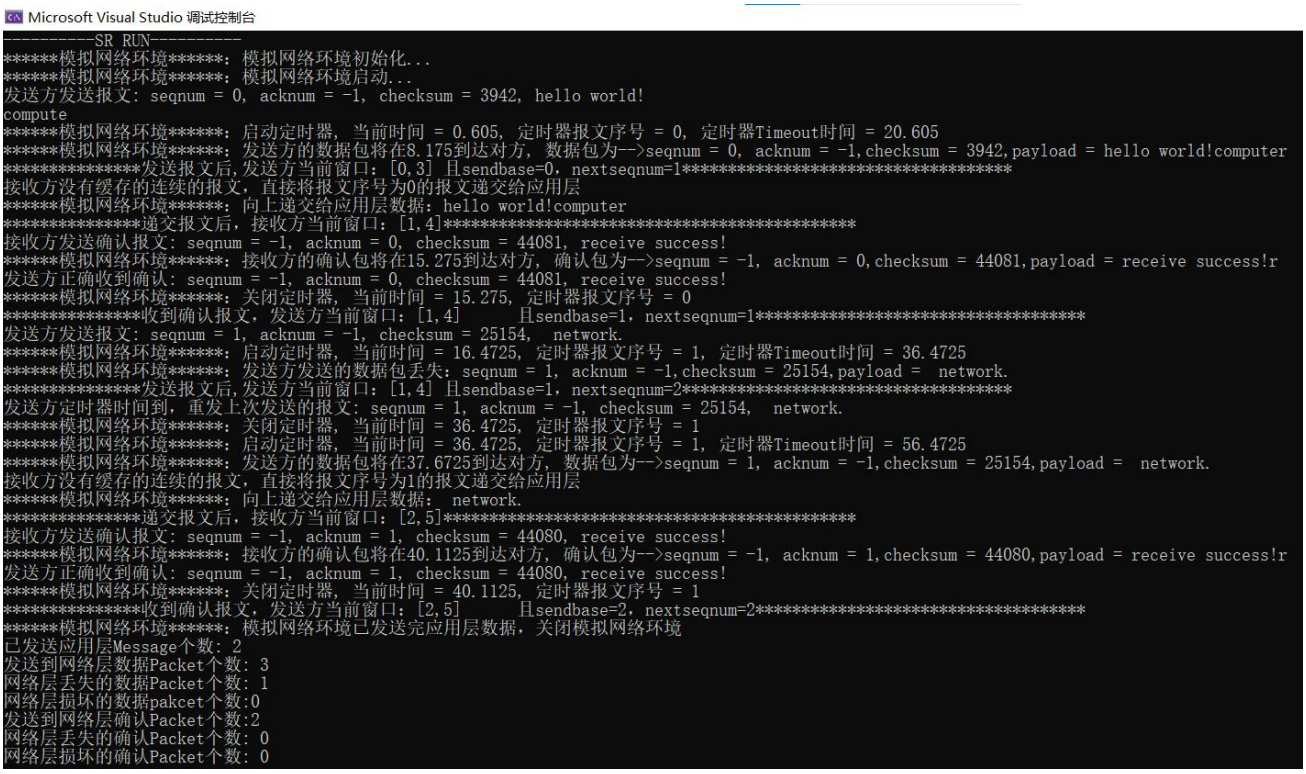

图 2 SR 运行结果图

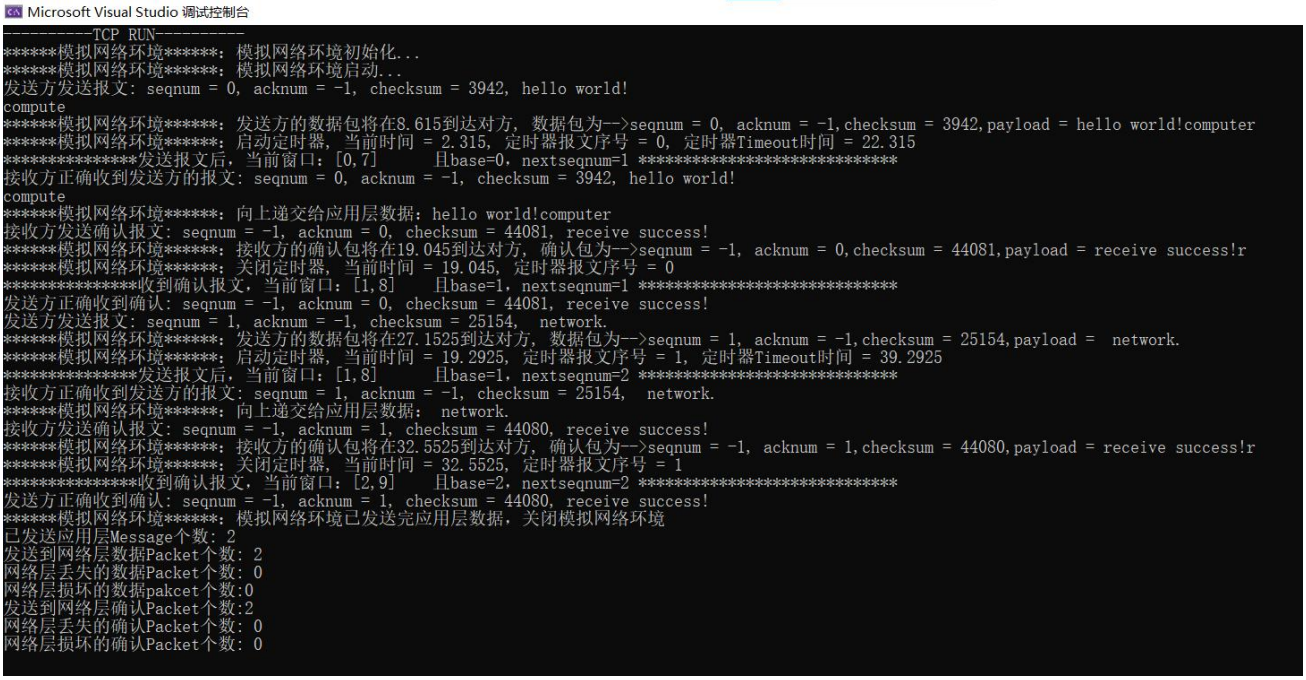

图 3 TCP 运行结果图

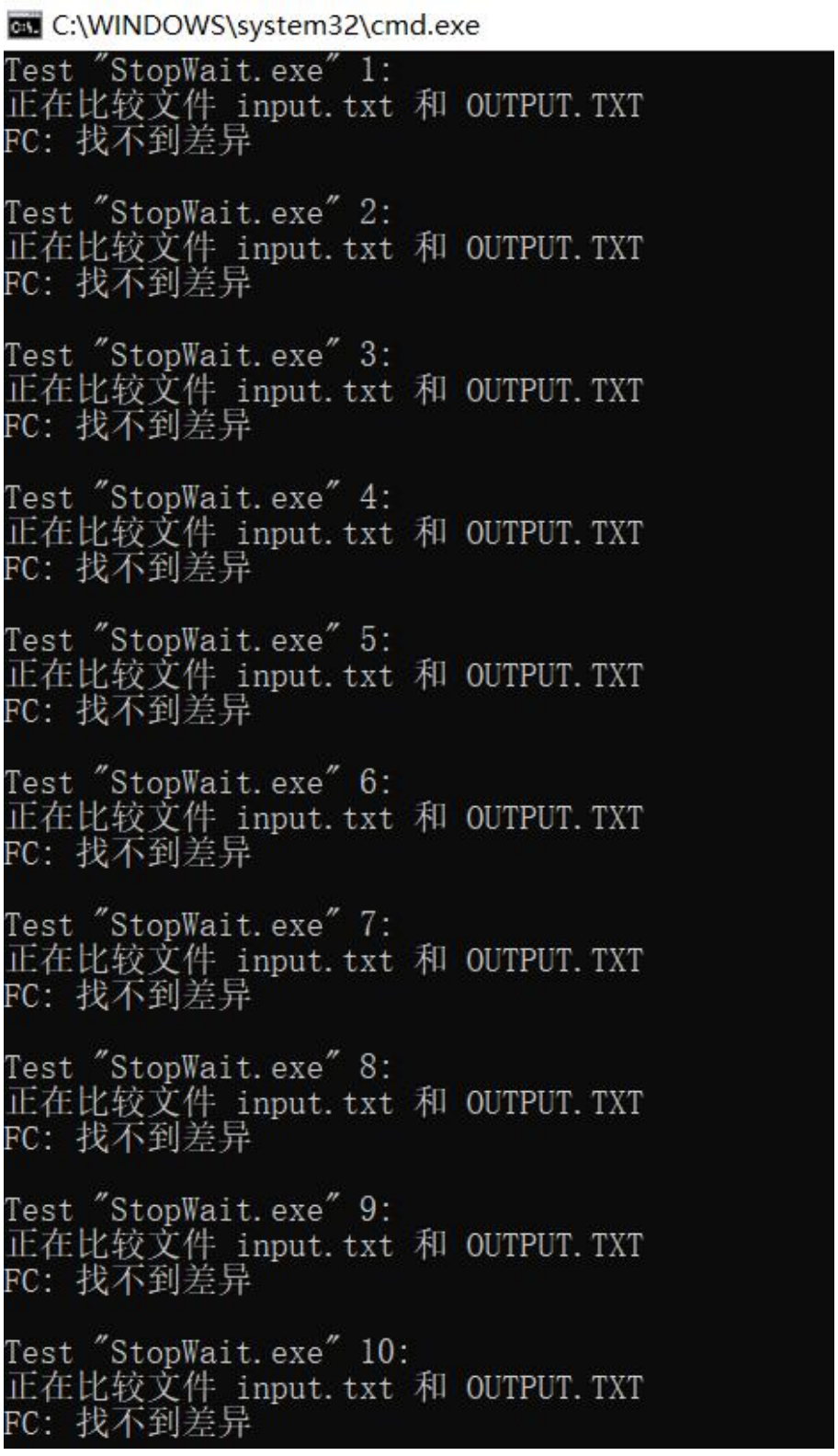

图 4 脚本运行结果图

1.6 其它需要说明的问题

在运行的过程中，需要注意路径的相关信息，脚本文件、运行文件等都要在同文件下。

1.7 参考文献

1. 计算机网络·自顶向下的方法 James F.Kurose,Keith W.Ross
2. 模块二 可靠数据传输协议设计(修订版 V2020)
3. 百度百科&CSDN

心得体会与建议

2.1 心得体会

在该实验中，我实现了 GBN、SR、TCP 三个协议的代码以及通过模拟网络环境验证了代码的正确性，整个过程是曲折但充满收获的。

老师提供的文件中给了一份简易的传输协议实现代码，这些代码虽然不是我们实验需要检查的一部分，但是向我们提供了很多信息。在这次的实验中，需要用到许多模拟网络环境中的函数，来完成最终整体的功能，而这些函数便需要我们阅读提供的文档来知晓他们的功能从而实现灵活的调用，但是文字版的代码总是不易懂的，而提供的 StopWait 中代码调用了这些函数来为自己服务，使得我对这些函数的功能以及使用更加的清晰了，他相当于一份协议代码实现模板！

虽然有了这样的一个模板，但是还需要深入理解三个协议的原理从而结合该模板来完整最后的实验。在理论课上，这三个协议都被详细的讲解过，因此原理层面并不难，重要的是代码实现层面。

这个实验很好的将代码实现层面与原理层面相结合，非常贴合课上所学的内容，因此对理论课的学习相当于是一次巩固，同时还锻炼了代码能力，让我们对计算机网络的底层有了更多的了解。尤其是模拟网络环境的使用，极大的简便了大家对操作环境的安装。

这个实验让我学到了很多，在这个过程中我也非常感谢解答我问题的各位老师、助教和同学，让我对计算机网络有了更深入的学习。

2.2 建议

这门课程相较于理论课的开设时间有点靠后，或者将该实验课设置的靠前一些，在相关知识刚好讲完的时候就开始做实验，或许可以有更好的效果。