北京邮电大学《计算机网络》课程实验报告

<table>
<tr><td>实验
名称</td><td colspan="2">数据链路层滑动窗口协议的设计与实现</td><td>学
院</td><td>计算机</td><td>指导教师</td><td>王晓茹</td></tr>
<tr><td>班 级</td><td>班内序号</td><td colspan="2">学 号</td><td>学生姓名</td><td colspan="2">成绩</td></tr>
<tr><td>2017211318</td><td>14</td><td colspan="2">2017211661</td><td>陈斌</td><td colspan="2"></td></tr>
<tr><td>2017211318</td><td>15</td><td colspan="2">2017211666</td><td>李奕阳</td><td colspan="2"></td></tr>
<tr><td>2017211318</td><td>16</td><td colspan="2">2017211681</td><td>罗阳</td><td colspan="2"></td></tr>
<tr><td>实
验
内
容</td><td colspan="6">本次实验选用的滑动窗口协议为回退 N 帧协议以及选择重传协议，并使用了 NAK 机制，利用所学数据链路层原理，自行设计一个滑动窗口协议，在仿真环境下编程实现有噪音信道环境下两站点之间无差错双工通信。信道模型为 8000bps 全双工卫星信道，信道传播时延 270 毫秒，信道误码率为 10^{-5}，信道提供帧传输服务，网络层分组长度固定为 256 字节。</td></tr>
<tr><td>学生实验报告</td><td colspan="6">（详见“实验报告和源程序”册）</td></tr>
<tr><td>课
程
设
计
成
绩
评
定</td><td colspan="6">评语：

成绩：

指导教师签名：
2019 年 月 日</td></tr>
</table>

目录

一、实验内容和实验环境描述

1、 实验内容

2、 实验目标

3、 实验设备环境

二、协议设计

三、软件设计

1、 数据结构

2、 模块结构

3、 算法流程

四、实验结果分析

五、探究问题

六、实验总结与心得体会

七、源程序清单

计算机网络滑动窗口协议实验报告

一、 实验报告和实验环境描述

1、 实验内容：

利用所学数据链路层原理，自己设计一个滑动窗口协议，在仿真环境下编程实现有噪音信道环境下两站点之间无差错双工通信。

信道模型为8000bps 全双工卫星信道，信道传播时延270毫秒，信道误码率为10^{-5}，信道提供字节流传输服务，网络层分组长度最长为256字节。

2、 实验目标：

通过该实验，进一步巩固和深刻理解数据链路层误码检测的CRC校验技术，以及滑动窗口的工作机理。滑动窗口机制的两个主要目标：

(1) 实现有噪音信道环境下的无差错传输；

(2) 充分利用传输信道的带宽。

在程序能够稳定运行并成功实现第一个目标之后，运行程序并检查在信道没有误码和存在误码两种情况下的信道利用率。

为实现第二个目标，提高滑动窗口协议信道利用率，需要根据信道实际情况合理地为协议配置工作参数，包括滑动窗口的大小和重传定时器时限以及ACK 搭载定时器的时限。

3、 实验环境：

本次实验的环境可分为硬件环境与软件环境，其具体内容分别如下所示：

① 硬件环境：已装载 Windows 10 系统环境的 PC 机；

② 软件环境：Microsoft Visual Studio 2017 集成开发系统。

二、 协议设计

本次实验中我们小组设计的协议共涉及网络参考模型中的三个层次： 物理层、数据链路层、网络层。

① 网络层：利用数据链路层提供的“可靠的分组传输”服务，在站点A与站点B之间交换长度最长为为256 字节的数据分组。网络层把产生的分组交付数据链路层，并接受数据链路层提交来的数据分组；

② 数据链路层：从网络层接收要发送的数据包，将之组装成帧，向物理层发送，启动计时器；进行适当的流量控制；数据帧经信道传送给接收方；接收方数据链路层终止定时器（或启动 ack 定时器，ack 成帧传送），判断数据是否出错，若正确，则判断是否为预定接受数据，最终将其提交给网络层；

③ 物理层：为数据链路层提供的服务为 8000bps，270ms 传播延时，10-5 误码率的字节流传输通道。

本次实验主要设计数据链路层，为实现有噪声环境下高信道利用率传输，我们采用 Go-Back-N 协议和选择重传协议分别进行设计与测试

可靠通信的实现方式：通过捎带确认来完成可靠的数据通信。

1）出现信道误码导致收帧出错时，接收方发 NAK 帧要求发送方重传；

2）在 Go-Back-N 协议中接受方发 NAK 帧的方式要求对方进行重传；在选择重传协议中，当收到的帧位于接收窗口内，但不是接收窗口下边界的一帧时，将该帧进行缓存,待窗口下边界的帧到来后一起向网络层递交；

3）接收方正常情况下使用稍待确认方式，将数据与 ACK 确认一同发送。当 ACK 定时器时间内，无数据传送导致发送方无法收到捎带确认时，接收方 ACK 定时器超时，构造 ACK 帧单独传送。

三、 软件设计

① Go-back-N 协议

1、数据结构定义：

变量类型定义：

```
typedef unsigned char seq_nr;
typedef unsigned char packet[PKT_LEN];
```

帧结构定义：

```
typedef struct{
unsigned char kind;  //帧类型
seq_nr ack;          //ack 序号
seq_nr seq;          //发送帧序号
packet data;         //传送数据
unsigned int crc;    //CRC 校验位
}FRAME;
```

各类型的帧格式如下：

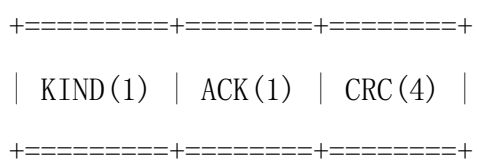
DATA Frame

KIND(1)	ACK(1)	SEQ(1)	DATA(240~256)	CRC(4)

ACK Frame

KIND(1)	ACK(1)	CRC(4)

NAK Frame



宏定义：

帧的类型：

```
#define FRAME_DATA 1
#define FRAME_ACK  2
#define FRAME_NAK  3
```

发生事件类型：

```
#define NETWORK_LAYER_READY  0
#define PHYSICAL_LAYER_READY   1
#define FRAME_RECEIVED        2
#define DATA_TIMEOUT          3
#define ACK_TIMEOUT           4
```

通信协议控制参数：

```
#define DATA_TIMER  2000
#define ACK_TIMER 300
#define MAX_SEQ 7
```

序号增加函数：

```
#define inc(x) x = (x + 1)%(MAX_SEQ + 1)
```

全局变量定义：

```
seq_nr  frame_nr = 0;
```

```
seq_nr  ack_expected = 0;
seq_nr  nbuffered = 0;
seq_nr  frame_expected = 0;
packet buffer[MAX_SEQ + 1];
int no_nak = 1;
```

主函数变量定义

```
int event, arg;  //事件及参数
struct FRAME f; //接收帧
int len = 0 ;    //记录帧长度
```

2、模块结构

① 子函数定义及功能介绍

1）static int between(unsigned char a, unsigned char b, unsigned char c)

功能：在一组具有循环性质的序号中判断 a、b、c 是否满足 b 在 a 与 c 之间，以判断帧是否落在窗口内。

2）static void put_frame(unsigned char *frame, int len)

参数： frame 表示该帧在内存中的首地址，len 表示该帧的长度；

功能： 在该帧尾计算添加校验和，并向物理层发送。

3)static void send_data(unsigned char frame_nr, unsigned char frame_expected, unsigned char buffer[MAX_SEQ + 1][PKT_LEN])

参数： frame_nr 表示将要发送的帧的序列号；

frame_expected 表示接受窗口的下边界；

buffer[MAX_SEQ + 1][PKT_LEN] 表示发送方缓冲区；

功能：向物理层发送数据帧。

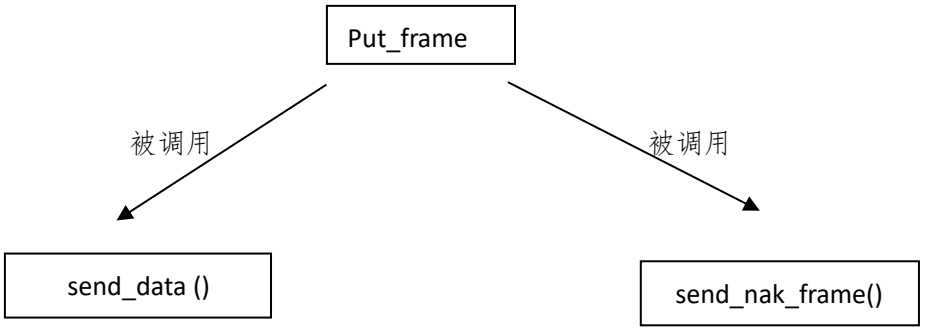
4）static void send_nak_frame(unsigned char frame_expected)

参数：frame_expected 表示接受窗口下边界；

功能： 向物理层发送 NAK 帧。

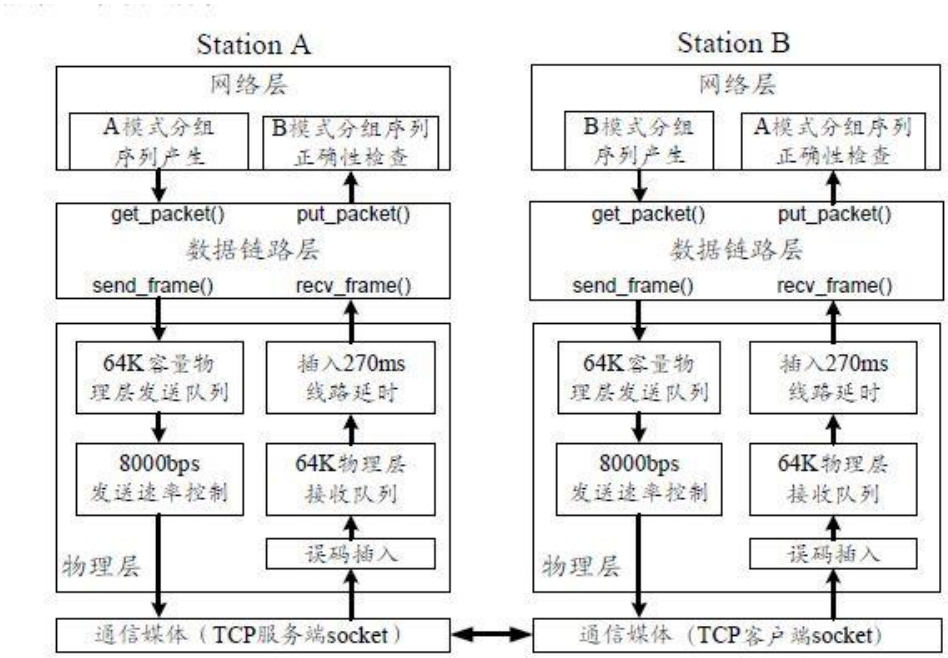
② 子函数之间的调用关系：

函数 3）4）各自均调用函数 2）以完成帧的发送

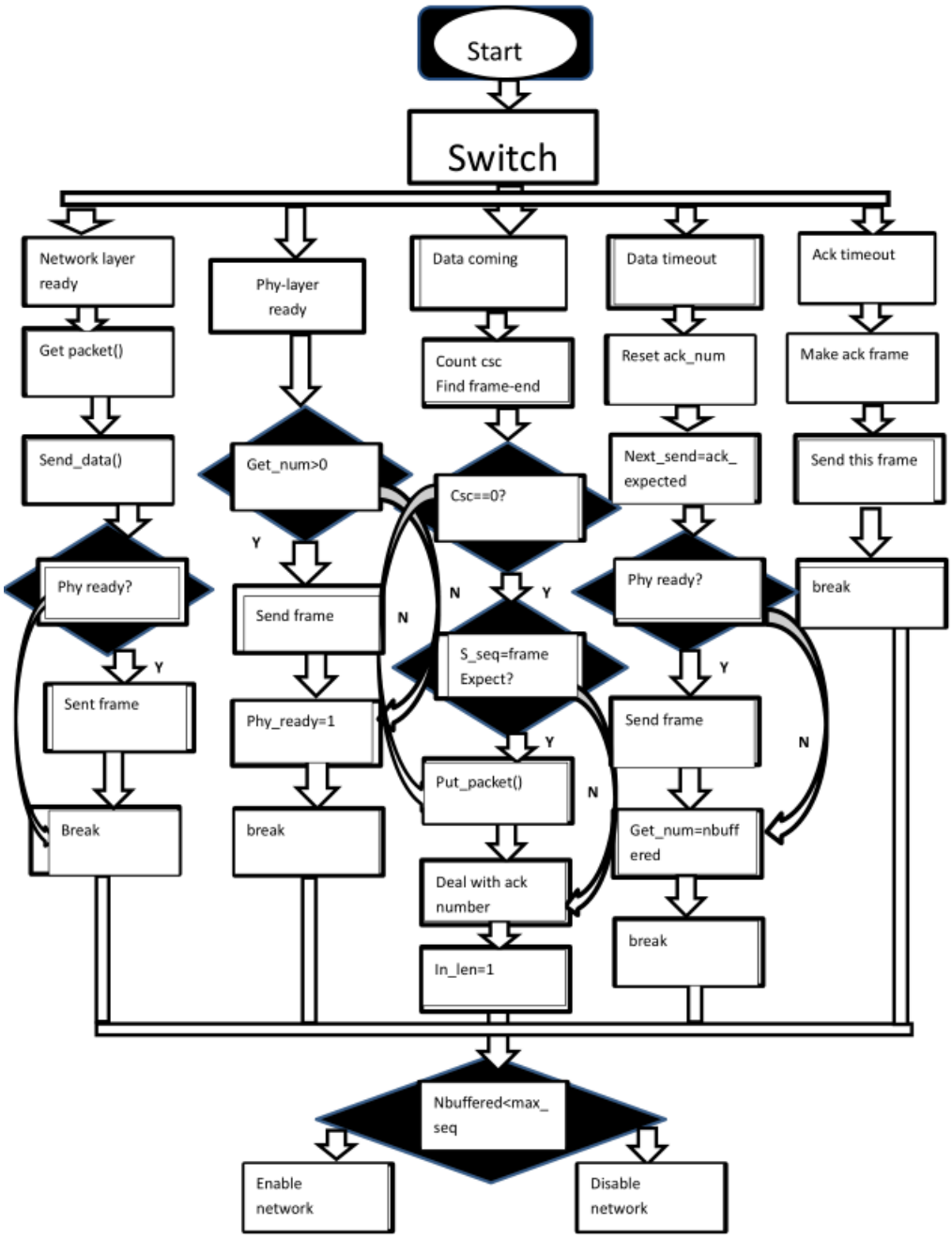


3、算法流程

① 程序总体机构



③ 算法流程图



② 选择重传协议

1、数据结构定义：

变量类型定义：

```
typedef enum { false, true } bool;
```

```
typedef unsigned char seq_nr;
typedef unsigned char packet[PKT_LEN];
```

帧结构定义：

```
typedef struct{
unsigned char kind;  //帧类型
seq_nr ack;          //ack 序号
seq_nr seq;          //发送帧序号
packet data;         //传送数据
unsigned int crc;    //CRC 校验位
}FRAME;
```

各类型的帧格式如下：

DATA Frame

KIND(1)	ACK(1)	SEQ(1)	DATA(240~256)	CRC(4)

ACK Frame

KIND(1)	ACK(1)	CRC(4)

NAK Frame

KIND(1)	ACK(1)	CRC(4)

宏定义：

帧的类型：

```
#define FRAME_DATA 1
#define FRAME_ACK  2
#define FRAME_NAK  3
```

发生事件类型：

```
#define NETWORK_LAYER_READY  0
#define PHYSICAL_LAYER_READY   1
#define FRAME_RECEIVED        2
#define DATA_TIMEOUT          3
#define ACK_TIMEOUT           4
```

通信协议控制参数：

```
#define DATA_TIMER  2000
#define ACK_TIMER 300
#define MAX_SEQ 7
#define NR_BUFS ((MAX_SEQ + 1) / 2)
```

序号增加函数：

```
#define inc(x) x = (x + 1)%(MAX_SEQ + 1)
```

全局变量定义：

```
static int phl_ready = 0;
bool no_nak = true;
```

主函数变量定义：

```
seq_nr next_frame_to_send = 0,
       ack_expected = 0,
```

```
        frame_expected = 0,
        too_far = NR_BUFS,
       nbuffered = 0,
       i;
packet out_buf[NR_BUFS],
       in_buf[NR_BUFS];
bool arrived[NR_BUFS];
int event, arg, len = 0;
frame r;
```

2、模块结构

子函数定义及功能介绍

1）static int between(unsigned char a, unsigned char b, unsigned char c)

功能：在一组具有循环性质的序号中判断 a、b、c 是否满足 b 在 a 与 c 之间，以判断帧是否落在窗口内。

2）static void put_frame(unsigned char *frame, int len)

参数： frame 表示该帧在内存中的首地址，len 表示该帧的长度

功能： 在该帧尾计算添加校验和，并向物理层发送

3） static void send_data(unsigned char fk, seq_nr frame_nr, seq_nr frame_expected, packet buffer[])

参数： fk 表示发送帧的类型

frame_nr 表示将要发送的帧的序列号 ，

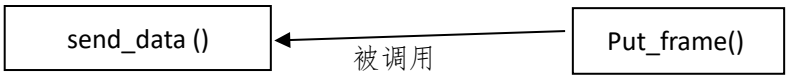
frame_expected 表示接受窗口的下边界，

buffer[] 表示发送方缓冲区

功能：向物理层发送数据帧

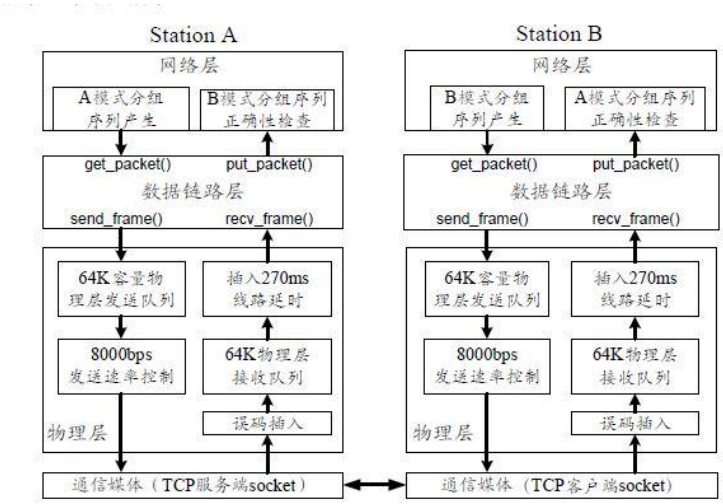
子函数之间的调用关系：

如下图所示，函数 3）调用函数 2）以完成帧的发送操作

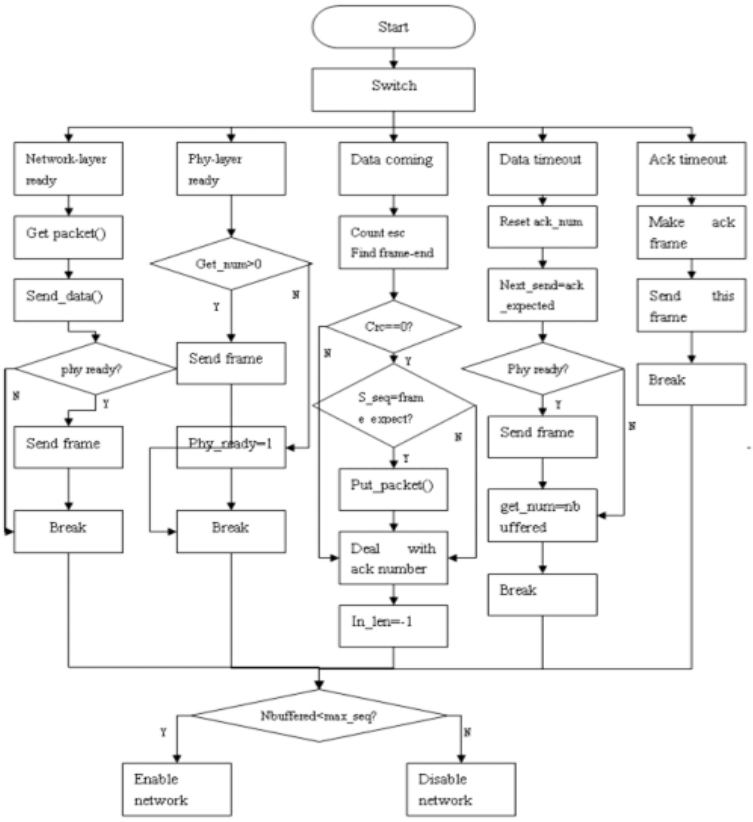


3、算法流程

① 程序总体机构



② 算法流程图



四、实验结果预测分析

(1) 描述你所实现的协议软件是否实现了有误码信道环境中无差错传输功能。

我们小组成功实现了有误码信道环境中无差错传输功能；协议可通过校验和发现错误，从而让错误帧进行重传操作，而且保证数据帧的有效顺序递交给网络层。

(2) 程序的健壮性如何，能否可靠地长时间运行。

我们小组设计的协议程序健壮性良好，能够可靠地运行较长时间。效率在程序运行随着时间推移稍有降低，约20分钟后达到一个稳定的状态。

(3) 协议参数的选取：

1）滑动窗口大小确定：

由于分组的最大长度为256字节，加上于链路层成帧时在数据前后加入的帧头以及校验位，分别为1个字节的类型说明frame.kind，1个字节的帧序列号frame.seq，1个字节的ACK号frame.ack，以及4个字节的CRC校验位frame.padding。因此，每个稍待确认的数据帧都是263字节，所以发送一帧的时间为：

$$263\text{B} \times 8 \div 8000\text{bps} = 263\text{ms}。$$

由于线路延迟为270ms，并且采用ACK搭载回传技术。最好的情况下，设发送窗口大小为N，则为了获得更高的线路利用率，需要满足以下不等式：

$$263\text{ms} \times \text{N} \geq (263\text{ms} + 270\text{ms}) \times 2$$

由上式我们可求得N的最小值为5，即在理想情况下，当N≥5时，信道的利用率就可以达到100%，但这并不能说明滑动窗口大小的最优值为5。这是由于接收方收到一个帧后并不一定立即有数据回传，以及其他传输过程中发生的延迟与错误等现象，因此N值的设定还需要进一步实验进行确定。

考虑ACK发送超时时间，窗口大小应为：

$$[2 \times （发送时间 + 线路延迟） + 数据帧延迟] \div 发送时间$$

一般情况下，窗口越大，信道的利用率越高，通信效率会越高。但窗口太大可能会导致下列问题出现：1.发送的帧太多，物理层排序的队列太长，会出现拥塞现象；2.出现错误的情况下，容易造成很多帧要重传。

我们小组在通过设置不同的窗口大小，经过大量测试比较其实际效率，发现在Go-Back-N协议中的MAX_SEQ设为7时效率达到较好的水平，此时其发送窗口大小为7，接收窗口大小为1；在我们设计的Selective协议程序中，当MAX_SEQ设为15时，协议的整体运行效率最高，此时发送窗口与接收窗口的大小均为8。如下表所示：

	Go-Back-N Protocol	Selective Repeat Protocol
发送窗口大小	7	8
接收窗口大小	1	8

2)重传定时器时限确定：

正如1）中所算结果，从发送一帧到该帧的ACK通过捎带技术回传至发送端，至少需要：

$$(263\text{ms} + 270\text{ms}) \times 2 = 1066\text{ms}。$$

考虑到对方接收到一帧后不一定马上有分组要回传，另外加上回传时需要在物理层排队队列中等待的时间。若时限太大，则当发生丢失等错误后，延迟会很长，阻塞了正常的通信。若时限太小，会导致很多不必要的重传，从而降低通信效率。通过尝试不同的取值，我们小组在Go-Back-N协议程序中设置数据帧重传定时器时间为2000ms，在Selective协议程序中设置数据帧重传定时器时间为3000ms。

3)ACK搭载定时器时限确定：

接受方接收到一个帧后，启动ACK定时器，这个时间至少设定为一个帧发送所要时间

$$263\text{B} \times 8 \div 8000\text{bps} = 263\text{ms}$$

再考虑到等待回传帧出现的时间和回传帧在物理层排队的时间，小组在通过大量实践尝试不同取值，最终在两个协议程序中均将ACK发送超时时间确定为300ms。

故本次实验中所使用的两个协议的重传定时器参数分别如下表所示：

	Go-Back-N Protocol	Selective Repeat Protocol
数据重传定时器	2000ms	3000ms
ACK重传定时器	300ms	300ms

(4)信道传输效率理论分析：

由于每一个帧中不仅含有数据，同时还需要携带其他信息，所以在理想状态下的最大信息利用率为：

$$\frac{256}{256+4+3} \times 100\% \approx 97.19\%$$

数据链路层提供的服务为 8000bps，所以每传输一个字节耗时 1ms，每帧的附加信息固定为 7byte，发送需耗时 7ms。

我们将模型进行简化，假设信道上始终有数据需要传输，均可以使用稍待确认。则在误码率为10^{-5}的信道上，发送 100000 个比特时已发数据包数量大致为：

$$\frac{100000}{263 \times 8} \approx 48$$

即每传送 48 个数据包将有 1 个数据包出错。假设在限定时间内可以重传的该帧均不再发生错误，则每传送 48 个数据包至少需传送 48+1=49 次，所以此时

的信道利用率为：

$$\frac{48\times256}{49\times263}\times100\%\approx95.35\%$$

若程序设计采用 Go-Back-N 协议，假设每发送 48 个数据帧中的第一个帧出错，则此协议下的信道利用率约为：

$$\frac{48\times256}{(48+N)\times263}\times100\%$$

其中 N 为发送窗口大小。当 N=7 时，信道利用率约为 84.94%。当在误码率为 1e-4 的信道上进行通信时，发送 10000 个比特时已发数据包数量大致为：

$$\frac{10000}{263\times8}\approx4.8$$

即大约每 5 个帧中将有一个帧出错。故此时的信道利用率为：

$$\frac{5\times256}{(5+N)\times263}\times100\%$$

当 N=7 时，此时理想状态下的信道利用率仅约为 40.55%

(5)实验结果记录与分析：

Go-back-N 协议程序性能测试记录表

序号	命令	说明	运行时间(秒)	效率(%)		备注
				A	B	
1	datalink au datalink bu	无误码信道数据传输	1800	58.01	96.97	
2	datalink a datalink b	站点A分组层平缓方式发出数据，站点B周期性交替“发送100秒,停发100秒”	1800	50.61	88.30	

3	datalink afu datalink bfu	无误码信道，站点 A 和站点 B 的分组层都洪水式产生分组	1800	96.97	96.97	
4	datalink af datalink bf	站点 A/B 的分组层都洪水式产生分组	1800	86.37	86.19	
5	datalink af -ber 1e-4 datalink bf -ber 1e-4	站点 A/B 的分组层都洪水式产生分组，线路误码率设为 10^{-4}	1800	39.61	40.48	

Selective 协议程序性能测试记录表

序号	命令	说明	运行时间(秒)	效率(%)		备注
				A	B	
1	datalink au datalink bu	无误码信道数据传输	1800	54.97	96.97	
2	datalink a datalink b	站点 A 分组层平缓方式发出数据，站点 B 周期性交替“发送 100 秒，停发 100 秒”	1800	53.40	94.44	
3	datalink afu datalink bfu	无误码信道，站点 A 和站点 B 的分组层都洪水式产生分组	1800	96.97	96.97	
4	datalink af datalink bf	站点 A/B 的分组层都洪水式产生分组	1800	94.02	93.55	
5	datalink af -ber 1e-4 datalink bf -ber 1e-4	站点 A/B 的分组层都洪水式产生分组，线路误码率设为 10^{-4}	1800	60.11	57.33	

四、 探究问题

CRC校验能力：

CRC校验码的检错能力很强，它除了能检查出离散错外，还能检查出突发错，CRC校验码具有以下检错能力：CRC校验码能检查出全部单个错；CRC校验码能检查出全部离散的二位错；CRC校验码能检查出全部奇数个错；CRC校验码能检查出全部长度小于或等于K位的突发错；CRC校验码能以[1-（1/2）^(K-1)]的概率检查出长度为（K+1）位的突发错。

由于本次实验过程的误码信道是一个比较固定的误码率，而在实际生活当中的误码率不是稳定的，可能会因为传输环境的不同，使得他的误码率波动比较大的，例如，下雨天和晴天，高噪声和低噪声的情况，传输的距离也是影响因素。对于这种动态的误码率的通信过程，可能需要其他的一些参数来控制基本参数值（窗口大小，重传时间等等）来完成。

get_ms()实现：

C语言的time.h当中提供了一些关于时间操作的函数可以实现get_ms()函数。可以利用的函数有clock()函数原型为：clock_t clock()。该函数返回程序开始执行后占用的处理器时间，如果无法获得占用时间则返回-1。因为我们计时的起点并不是程序开始之时，而是开始通信之时，所以需要一个静态变量start_time来记录通信起始的时间。然后在每次调用get_ms()后，获取当前的时间current_time。然后再返回start_time-current_time即可。

定时器区别：

start_timer()是对数据帧的定时，该定时时限的时间起点应该是该帧开始发送的时刻，所以要等到物理层排队序列低于50个字节，该帧可以发送才开始计时；而且是针对每个数据帧都有自己的一个定时器，所以参数里要有帧的序列号；在同一帧的一个定时器到时之前重新调用该函数，说明重传这帧，要从头开始计时；

start_ack_timer()是对ACK回传的计时，它应该从一帧到达接收方开始，也就是调用该函数的时刻；在没有超时之前，重新调用该函数，一定是其它帧的ACK

回传计时，所以不影响原来的计时。

对等协议实体之间的流量控制：

我们小组认为，通过控制每个站上下层软件实体之间的数据流量，每个站的发送流量就已经间接的得到了控制。网络层向链路层发送分组受到窗口大小的控制，而且链路层给物理层发送一帧，也必须等到物理层排队序列小于50字节才可以。由于网络分组都是256字节长，而且双方的窗口大小一样，就可以让它们的通信流量保持在相同的稳定值，从而实现两个站之间的对等实体流量控制。

与标准协议的对比：

Question:如果现实中有两个相距5000公里的站点要利用你所设计的协议通过卫星信道进行通信，还有哪些问题需要解决？实验协议离实用还有哪些差距？你觉得还需要增加哪方面的功能？从因特网或其他资料查 阅 LAPB 相关的协议介绍和对该协议的评论，用成熟的 CCITT 链路层协议标准对比你所实现的实验性协 议，实验性协议的设计还遗漏了哪些重要问题？：

Answer: 如果是相距很远的卫星通信，还要考虑到的因素主要是信道的因素，包括信道的连通性，是否有可能会出现卫星正在偏离通信范围等；还有信道的稳定性，误码率和延迟的波动范围等，根据这些因素可能需要考虑动态调整协议中的参数（包括ACK_TIMER、DATA_TIMER等）。

六、实验总结心得体会

(1)完成本次实验的实际上机调试时间是多少？

我们小组采用三人合作的形式，三个人首先各自进行Go-Back-N协议的代码书写。这方面主要是按照课本所给的协议进行加工，然后将课本中没有的NAK机制加入了其中。两个协议的代码一共花费大概2天的书写和调试工作。最终我们经过讨论，吸取对方的优点，整合出一份最佳的协议程序，又花费了3天左右的时间进行参数的调整和测试，最终达到了比较好的通信效果。

(2)编程工具方面遇到了哪些问题？包括Windows环境和VC软件的安装问题。

我们小组在这方面进展比较顺利，未遇到任何问题。

(3)编程语言方面遇到了哪些问题？包括C语言使用和对C语言操控能力上的问题。

由于我们学习使用C语言时间较长，运用起来并没有什么阻碍。但由于通信协议在调试方面与传统的程序有所不同，所以在书写错误的时候往往很难检查。编译通过后运行的时候程序崩溃，这给我们修改程序bug带来了不少困难。

(4)协议方面遇到了哪些问题？包括协议机制的设计错误，发现协议死锁，或者不能正确工作，协议参数的调整等问题。

由于我们缓冲区判断满溢的条件存在问题，缓冲区溢出时网络层依然没有被关闭，导致运行程序时发现数据帧“丢失”；在协议参数的选择上也花了较长时间。

(5)开发库方面遇到了哪些问题？包括库程序中的BUG，库函数文档不够清楚导致误解，库函数在所提供的功能结构上的缺憾导致编程效率低下。这些问题或建议影响不同模块之间功能界限的划分。

无法看到库函数的实现部分的代码，对所给函数的每个参数含义的理解上在开始的时候有一些偏差，所以掌握这些函数，并且可以熟练的运用上面，花了一些时间。使用时也因为对这些封装好的函数理解不到位，出现过一些错误。

(6)总结本次实验，你在C语言方面，协议软件方面，理论学习方面，软件工程方面等哪些方面上有所提高？

通过这次实验，我们对数据链路层的选择重传协议和Go-back-N协议的机制有了很深刻的了解。此外，我们还将选择重传协议中所用到的no_nak变量引入了Go-Back-N协议程序中，大大提升了传输效率。在此前很多书本上不是特别了解的知识，在我们进行实验的过程中，通过对模拟结果的分析与思考都有了较为深刻的理解。这次实验是我们第一次自己动手实现模拟通信，让我们学会了windows下观察网络收发数据包模拟环境的搭建，对我们来说是一个非常良好的开端。

七、源程序清单

① Go-Back-N Protocol

```
/*********************************************************
* Course Name: Computer Networks                         *
* @Name: Chen Bin  ( ID: 2017211661)                     *
*        Luo Yang  ( ID: 2017211681)                     *
*         Li Yiyang ( ID: 2017211666)                     *
* @Teacher: Wang Xiaoru      @Class Number: 2017211318   *
* ------------------------------------------------------ *
* Experiment 2 : Design and Implementation of Sliding    *
*                  Window Protocol in Data Link Layer    *
* ------------------------------------------------------ *
* Protocol 5 : Go Back N                                 *
*********************************************************/

#include <stdio.h>
#include <string.h>
#include "protocol.h"

#define DATA_TIMER  2000
#define ACK_TIMER 300
#define MAX_SEQ 7

#define FRAME_DATA 1
#define FRAME_ACK 2
#define FRAME_NAK 3

#define inc(x) x = (x + 1)%(MAX_SEQ + 1)
int no_nak = 1;
```

```
typedef unsigned char seq_nr;
typedef unsigned char packet[PKT_LEN];
typedef struct
{
    unsigned char kind;
    seq_nr ack;
    seq_nr seq;
    packet data;
    unsigned int crc;
}FRAME;

/*
    DATA Frame
    +=========+========+========+===============+========+
    | KIND(1) | ACK(1) | SEQ(1) | DATA(240~256) | CRC(4) |
    +=========+========+========+===============+========+

    ACK Frame
    +=========+========+========+
    | KIND(1) | ACK(1) | CRC(4) |
    +=========+========+========+

    NAK Frame
    +=========+========+========+
    | KIND(1) | ACK(1) | CRC(4) |
    +=========+========+========+
*/

seq_nr frame_nr = 0,
```

```
        ack_expected = 0,
        nbuffered = 0,
        frame_expected = 0;

packet buffer[MAX_SEQ + 1];

static int phl_ready = 0;

static void put_frame(unsigned char *frame, int len)
{
    *(unsigned int *)(frame + len) = crc32(frame, len);
    send_frame(frame, len + 4);
    phl_ready = 0;
}

static int between(unsigned char a, unsigned char b, unsigned char c)
{
    return ((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a));
}

static void send_data(unsigned char frame_nr, unsigned char frame_expected, unsigned char
buffer[MAX_SEQ + 1][PKT_LEN])
{
    FRAME s;
    s.kind = FRAME_DATA;
    s.seq = frame_nr;
    s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
    memcpy(s.data, buffer[frame_nr], PKT_LEN);
    put_frame((unsigned char *)&s, 3 + PKT_LEN);
    start_timer(frame_nr, DATA_TIMER);
```

```
    stop_ack_timer();
    dbg_frame("Send DATA %d %d, ID %d\n", s.seq, s.ack, *(short *)s.data);
}

static void send_nak_frame(unsigned char frame_expected)
{
    FRAME s;
    s.kind = FRAME_NAK;
    s.ack = frame_expected;
    put_frame((unsigned char *)&s, 2);
    no_nak = 0;
    stop_ack_timer();
    dbg_frame("Send NAK  %d\n", s.ack);
}

int main(int argc, char **argv)
{
    int len = 0, arg, event;
    FRAME r;
    protocol_init(argc, argv);
    disable_network_layer();
    lprintf("Designed by Chen Bin, Luo Yang and Li Yiyang. Build: " __DATE__"
"__TIME__"\n");
    for(;;)
    {
        event = wait_for_event(&arg);
        switch (event)
        {
        case NETWORK_LAYER_READY:
            nbuffered++;
```

```
        get_packet(buffer[frame_nr]);
        send_data(frame_nr, frame_expected, buffer);
        inc(frame_nr);
        break;

    case PHYSICAL_LAYER_READY:
        phl_ready = 1;
        break;

    case FRAME_RECEIVED:
        len = recv_frame((unsigned char *)&r, sizeof r);
        if (len < 5 || crc32((unsigned char *)&r, len) != 0)
        {
            if (no_nak && r.seq == frame_expected)
                send_nak_frame(frame_expected % (MAX_SEQ + 1));
            dbg_event("**** Receiver Error, Bad CRC Checksum\n");
            break;
        }

        if (r.kind == FRAME_DATA && r.seq == frame_expected)
        {
            dbg_frame("Recv DATA %d %d, ID %d\n", r.seq, r.ack, *(short *)r.data);
            put_packet(r.data, len - 7);
            start_ack_timer(ACK_TIMER);
            no_nak = 1;
            inc(frame_expected);
        }

        else if (r.kind == FRAME_ACK)
            dbg_frame("Recv ACK %d \n", r.ack);
```

```
    else if (r.kind == FRAME_NAK)
    {
        dbg_frame("Recv NAK %d\n", r.ack);
        frame_nr = r.ack;
        r.ack = (r.ack + MAX_SEQ) % (MAX_SEQ + 1);
        while (between(ack_expected, r.ack, frame_nr))
        {
            nbuffered--;
            stop_timer(ack_expected);
            inc(ack_expected);
        }
        for (int i = 0; i < nbuffered; i++)
        {
            send_data(frame_nr, frame_expected, buffer);
            inc(frame_nr);
        }
        break;
    }

    while (between(ack_expected, r.ack, frame_nr))
    {
        stop_timer(ack_expected);
        nbuffered--;
        inc(ack_expected);
    }
    break;

case ACK_TIMEOUT:
{
```

```
            FRAME s;
            s.kind = FRAME_ACK;
            s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
            put_frame((unsigned char *)&s, 2);
            dbg_frame("Send ACK  %d\n", s.ack);
            stop_ack_timer();
            break;
        }

        case DATA_TIMEOUT:
            dbg_event("---- DATA %d timeout\n", arg);
            frame_nr = ack_expected;
            for (int i = 0; i < nbuffered; i++)
            {
                send_data(frame_nr, frame_expected, buffer);
                inc(frame_nr);
            }
            break;
        }
        if (nbuffered < MAX_SEQ && phl_ready)
            enable_network_layer();
        else
            disable_network_layer();
    }
}
```

② Selective Repeat Protocol

```
/*************************************************************
* Course Name: Computer Networks                             *
* @Name: Chen Bin  ( ID: 2017211661)                         *
*        Luo Yang  ( ID: 2017211681)                         *
*         Li Yiyang ( ID: 2017211666)                         *
* @Teacher: Wang Xiaoru      @Class Number: 2017211318       *
* ---------------------------------------------------------- *
* Experiment 2 : Design and Implementation of Sliding        *
*                  Window Protocol in Data Link Layer        *
* ---------------------------------------------------------- *
* Protocol 6 : Selective                                     *
*************************************************************/


#include <stdio.h>
#include <string.h>
#include "protocol.h"


#define DATA_TIMER  3000
#define ACK_TIMER 300
#define MAX_SEQ 15


#define NR_BUFS ((MAX_SEQ + 1) / 2)
#define FRAME_DATA 1
#define FRAME_ACK 2
#define FRAME_NAK 3
#define inc(x) x = (x + 1) % (MAX_SEQ + 1)


typedef enum { false, true } bool;
```

```
typedef unsigned char seq_nr;
typedef unsigned char packet[PKT_LEN];
typedef struct
{
    unsigned char kind;
    seq_nr ack;
    seq_nr seq;
    packet data;
    unsigned int crc;
}frame;




/*
    DATA Frame
    +=========+========+========+===============+========+
    | KIND(1) | ACK(1) | SEQ(1) | DATA(240~256) | CRC(4) |
    +=========+========+========+===============+========+

    ACK Frame
    +=========+========+========+
    | KIND(1) | ACK(1) | CRC(4) |
    +=========+========+========+

    NAK Frame
    +=========+========+========+
    | KIND(1) | ACK(1) | CRC(4) |
    +=========+========+========+
*/
```

```
static int phl_ready = 0;

bool no_nak = true;


static int between(seq_nr a, seq_nr b, seq_nr c)

{

    return((a <= b && b < c) || (c < a && a <= b) || (b < c && c < a));

}


static void put_frame(unsigned char *frame, int len)

{

    *(unsigned int *)(frame + len) = crc32(frame, len);

    send_frame(frame, len + 4);

    phl_ready = 0;

}


static void send_data(unsigned char fk, seq_nr frame_nr, seq_nr frame_expected, packet

buffer[])

{

    frame s;

    s.kind = fk;

    s.seq = frame_nr;

    s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);


    if (fk == FRAME_DATA)

    {

        memcpy(s.data, buffer[frame_nr % NR_BUFS], PKT_LEN);

        put_frame((unsigned char *)&s, 3 + PKT_LEN);

        start_timer(frame_nr % NR_BUFS, DATA_TIMER);

        dbg_frame("Send DATA %d %d, ID: %d\n", s.seq, s.ack, *(short*)&(s.data));
```

```
    }
    if (fk == FRAME_NAK)
    {
        put_frame((unsigned char *)&s, 2);
        no_nak = false;
        dbg_frame("Send NAK %d\n", s.ack);
    }
    if (fk == FRAME_ACK)
    {
        put_frame((unsigned char *)&s, 2);
        dbg_frame("Send ACK %d\n", s.ack);
    }
    stop_ack_timer();
}

int main(int argc, char **argv)
{
    seq_nr next_frame_to_send = 0,
           ack_expected = 0,
           frame_expected = 0,
           too_far = NR_BUFS,
           nbuffered = 0,
           i;

    packet out_buf[NR_BUFS],
           in_buf[NR_BUFS];

    bool arrived[NR_BUFS];

    int event, arg, len = 0;
```

```
    frame r;

    for (i = 0; i < NR_BUFS; i++)
        arrived[i] = false;

    protocol_init(argc, argv);
    lprintf("Designed by Chen Bin, Luo Yang and Li Yiyang. Build: " __DATE__ "
" __TIME__ "\n");
    enable_network_layer();

    for (;;)
    {
        event = wait_for_event(&arg);

        switch (event)
        {
        case NETWORK_LAYER_READY:
            get_packet(out_buf[next_frame_to_send % NR_BUFS]);
            nbuffered++;
            send_data(FRAME_DATA, next_frame_to_send, frame_expected, out_buf);
            inc(next_frame_to_send);
            break;

        case PHYSICAL_LAYER_READY:
            phl_ready = 1;
            break;

        case FRAME_RECEIVED:
            len = recv_frame((unsigned char *)&r, sizeof r);
```

```
            if (len < 5 || crc32((unsigned char *)&r, len) != 0)
            {
                if (no_nak)
                    send_data(FRAME_NAK, 0, frame_expected, out_buf);
                dbg_event("**** Receiver Error, Bad CRC ******\n");
                break;
            }
            if (r.kind == FRAME_DATA)
            {
                if ((r.seq != frame_expected) && no_nak)
                    send_data(FRAME_NAK, 0, frame_expected, out_buf);
                else
                    start_ack_timer(ACK_TIMER);
               if (between(frame_expected, r.seq, too_far) && (arrived[r.seq % NR_BUFS]
== false))
                {
                    arrived[r.seq % NR_BUFS] = true;
                    memcpy(in_buf[r.seq % NR_BUFS], r.data, PKT_LEN);

                    while (arrived[frame_expected % NR_BUFS])
                    {
                        put_packet(in_buf[frame_expected % NR_BUFS], len - 7);
                        no_nak = true;
                        arrived[frame_expected % NR_BUFS] = false;
                        inc(frame_expected);
                        inc(too_far);
                        start_ack_timer(ACK_TIMER);
                    }
                    dbg_frame("Recv  DATA  %d  %d,  ID:  %d  \n",  r.seq,  r.ack,
*(short*)&(r.data));
```

```
                }
            }
            else if ((r.kind == FRAME_NAK) && between(ack_expected, (r.ack + 1) %
(MAX_SEQ + 1), next_frame_to_send))
            {
                send_data(FRAME_DATA, (r.ack + 1) % (MAX_SEQ + 1), frame_expected,
out_buf);
                dbg_frame("Recv NAK %d\n",r.ack);
            }

            while (between(ack_expected, r.ack, next_frame_to_send))
            {
                stop_timer(ack_expected % NR_BUFS);
                nbuffered--;
                inc(ack_expected);
            }
            break;

        case DATA_TIMEOUT:
            send_data(FRAME_DATA, ack_expected, frame_expected, out_buf);
            dbg_event("******** DATA %d timeout **********\n", arg);
            break;

        case ACK_TIMEOUT:
            send_data(FRAME_ACK, 0, frame_expected, out_buf);
            dbg_event("******** ACK %d timeout *********\n", arg);
            break;
        }

        if (nbuffered < NR_BUFS&& phl_ready)
```

```
            enable_network_layer();
        else
            disable_network_layer();
    }
}
```