数据链路层

滑动窗口协议的设计与实现

（计算机网络实验一）

实验报告

GBN 协议 & SR 协议

网络工程 14 班

实验组号 14

罗暄澍 2015211527 06

周飞宇 2015211534 13

一、实验内容和实验环境描述

实验内容

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。使用如下信道模型：

8000BPS 全双工卫星信道

单向传播时延 270 毫秒

信道误码率为 10^{-5}

物理层接口：提供帧传输服务，帧间有 1MS 帧边界

网络层属性：分组长度固定 256 字节

本次实验中，小组人员通过讨论和分工分别实现了 Go-Back-N 协议和 SR 协议，两者均使用了带 ACK 帧和 NAK 帧的反馈机制。通过计算机模拟仿真实验比较了两种协议在不同信道条件下的信道利用率，分析了两种协议在不同情况下的综合性能。

实验环境

操作系统：Windows 10 15063.296 version1703

编译器：Microsoft Visual Studio 2017

二、协议设计

成帧方法采用字节填充的标志字节法。每一帧都用一些特殊的字符 FLAG 作为开始和结束的边界。当有效载荷中含有标志字节或转义字节时，在每个在有效载荷中出现的标志字节或转义字节前加一个 ESC 转义字节。接收端的数据链路层在将数据送给网络层之前删掉这些转义字符。

协议工作时，通过程序建立的两个站点 A、B 通过互发数据包交换数据，而控制讯息则捎带在数据讯息中传递。当出现帧丢失时，如收到帧的序号有跳跃，或者出现 CRC 校验出错丢弃了某帧，会主动发送 NAK 否定帧。若长期未产生反向数据帧，则出现 ACK 超时事件，主动发送 ACK 帧提示确认，对方收到确认后，滑动窗口继续发送，若一直未收到确认讯息，则出现数据帧超时事件，发送方会自动重发相应的数据帧。

帧中各个字段的定义和编码

- **DATA Frame**

```
+=========+========+========+===============+========+
| KIND(1) | SEQ(1) | ACK(1) | DATA(240~256) | CRC(4) |
+=========+========+========+===============+========+
```

- **ACK Frame**

```
+=========+========+=========+
```

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

- **NAK Frame**

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

三、软件设计

数据结构

(1) 帧格式

```
struct FRAME {
      unsigned char kind; //FRAME_DATA/FRAME_ACK/FRAME_NAK
      unsigned char ack;  //ACK序号
      unsigned char seq;   //数据帧序号
      unsigned char data[PKT_LEN];
      unsigned int  padding;
};
```

(2) GBN 算法中定义的常量

```
#define ACK_TIMER 300    //ACK计时器
#define MAX_SEQ 7        //最大发送序号
#define DATA_TIMER 2800 //重发计时器
static int phl_ready = 0;                    //物理层
static int no_nak = 1;                       //NAK标记
static unsigned char ack_expected = 0;       //发送窗口下界
static unsigned char next_frame_to_send = 0;  //发送窗口上界 + 1
static unsigned char frame_expected = 0;     //接收窗口
static unsigned char nbuffered = 0;          //当前发送缓存数目,nbuffered必须小于等于MAX_SEQ
static unsigned char out_buf[MAX_SEQ + 1][PKT_LEN], in_buf[PKT_LEN]; //发送缓存和接收缓存
```

(3) SR 算法中定义的常量

```
#define DATA_TIMER  5000 //Data帧超时时间
#define ACK_TIMER 300    //Ack帧超时时间
#define MAX_SEQ 31       //帧的序号空间，应当是2^n-1
#define NR_BUFS 16      //窗口大小，NR_BUFS=(MAX_SEQ+1)/2
#define inc(k) if(k < MAX_SEQ)k++;else k=0       //计算k+1
```

模块结构

(1) 共同模块

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

该函数用于判断帧是否在窗口中。

- 若 a 代表接收窗口的上界(frame_expected)，c 代表接收窗口的下界(too_far)，则 b 代表当前数据帧的序号(r.seq)。
- 若 a 代表当前发送窗口的上界(ack_expected)，c 代表发送窗口的下界(next_frame_to_send)，则 b 代表确认序号(r.ack)

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

该函数用于添加4位 CRC 校验和以及向物理层发送帧

(3)protocol.h 头文件中的相关定义和函数原型：

```
/* Network Layer functions */
#define PKT_LEN 256
void enable_network_layer(void);
void disable_network_layer(void);
int get_packet(unsigned char *packet);
void put_packet(unsigned char *packet, int len);
/*Physical Layer funcitons*/
void send_frame(unsigned char *frame, int len);
int recv_frame(unsigned char *buf, int size);
int phl_sq_len(void);
```

(3) GBN中的独有模块

```
static void send_4frame(unsigned char frame_kind,
                        unsigned char frame_nr,
                        unsigned char frame_expected,
                        unsigned char buffer[][PKT_LEN])
```

该函数用于在不同种情况时发送不同帧的控制

具体解释如下：

frame_kind指的是帧的种类。根据不同帧种类，执行不同的操作。

frame_nr，frame_expected，buffer[]，分别代表当前数据帧的序号位，ack 值和数据信息。

若 frame_kind==FRAME_DATA：当主函数中为 NETWORK_LAYER_READY 事件时，在接收到网络层传来的数据之后调用该函数，此函数调用put_frame()函数实现数据帧的帧头和校验和的添加和发送数据帧。

若 frame_kind==FRAME_ACK：虽然实验中所采用的帧格式均携带ACK的部分(`unsigned char ack;`)，但是在ACK计时器超时要求重发时，仍然需要重发单独的ACK帧，此时即可重发真个是较短的单独ACK帧。

若 frame_kind==FRAME_NAK：将no_nak标记更新，剩下的操作与发送ACK帧时相同，要继续调用put_frame()函数为NAK帧添加校验并发送到物理层进行传输。

(4) SR中的独有模块

```
static void send_data_frame(unsigned char frame_nr);
static void send_ack_frame(void);
static void send_nak_frame(void);
```

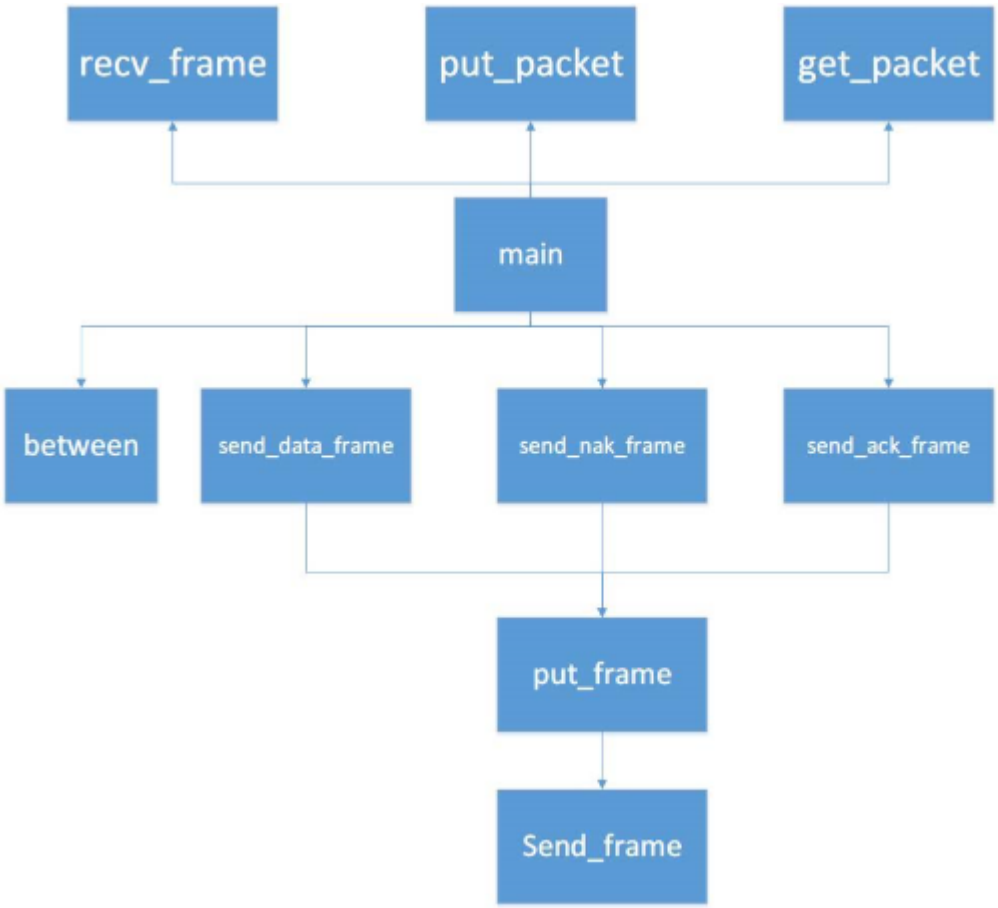
以上三个函数用于将三种不同类型的帧发送至物理层。

其中 frame_nr 表示当前帧序号，上述三个函数均是将

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

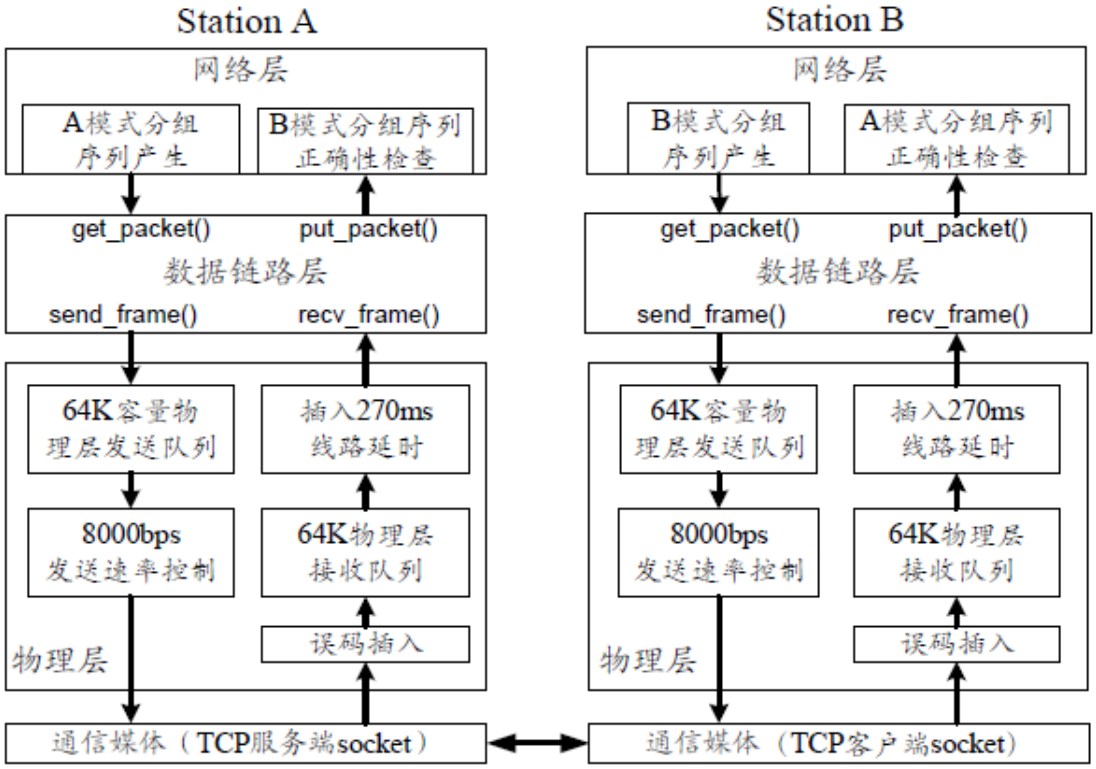
进行封装，根据不同的帧格式调用不同的函数进行发送。三种函数的内部机制和GBN中对三种不同帧的处理方式相同，在此就不再赘述。

各个模块之间的调用如下图：



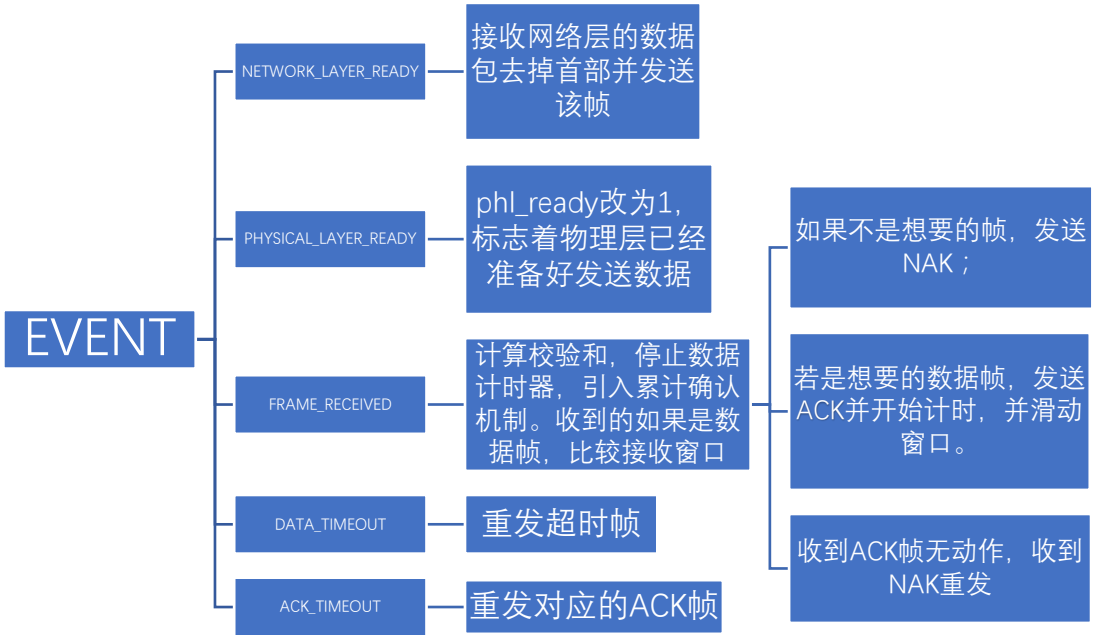
算法流程

程序的总体结构：

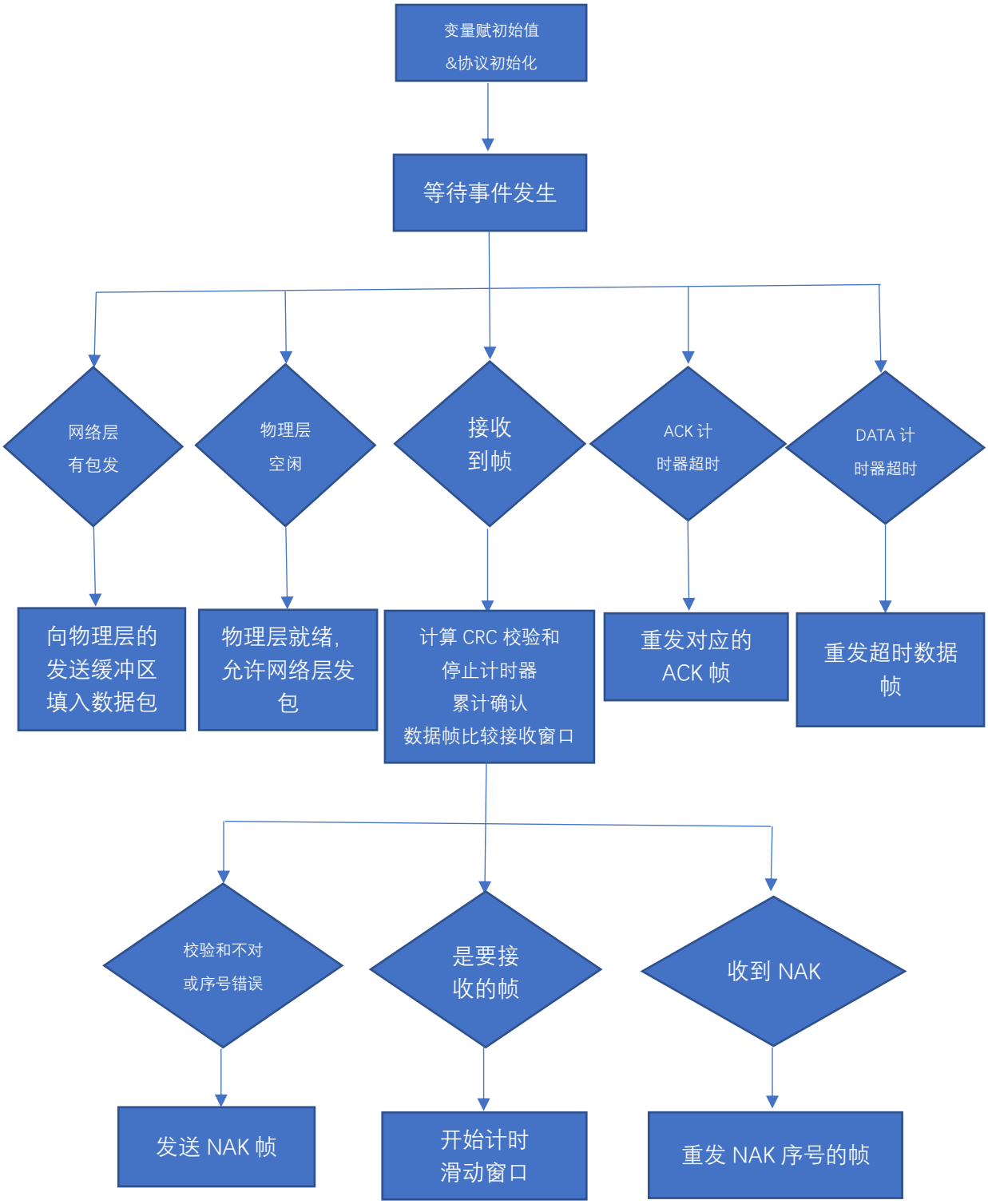


数据链路层的具体算法流程:

（1）GNB 协议



（2）SR 协议


变量赋初始值
&协议初始化
等待事件发生
网络层
有包发
物理层
空闲
接收
到帧
ACK 计
时器超时
DATA 计
时器超时
向物理层的
发送缓冲区
填入数据包
物理层就绪，
允许网络层发
包
计算 CRC 校验和
停止计时器
累计确认
数据帧比较接收窗口
重发对应的
ACK 帧
重发超时数据
帧
校验和不对
或序号错误
是要接
收的帧
收到 NAK
发送 NAK 帧
开始计时
滑动窗口
重发 NAK 序号的帧

（3）GBN 与 SR 协议在算法上的区别;

- 窗口大小：

  GBN：发送方窗口大小小于等于最大序号值

  接收方窗口大小为 1

  SR：$Ws+Wr <= 2^n$

  $Ws>=Wr$

  $Ws,Wr<= 2^{n-1}$

- 收到错帧或数据包丢失后的重传策略：

  GBN：退回到错误帧，重新传送错帧或丢失分组后所有的帧。

  SR：只重发丢失或错误的分组

四、实验结果分析

(1)在 GBN 和 SR 两种滑动窗口机制下，均通过进行 CRC 校验和计时器计时重传的方法实现了在有差错信道环境下的无差错传输。

(2)协议中使用 CRC 校验，ACK 和 NAK 机制提高了程序的容错性和健壮性。通过测试程序在不同模式下，均能可靠的正常运行超过两小时，虽未进行更长时间的测试，但可认为程序能可靠地长时间运行。

(3)协议参数的选择：

经过反复测试、比较和调整，最终我们选定以下参数：

GBN：

MAX_SEQ=7

ACK_TIMER=300 ms

DATA_TIMER=5000 ms

SR：

MAX_SEQ=31

NR_BUFS=16

ACK_TIMER=300 ms

DATA_TIMER=5000 ms

在选定以上参数后，实验测得的数值较为接近参考的理论数值。

(4)**题设环境下的窗口大小的理论分析**

题设条件如下：

8000BPS 全双工卫星信道

单向传播时延 270 毫秒

信道误码率为 10^{-5}

物理层接口：提供帧传输服务，帧间有 1MS 帧边界

网络层属性：分组长度固定 256 字节

我们采用的是 ACK/NAK 的反馈机制故而效率公式如下：

$$\delta = \frac{N}{1+\alpha}$$

其中完整的一帧包括有效数据 256B，Kind 1B，SEQ 1B，ACK 1B，CRC 校验 4B。

$$\alpha = \frac{\text{传输时延}}{\text{发送时延}} = \frac{270 \times 2}{\frac{(256+1+1+1+4)\times 8}{8000}} \approx 2.1$$

代入数据我们可以发现，为使滑动窗口的效率达到 100%，窗口 N 应当满足：

$$N \geq 1 + 2.1 \approx 4$$

但由于实际信道的误码率以及 B 站在发送时采取的 100s 发送，100s 停止发送的策略，实际的窗口大小应高于该理论值，相应地，计时器的超时时限也应当提高。而对于 ACK_TIMER，由于 ACK 帧的长度较短，且能做到即时发送，其值只需略高于信道的单向传输时延即可，此处选 300ms。

(5)信道利用率的进一步理论分析

- 无误码信道的信道利用率$\eta = \frac{256}{256+1+1+1+4} \times 100\%$=97.3%
- 存在误码的信道:
  假设在重传过程中 ACK 帧和重传的数据帧能够即时且正确传输，即不存在二次传输。

  当信道误码率为10^{-5}时，$\frac{100000}{263\times 8} \approx 48$。即平均每 48 帧存在一帧错误，则：

$$\eta = \frac{48}{48+1} = 95.3\%$$

  其他误码率时，计算方法类似。

“性能测试记录表”，实验结果分析

表 1 命令行单字母选项功能表

选项	功能
a/b	指定启动的新进程是站点 A 还是站点 B
u	(Utopia)设置本站接收方向的信道误码率为 0。默认值为 1.0e-5
f	(Flood)洪水模式。本站网络层有无穷量分组流及时供应给数据链路层发送。默认情况下，站点 A 以较平缓方式断断续续不停产生分组流，站点 A 以 100 秒为周期发一段停一段
i	(Idle-...)该选项仅对站点 B 有意义。站点 B 默认工作方式下，第一个 100 秒有数据发送，第二个 100 秒极少数据发送，第三个 100 秒又有数据发送，.....，如此 BUSY-IDLE-BUSY - IDLE-.....反复。指定 i 选项后，站点 B 的分组序列产生模式与此相反，以 IDLE-BUSY-IDLE-BUSY-.....周期反复。IDLE 周期大约每 4 秒才发送一帧。
n	(No log)取消日志文件。如果需要连续 24 小时执行程序又不打算让日志信息占用磁盘空间，可以使用此选项。默认情况下，日志文件的位置和命名参见 8.2
0-3	设定 debug_mask 变量的值为 0~3 其中之一。用于控制程序中 dbg_event()和 dbg_frame()的输出。变量 debug_mask 的默认值为 0

GBN 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	无误码信道数据传输	2000			

	datalink bu			51.61	96.96	
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”	2000	47.15	88.37	
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	2000	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	2000	86.33	86.60	
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}	2000	40.15	36.88	

SR 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	51.61	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	51.48	94.02	
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	94.55	94.56	
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	56.63	57.29	

通过比较测试数据和参考数据可以发现：

实验所得的数据与参考数据基本一致，仅在误码率较高的情况下存在 5%左右的偏差，鉴于模拟实验的计算机配置不同和窗口大小以及计时器时限存在差异，可认为实验结果是基本符合预期的。

另外，应注意到，表中部分数据 AB 站效率存在的差异均是由于 AB 站发送数据采用了不同的策略造成的，部分模式中 B 站采用了以 100s 为周期的发一段停一段策略，具体情况见下表：

表 1 命令行单字母选项功能表

选项	功能
a/b	指定启动的新进程是站点 A 还是站点 B
u	(Utopia)设置本站接收方向的信道误码率为 0。默认值为 1.0e-5
f	(Flood)洪水模式。本站网络层有无穷量分组流及时供应给数据链路层发送。默认情况下，站点 A 以较平缓方式断断续续不停产生分组流，站点 A 以 100 秒为周期发一段停一段
i	(Idle-...)该选项仅对站点 B 有意义。站点 B 默认工作方式下，第一个 100 秒有数据发送，第二个 100 秒极少数据发送，第三个 100 秒又有数据发送，.....，如此 BUSY-IDLE-BUSY - IDLE-.....反复。指定 i 选项后，站点 B 的分组序列产生模式与此相反，以 IDLE-BUSY-IDLE-BUSY-.....周期反复。IDLE 周期大约每 4 秒才发送一帧。
n	(No log)取消日志文件。如果需要连续 24 小时执行程序又不打算让日志信息占用磁盘空间，可以使用此选项。默认情况下，日志文件的位置和命名参见 8.2
0-3	设定 debug_mask 变量的值为 0~3 其中之一。用于控制程序中 dbg_event()和 dbg_frame()的输出。变量 debug_mask 的默认值为 0

GBN 与 SR 适用情况比较：

通过对不同信道情形的测试不难发现：

当信道误码率高，重传率高时，采用选择重传协议可以明显提高信道利用率。

当收发双方的缓冲区较小，且信道较为稳定时 GNB，直接丢弃的策略可以节省缓冲空间，同时也不至于降低信道的利用率。

存在的问题和改进思路

存在的问题

协议中，虽然存在单独的 ACK 帧，但其实其仅用于 ACK 计时器超时后补发的 ACK，而实际上的 ACK 信息是依赖于数据帧的捎带确认，即 ACK 信息过于依赖数据帧，从而可能造成缓冲区有大量未被处理的帧，使得实际接收成功的包的 ACK 无法及时到达发送方。

改进思路

- 一次重发多个 NAK 帧。通过增加 NAK 帧的发送次数，使发送方尽快重发错误的帧，保证接收窗口中未被处理的帧的数量维持在一个较小的水平中。
- ACK 帧单独设计，不再使用捎带确认，避免因为数据帧发送的延迟导致 ACK 超时重传，降低了信道的利用率。

五、研究和探索的问题

1. CRC 检验能力

本次实验使用的 32 位的 CRC 校验码。在理论上，可以检测出：所有的奇数个错误，所有双比特错误和小于等于 32 位的突发错误。但是检测不错大于 32 位的突发错误。因此如果出现 CRC32 不能检测出的错误，至少需要出现 33 位突发错误。

可将错误发生的过程视为泊松分布，则在一帧中出现错误个数的期望为：

$$\lambda = 256 \times 8 \times 10^{-5}=0.02096$$

所以一帧中出现多于 33 个错误的概率为

$$\mathrm{P}\{33 \leq \mathrm{X} \leq 2096\} = \sum_{k=33}^{2096} \frac{\lambda^k}{k!} \times e^{-\lambda} = 4.55 \times 10^{-93}$$

这样极低的概率可看作不可能事件，因此证明了 CRC 校验的可靠性很高。

2. get_ms()和 log_printf 的实现

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。

printf 是用变长参数实现的。prinf 的函数原型为 printf(char * fmt,…),后面…即表示变长参数。通过对 fmt 字符串进行解析，将 fmt 字符串转化成 终的字符串，然后通过系统调用输出到屏幕上。

六、实验总结和心得体会

本次实验，两位小组成员从研究实验指导书，参考示例，到讨论代码实现方式和分工编写代码，测试和数据与结果的分析，用掉了一整天的时间。试验中，一开始我门忽略了老师提供的函数的结构，模仿书上的代码进行设计，发现与实验提供的 protocol.h 中的函数并不兼容，绕了不少弯路。除了编写代码中遇到的一些难点，测试时，根据结果对理论窗口值的调整同样花费了不少时间。

尽管过程中插曲不断，困难也接踵而至，但是我们小组成员通过不断讨论，得出可靠的结论，对滑动窗口协议的原理，实现过程有了更为准确深刻的理解。而我们的团队合作能力，表达交流能力也在实验的过程中又一次提升。相信今后有类似的实验我们一定会做到更好！

七、源程序清单

按照“源程序书写格式”要求的源程序书写规范，格式化源程序

打印模版：源程序清单.DOC