北京邮电大学　计算机学院

《计算机网络》实验报告

姓名　　王睿嘉

学号 2015211906

班级 2015211307

数据链路层选择重传协议的设计与实现

一、 实验内容和环境描述

1. 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上两站点无差错双工通信。

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

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后，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，需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、重传定时器时限及 ACK 搭载定时器时限。这些参数的设计，需充分理解滑动窗口协议的工作原理并利用所学的理论知识，经认真推算，计算出最优取值，并通过程序的运行进行验证。

2. 环境描述

由实验指导教师提供协议软件设计的基本程序库，利用仿真环境下所提供的物理层服务和定时器机制为网络层提供服务。

2.1 程序的总体结构

数据链路层通信的两个站点分别为 A 和 B，仿真环境利用 Win10 环境下的 TCP 协议和 Socket 客户端/服务器机制构建两个站点之间的通信。

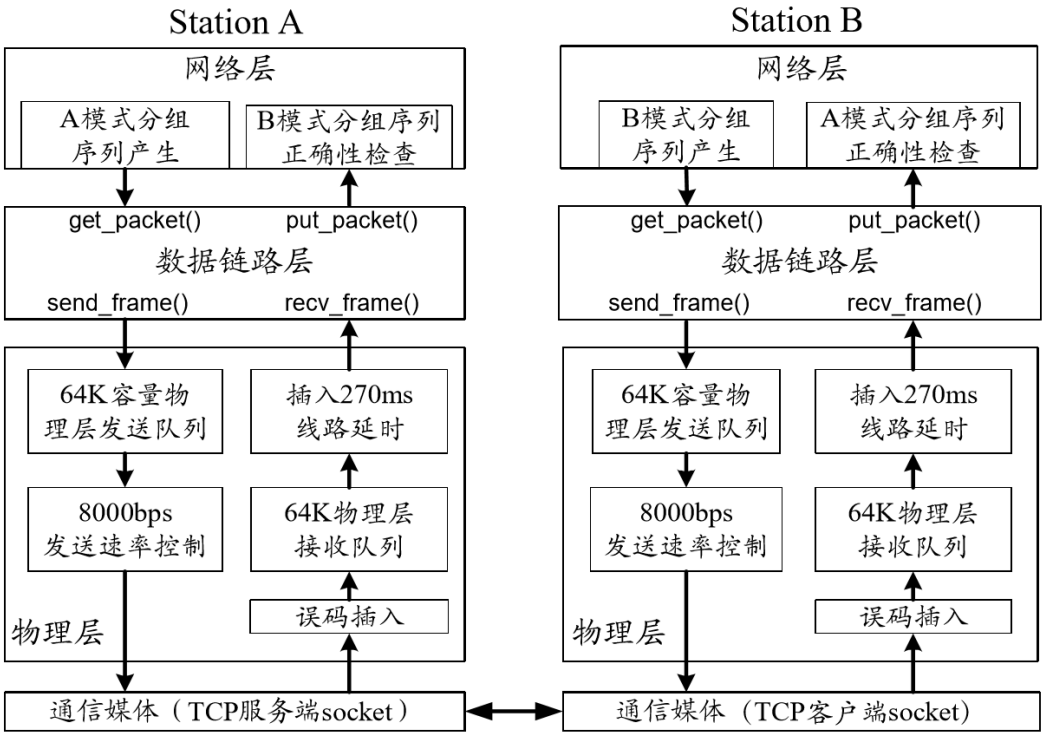
编译、链接之后生成的可执行程序（datalink.exe）为命令行程序。在两个 DOS 窗口中使用不同的命令行参数启动两个进程，分别仿真站点 A 和站点 B。
其中，站点 A 以 TCP 服务器的身份运行，默认监听 TCP 端口 59144；站点 B 以 TCP 客户端身份运行，默认连接 127.0.0.1 的 TCP 端口 59144。

可执行程序中包括物理层、数据链路层和网络层三部分内容。物理层和网络层程序由程序库提供，数据链路层程序自行设计。

物理层：为数据链路层提供 8000bps、270ms 传播延时、10^{-5} 误码率的字节流传输通道。
为仿真实现上述服务质量的信道利用在同一台计算机上 TCP socket 完成两站点间的通信。由于同一计算机上 TCP 通信传播时延短、传播速度快、没有误码，物理层仿真程序在发送端利用“令牌桶”限制发送速率；在接收端误码插入模块利用一个伪随机数“随机地”篡改从 TCP 收到的数据，使得所接收到的每个比特出现差错的概率为 10^{-5}；接收到的数据缓冲后延时 270ms 才提交给数据链路层程序，反应仿真信道的传播时延特性。为简化程序，省略了“成帧”功能，数据链路层利用接口函数 send_frame() 和 recv_frame()发送和每接收一帧。

数据链路层：通过物理层提供的帧发送和接收函数，利用物理层提供的服务，通过 get_packet()从网络层得到分组；当数据链路层成功接收到分组后，通过 put_packet()提交给网络层。

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 间交换长度固定为 256 字节的数据分组，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。



2.2 Win10 环境下编译及运行方法

指导教师提供的开发包解压后得到下图所示的目录树。（长方形表示目录，椭圆形表示文件）

protocol.h：库函数中包括的函数原型及相关的宏定义，调用库函数的 C 语言源程序应当包含此文件。

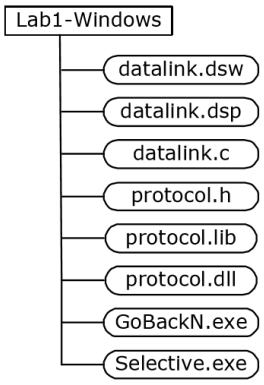
datalink.c：数据链路层程序文件。

protocol.lib：链接库文件。

protocol.dll：动态链接库。

打开工程文件 datalink.dsw 后，编辑 datalink.c，输入自己设计的程序，编译链接，生成可执行文件 datalink.exe。

协议程序运行过程中，将生成日志文件。默认情况下，日志文件放在与可执行文件相同的目录。文件名为 datalink-A.log 和 datalink-B.log，分别对应站点 A 和站点 B 的日志文件。



2.3 协议运行环境的初始化

datalink.c 中的主程序 main()必须首先调用 protocol_init(int argc, char **argv)对运行环境初始化。该函数的两个参数必须传递 main()函数的两个同名参数。目的是从命令行参数中获取站点名及某些选项以提供一种配置系统参数的手段。这些选项包括重新指定日志文件，指定 TCP 端口号、设定误码率等。当命令行中重新指定了新的参数值，默认值就不再起作用。

protocol_init()建立两个站点之间的 TCP 连接，并设定时间坐标的参考 0 点，通信的两个站点的时间坐标 0 点在建立 TCP 连接时被设置成相同的参考时间点。

2.4 与网络层模块的接口函数

protocol.h 中的相关定义和函数原型：

```
#define PKT_LEN 256
void enable_network_layer();
void disable_network_layer();
int get_packet(unsigned char *packet);
void put_packet(unsigned char *packet, int len)
```

网络层和数据链路层的约定为：数据链路层在缓冲区满等条件下无法发送分组时通过disable_network_layer()通知网络层；在能够承接新的发送任务时执行 enable_network_layer()允许网络层发送数据分组。当网络层有新的分组需要发送并且未被链路层 disable，会产生 NETWORK_LAYER_READY 事件；否则网络层自行缓冲待发送分组。

数据链路层在事件处理程序中调用 get_packet(p)将分组拷贝到指针 p 指定的缓冲区中，函数返回值为分组长度。put_packet()要求提供两个参数：存放收到分组的缓冲区首地址和分组长度。程序库在 put_packet()内部增加了统计功能。如果本次调用 put_packet()比上次调用该函数的时间间隔超过 2 秒，将给出一个接收方向的报告。

2.5 事件驱动函数

protocol.h 中的相关定义和函数原型：

```
int wait_for_event(int *arg);
#define NETWORK_LAYER_READY 0
#define PHYSICAL_LAYER_READY   1
#define FRAME_RECEIVED          2
#define DATA_TIMEOUT            3
#define ACK_TIMEOUT             4
```

wait_for_event()导致进程等待，直到一个“事件”发生。可能的事件有上述 5 种，函数返回值为上述 5 种事件之一。

参数 arg 用于获得已发生事件的相关信息，仅用于 DATA_TIMEOUT，获取产生超时事件的定时器编号。

NETWORK_LAYER_READY：网络层有待发送的分组。此事件发生后才可调用 get_packet()得到网络层待发送的下一个分组。

PHYSICAL_LAYER_READY：物理层发送队列的长度低于 50 字节。

FRAME_RECEIVED：物理层收到了一整帧。

DATA_TIMEOUT：定时器超时。

ACK_TIMEOUT：所设置的搭载 ACK 定时器超时。

2.6 与物理层模块的接口函数

protocol.h 中的相关定义和函数原型：

```
int recv_frame(unsigned char *buf, int size);
int phl_sq_len(void);
```

recv_frame()从物理层接收一帧，返回值为收到帧的实际长度。

为协调数据链路层和物理层之间的流量，采用的机制是：只要在事件处理周期内至少一次调用过 send_data_frame()函数，那么 wait_for_event()会在物理层发送队列低于 50 字节时，产生 PHYSICAL_LAYER_READY 事件。

在 PHYSICAL_LAYER_READY 事件后，如果数据链路层暂时没有需要发送的数据，记录物理层状态，当有数据需要发送时直接发送。

2.7 定时器管理

protocol.h 中的函数原型：

unsigned int get_ms(void);

void start_timer(unsigned int nr, unsigned int ms);

void stop_timer(unsigned int nr);

void start_ack_timer(unsigned int ms);

void stop_ack_timer(void)

get_ms()获取当前的时间坐标，单位为毫秒。

start_timer()用于启动一个定时器，两个参数分别为计时器的编号和超时时间值。超时发生时，产生 DATA_TIMEOUT 事件，并给出超时定时器编号。

stop_timer()中止一个定时器。

在定时器未超时之前直接对同一个编号的定时器执行 start_timer()调用，将按照新的时间设置产生超时事件。

start_ack_timer()和 stop_ack_timer()两个定时器函数为搭载 ACK 机制设置，在先前启动的定时器未超时之前重新执行 start_ack_timer()，定时器将依然按照先前的时间设置产生超时事件 ACK_TIMEOUT。

2.8 命令行选项

Usage: datalink <options> [-port <tcp-port#>] [-ber <ber>] [-log <filename>]

单字母的选项必须连写在一起。

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

除单字母选项外，另有几个选项，每个选项都需要携带一个参数：

-port：指定 TCP 通信的端口号，默认为端口 59144。

-log：指定一个日志文件名替代进程的默认日志文件。

-ber：指定信道误码率。

二、 软件设计

1. 数据结构

自定义结构体中各成员及全局变量意义如下：

```
#define DATA_TIMER  4000                    //定时器
#define MAX_SEQ     31                      //数据帧最大序号（自零始）
#define NR_BUFS     ((MAX_SEQ + 1) / 2)     //窗口及缓冲区大小
#define ACK_TIMER   2500                    //搭载 ACK 定时器


typedef struct FRAME {                      //帧结构
    unsigned char kind;                     //帧种类
    unsigned char ack;                      //搭载 ACK 序号
    unsigned char seq;                      //帧序号
    unsigned char data[PKT_LEN];            //网络层数据包内容
    unsigned int padding;                   //填充字段（CRC 校验）
}FRAME;


int no_nak = 1;                             //是否已发送过 NAK
static int phl_ready = 0;                   //物理层是否就绪

```

主函数中变量及其意义：

```
int main(int argc,char **argv) {
    unsigned char out_buf[NR_BUFS][PKT_LEN];                //发送方缓冲区
    unsigned char in_buf[NR_BUFS][PKT_LEN];                 //接收方缓冲区
    static unsigned char frame_nr = 0, nbuffered = 0;
    static unsigned char frame_expected = 0, ack_expected = 0, next_frame_to_send = 0, too_far = NR_BUFS;
    int arg, i, len = 0,event;
    int arrived[NR_BUFS];                                   //标志位数组
```

frame_nr：帧序号

nbuffered：已发送但未受到 ACK 的帧数量

frame_expected：接收方窗口下界

ack_expected：发送方窗口下界

next_frame_to_send：发送方窗口上界 +1

too_far：接收方窗口上界 +1

arg：用于获得已发生事件的相关信息

len：用于记录网络层数据包长度

event：进程所等待的事件

2. 模块结构

between：判断序号是否落在窗口或缓存区内

```
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));
}

```

b：待判断参数

a：窗口或缓存区下界；c：窗口或缓存区上界

put_frame：增加帧尾校验字段后，传递给物理层

```
static void put_frame(unsigned char *frame, int len) {
    *(unsigned int *)(frame + len) = crc32(frame, len);         //填充校验字段
    send_frame(frame, len + 4);
    phl_ready = 0;
}

```

frame：待填充校验字段的数据帧

len：帧长度

send_data_frame：根据帧种类扩充帧内容后，传递给物理层，并停止搭载 ACK 计时器

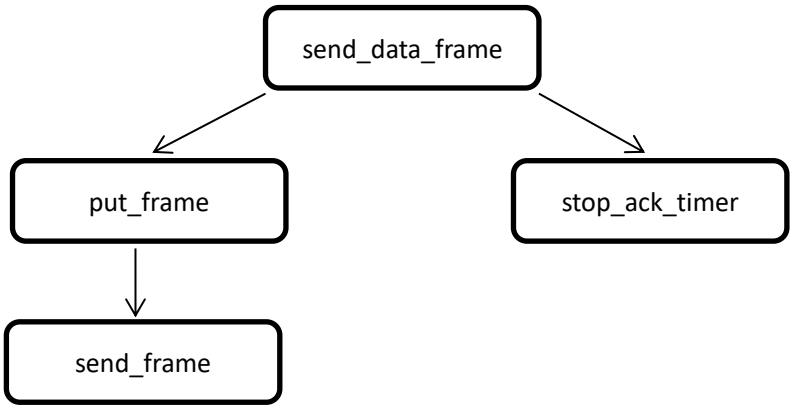
```
134    static void send_data_frame(unsigned char fk, unsigned char frame_nr, unsigned char frame_expected, unsigned char buffer[][PKT_LEN]) {
135        FRAME s;
136
137        s.kind = fk;
138        if (fk == FRAME_DATA)                   //若待发送帧为数据帧
139            memcpy(s.data, buffer[frame_nr%NR_BUFS], PKT_LEN);
140        s.seq = frame_nr;
141        s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
142        if (fk == FRAME_NAK)                    //若待发送帧为 NAK
143            no_nak = 0;
144
145        if (fk == FRAME_DATA)
146            dbg_frame("Send DATA %d %d, ID %d\n", s.seq, s.ack, *(short *)s.data);
147
148        put_frame((unsigned char *)&s, 3 + PKT_LEN);
149        if (fk == FRAME_DATA)                   //启动定时器
150            start_timer(frame_nr%NR_BUFS, DATA_TIMER);
151        stop_ack_timer();
152    }
153
```

fk：帧种类

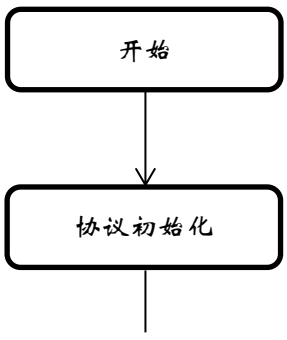
frame_nr：帧序号

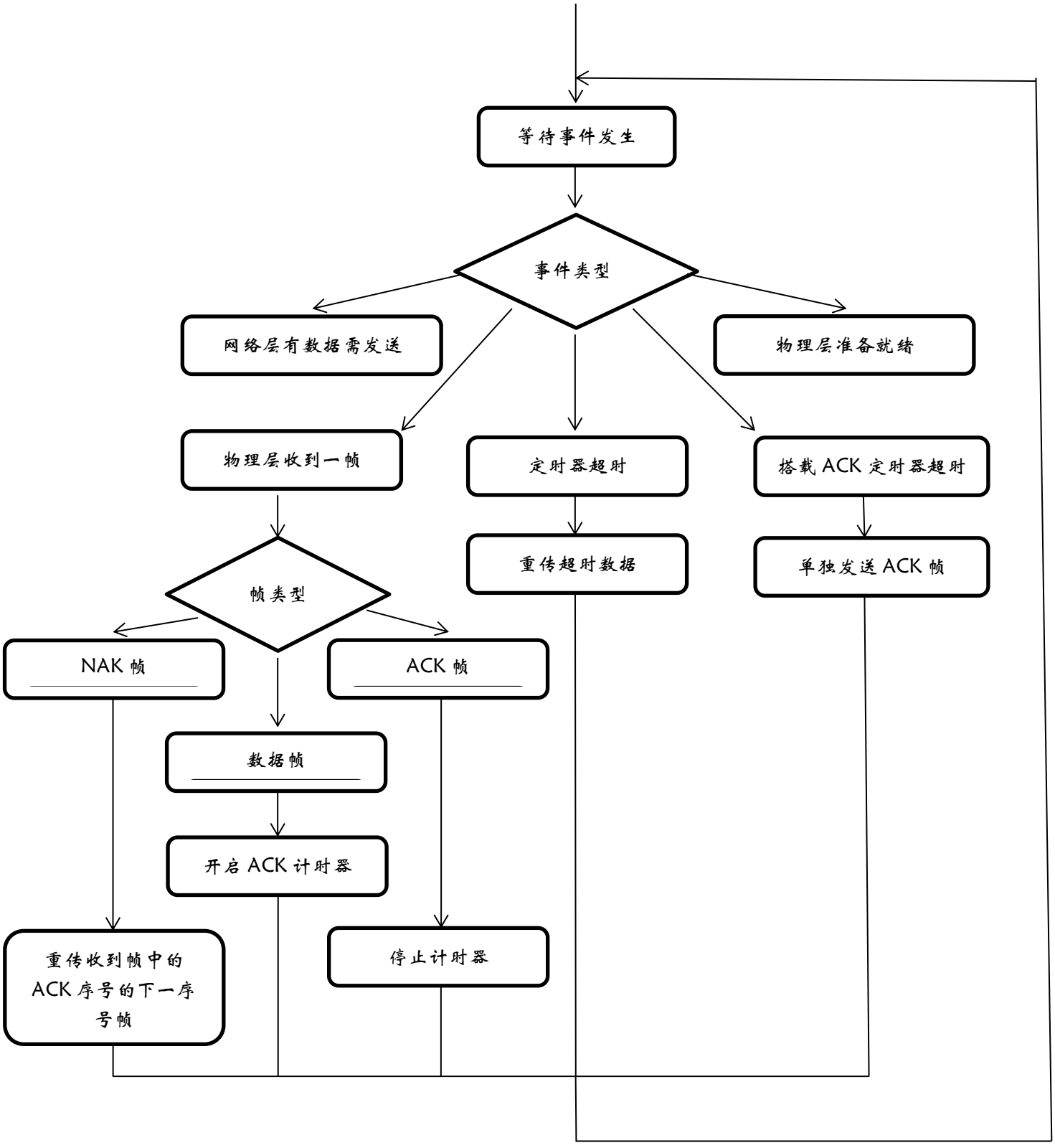
frame_expected：接收方窗口下界

buffer[][PKT_LEN]：包内容



3. 算法流程




等待事件发生
事件类型
网络层有数据需发送
物理层准备就绪
物理层收到一帧
定时器超时
搭载 ACK 定时器超时
重传超时数据
单独发送 ACK 帧
帧类型
NAK 帧
ACK 帧
数据帧
开启 ACK 计时器
重传收到帧中的 ACK 序号的下一序号帧
停止计时器

三、 实验结果分析

1. 程序健壮性

协议软件基本实现了有误码信道环境中无差错传输功能，
程序健壮性良好，能够可靠地长时间运行。

2. 协议参数的选取

滑动窗口的大小：16

物理层提供 8000bps、270ms 传播延时、10^{-5} 误码率的字节流传输通道。
数据帧最大为 1+1+1+256+4=263 字节，因而最大发送时延为 263×8÷8000×1000=263ms
传播时延与发送时延之比 a=270÷263
帧出错率 $p=1-(1-10^{-5})^{263\times8}$
设滑动窗口大小为 W，在捎带确认环境下，使信道利用率最高：

$$W>2\times(1+270\div263)\div(1-p)$$

又因 $W=2^{n-1}$（n 为帧序号位数），因而 W 至少为 8。
考虑存在一定的误码率，且 ACK、NAK 帧占用带宽，在程序中分别尝试窗口大小为 8 或 16。
窗口大小为 16 时，信道利用率较高。

重传定时器时限：4000ms

重传定时器时限设置过小，会出现大量重复数据，严重影响信道利用率，因而为保证窗口流畅滑动，
在发送完毕缓冲区一半数据后，最初帧若未收到 ACK，就应超时。
耗时 16÷2×（263+270）=4264ms
取整，约为 4000ms。

ACK 搭载定时器时限：2500ms

ACK 搭载定时器设置过小，便失去了捎带确认带来的效率的提高，因而约为重传定时器时限一半较为合适。
经程序中修改尝试，2500ms 时信道利用率较高。

3. 理论分析

无差错信道环境：

分组层能获得的最大信道利用率： 256÷(256+1+1+1+4)≈97.34%

有误码环境：

假设信道误码率为 10^{-5}，ACK 和 NAK 总能正确传输。
帧出错率 $p=1-(1-10^{-5})^{263\times8}\approx0.021$
因而，信道利用率约为 $979\times256\div(1000\times(256+1+1+1+4))\approx95.29\%$

假设信道误码率为 10^{-4}，ACK 和 NAK 总能正确传输。

帧出错率 $p=1-(1-10^{-4})^{263\times8}\approx0.19$

因而，信道利用率约为 $81\times256\div(100\times(256+1+1+1+4))\approx78.84\%$

4. 实验结果分析

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	55.62	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	53.46	91.59	
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	92.99	92.18	
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	51.85	52.18	

在误码率为 10^{-5} 的情况下，程序的实际信道利用率与理论值相接近。之所以稍有降低，是因为 ACK 和 NAK 可能丢失或出错。

在误码率为 10^{-4} 的情况下，程序的实际信道利用率与理论值相差较大。因为此种情况下 ACK 和 NAK 也很易出错，导致增加很多的超时重传，影响了效率。

5. 存在的问题

对于性能测试表中给出的所有测试方案，程序没有失败。

但误码率较高情况下效率仅有 52%左右，与参考数据有所差距。

四、 研究和探索的问题

1. CRC 校验能力

在 CRC32 中出现误码但未发现的概率是 2^{-32}。

信道模型为 8000bps 全双工卫星信道，误码率为 10^{-5}，网络层分组长度固定为 256 字节。

假定通信系统每天的使用率为 50%，则发生一次分组层误码事件，约需要 $1\div(8000\times60\times60\times24365\times50\%\times10^{-5}\times2^{-32})\approx3404.81$ 年。

若如此低的出错概率仍无法满足客户要求，则考虑使用 CRC64，代价是要增加校验位数，降低信道利用率。

2. 程序设计方面的问题

2.1 协议软件的跟踪功能

协议软件的跟踪功能对程序的调试有重要意义，只有将程序的实时状态（如收发帧、ACK、NAK 等）记录下来，当遇到 bug 时才能通过这些过程猜测可能出错的地方。

2.2 get_ms()函数

get_ms()易实现。需在连接建立时利用<time.h>中的 clock()函数获取初始时间，后调用 get_ms()时再调用 clock()获取当前时间，二者相减即可得到正确时间。

2.3 start_timer()和 start_ack_timer()

start_timer()记录数据发出后到现在经历的时间，用于数据的超时重传，因而需等数据全部发出后再开始计时。

在定时器未超时前重新执行 start_timer()，说明该序号位又有新数据发送，可推断出前一个数据必然已确认，故应重新计时。

而 start_ack_timer()记录从收到数据开始到现在经历的时间，若超时，则需发送单独 ACK。调用 start_ack_timer()时数据已全部接收，故应立即启动计时。只有有数据发出时，ack_timer()才停止计时。在先前启动的定时器未超时前重新执行 start_ack_timer()将依然按照先前的时间设置产生超时事件，因为 start_ack_timer()的再次调用说明新数据的到来而不是数据的发出。

3. 软件测试方面的问题

多种测试方案的目的是测试协议能否在各种网络环境下正常工作，从而保证可靠性。

a/b：测试误码率为 10^{-5} 情况下线路的利用率。

au/bu：测试理想信道上的利用率，即最大线路利用率。

af/bf：测试误码率为 10^{-5}，且网络层始终有数据情况下的利用率。当协议的流量控制存在问题时，会导致链路很快崩溃。

af/bf – ber 1e-4：数据流量大且误码率高，协议在流量控制和差错控制上的任何小瑕疵都会很快暴露。

4. 对等协议实体之间的流量控制

协议软件基本上解决了数据链路层对等实体间的流量控制问题。

这是通过限制数据链路层与网络层及物理层间的数据流量实现的。对两个对等体而言，在传输中的帧数量不会超过两倍窗口大小的限制，从而实现了两对等体间的流量控制。

5. 与标准协议的对比

相距很远的站点通信可能需要多个卫星中继，需要根据实际调整 ACK_TIME 和 DATA_TIME。
实际应用中，线路可能因为宕掉而无法传送数据，或因外界干扰导致误码率很高。这将导致数据计时器不断重传，直至线路恢复，而这一过程发送方并不知情。因而可设置重传计时器，当同一个数据重传超过一定次数时，便向上层返回错误信息，说明当前线路不可用，从而及时选择其他线路进行数据传输。

五、 实验总结和心得体会

1. 编程工具方面的问题

该程序为命令行调试，起初 VS 无法编译运行，经查阅资料得知需要修改编译环境。

2. 编程语言方面的问题

之前所写的程序未出现环境与 main 函数的参数交互，因而对 arg 参数的运用参考了例程和课本。
出现 bug 后，不知 debug 信息如何打印，仔细阅读实验指导书后得到解决。

3. 协议方面的问题

书中伪代码使用了 oldest_frame，每次收到 ACK 更新 oldest_frame 为(ack + 1) % NR_BUFS，实际测试中存在问题。因为不一定每次都是最先发出但未收到确认的帧超时，后参考例程，在获取事件时传入 arg，取得超时的数据计时器序号，从而解决重传问题。

4. 总结收获

加深了对选择重传协议的理解，同时也是第一次使用命令行方式完成编程测试，熟悉了一些命令行操作。