实验报告

**一、实验内容和实验环境描述**

* **实验内容**：利用所学数据链路层原理，自己设计一个选择重传滑动窗口协议，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，网络层分组长度固定为256字节。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。这些参数的设计，需要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，经过认真的推算，计算出最优取值，并通过程序的运行进行验证。通过该实验提高同学的编程能力和实践动手能力，体验协议软件在设计上各种问题和调试难度，设计在运行期可跟踪分析协议工作过程的协议软件，巩固和深刻理解理论知识并利用这些知识对系统进行优化，对实际系统中的协议分层和协议软件的设计与实现有基本的认识。
* **实验设备环境**：WindowsXP环境PC机，Microsoft Visual C++ 6.0 集成化开发环境。

**二、软件设计**

**(1) 数据结构：**

/\* Event Driver \*/

PROTOCOL\_API 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

#define PKT\_LEN 256 //分组长度

#define DATA\_TIMER 12000 //数据超时时间

#define ACK\_TIMER 1200 //ACK超时时间

#define MAX\_SEQ 15 //最大窗口序列号

#define NR\_BUFS 8 //窗口大小

typedef enum{false, true} bool; //定义布尔型变量

typedef unsigned char seq\_nr; //序号的类型

typedef struct { //分组

unsigned char data[PKT\_LEN];

} packet;

Struct FRAME { //帧

unsigned char kind; //帧类型

seq\_nr seq; //帧序号

seq\_nr ack; //ACK序号

packet info; //数据域

unsigned int padding; //CRC校验位

};

**全局变量：**

bool no\_nak = true; //是否未发送nak

static int phl\_ready = 0; //物理层是否能准备好

**主函数变量：**

int event; // event事件

int arg; //arg获取产生超时事件的定时器编号

int len = 0; // len帧长

int i; //i用于循环

seq\_nr ack\_expected; //发送窗口下界

seq\_nr next\_frame\_to\_send; //发送窗口上界

seq\_nr frame\_expected; //接收窗口下界

seq\_nr too\_far; //接收窗口上界

struct FRAME r; //帧

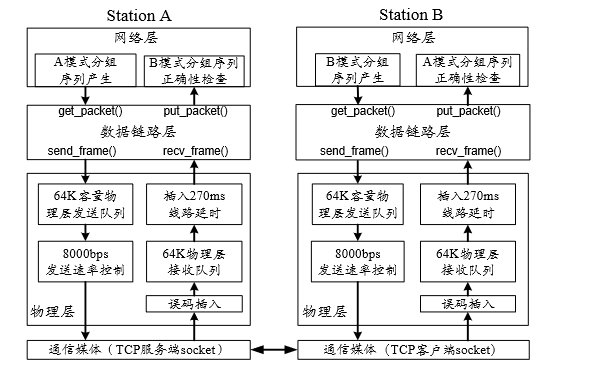
packet out\_buf[NR\_BUFS]; //发送缓存

packet in\_buf[NR\_BUFS]; //接收缓存

bool arrived[NR\_BUFS]; //标记接受缓存是否被占用

seq\_nr nbuffered; //发送窗口的大小

**(2) 模块结构：**



#define inc(k) if(k < MAX\_SEQ) k++; else k=0

//k循环增加1

static bool between(seq\_nr a, seq\_nr b, seq\_nr c);

//判断b是否落在a~c窗口内；a 接收窗口上限，c 接受窗口下限, b 收到的数据帧序号

static void put\_frame(unsigned char \*frame, int len)

//将frame所指的帧加上帧尾，送入物理层

static void send\_data\_frame(unsigned char fk, seq\_nr frame\_nr, seq\_nr frame\_expected, packet buffer[])

//发送帧；fk：帧类；frame\_nr：帧序号；frame\_expected 期待收到的数据帧 ,buffer[] 缓冲区

/\* Initalization \*/

PROTOCOL\_API void protocol\_init(int argc, char \*\*argv);

//运行环境初始化，两个参数是main的参数

/\* Network Layer functions \*/

PROTOCOL\_API void enable\_network\_layer(void);

//网络层就绪

PROTOCOL\_API void disable\_network\_layer(void);

//阻塞网络层

PROTOCOL\_API int get\_packet(unsigned char \*packet);

//将分组拷贝到指针p指定的缓冲区中

PROTOCOL\_API void put\_packet(unsigned char \*packet, int len);

//统计，存放收到分组的缓冲区首地址和分组长度

/\* Physical Layer functions \*/

PROTOCOL\_API int recv\_frame(unsigned char \*buf, int size);

/\*从物理层接收一帧，size 为用于存放接收帧的缓冲区buf 的空间大小，返回值为收到帧的实际长度。\*/

PROTOCOL\_API void send\_frame(unsigned char \*frame, int len);

/\*将内存frame 处长度为len 的缓冲区块向物理层发送为一帧，每字节发送\*/

PROTOCOL\_API int phl\_sq\_len(void);

/\*返回当前物理层队列的长度\*/

/\* CRC-32 polynomium coding function \*/

PROTOCOL\_API unsigned int crc32(unsigned char \*buf, int len);

/\* 采用的CRC 校验方案为CRC-32：x32+x26+x23+x22+x16+x12+x11+x10+x8+x7+x5+x4+x2+x1+1 \*/

/\* Timer Management functions \*/

PROTOCOL\_API unsigned int get\_ms(void);

/\*获取当前的时间坐标\*/

PROTOCOL\_API void start\_timer(unsigned int nr, unsigned int ms);

/\*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\*/

PROTOCOL\_API void stop\_timer(unsigned int nr);

/\*中止一个定时器\*/

PROTOCOL\_API void start\_ack\_timer(unsigned int ms);

/\* 启动ACK一个定时器\*/

PROTOCOL\_API void stop\_ack\_timer(void);

/\*中止一个ACK定时器\*/

/\* Protocol Debugger \*/

PROTOCOL\_API void log\_printf(char \*fmt, ...);

PROTOCOL\_API void lprintf(char \*fmt, ...);

PROTOCOL\_API char \*station\_name(void);

PROTOCOL\_API void dbg\_event(char \*fmt, ...);

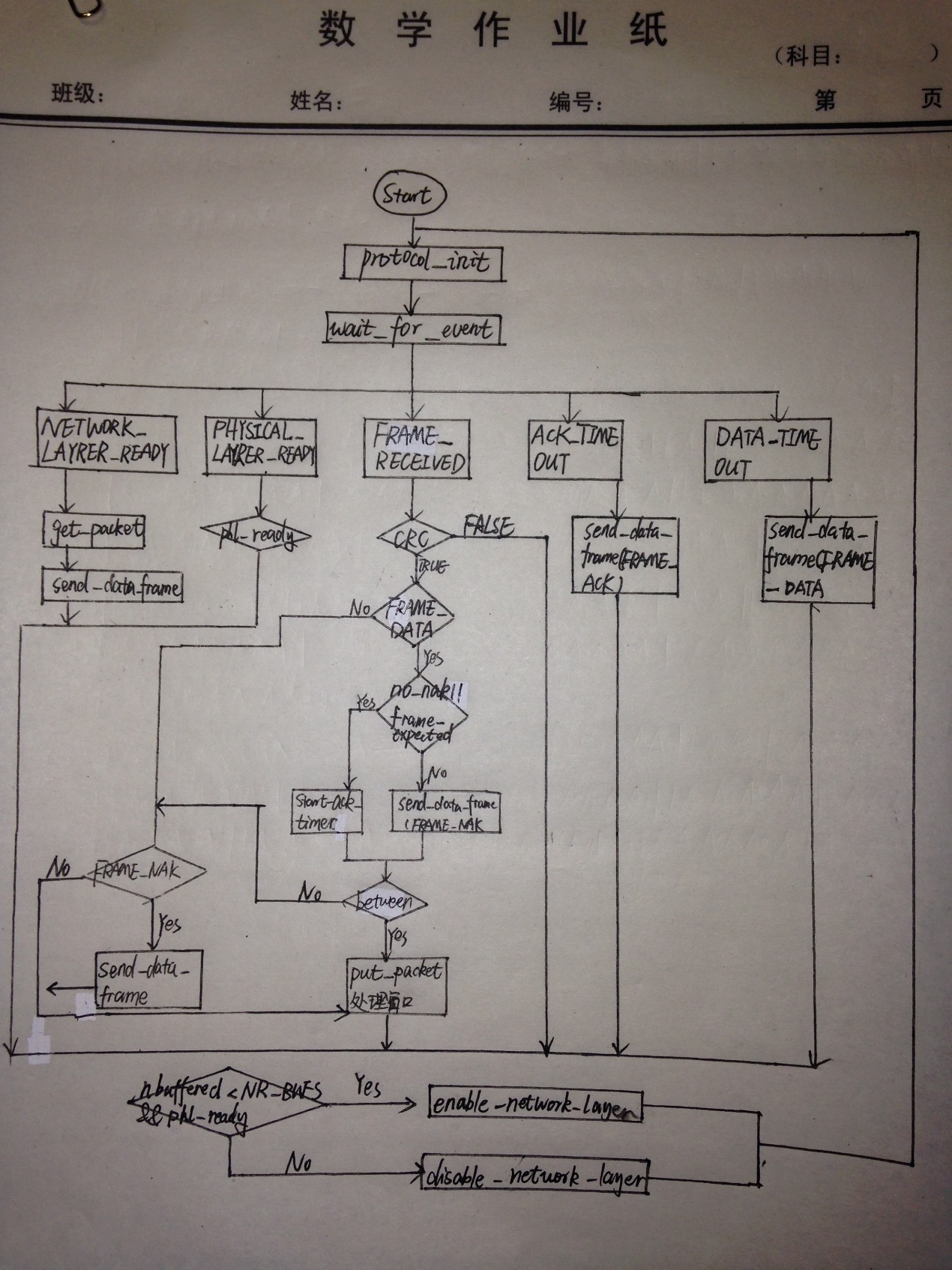
PROTOCOL\_API void dbg\_frame(char \*fmt, ...);

PROTOCOL\_API void dbg\_warning(char \*fmt, ...);

Figure 1程序调用关系图(a)

Figure 2程序调用关系图(b)

**(3) 算法流程：**



**三、实验结果分析**

**(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。**

能实现，采用了CRC校验和重传技术是错误得以被发现和纠正。

**(2) 程序的健壮性如何，能否可靠地长时间运行。**

程序健壮性强，在高负荷和高误码率等条件下均能正常工作。

**(3) 协议参数的选取：滑动窗口的大小，重传定时器的时限，ACK 搭载定时器的时限，这些参数是怎样确定的？根据信道特性数据，分组层分组的大小，以及你的滑动窗口机制，给出定量分析，详细列举出选择这些参数值的具体原因。**

由于定时器数量的限制，窗口大小最大只能设置到64，因此定时器和ACK的时限的选择就十分关键，本次实验中物理层提供了一种字节流传输服务，为了成帧，使用了字节填充技术。分组长度为 240~256字节，线路最快需要247毫秒就传输一帧，运气最不好的时候，一个256 字节分组全部充斥了需转义的字符，这样线路传输一帧，需要519毫秒左右。尽管这种极端情况很少出现，但是重传定时器的时限必须考虑这个上限，在无差错模式下，这个时限定在不太短的大小问题并不大，但在有差错的线路上，由于可能出现错误和重传，因此时限确定直接影响到线路的效率，当最初把时限定在1500时超时频率非常高，当定到一个比较高些的数字之后超时率明显下降，但若数字太大，当出现错误帧时，会影响到帧的发送效率，最后把超时时间限定在8000～12000之间，在此范围内根据不同的线路的情况选择。

**(4) 理论分析：根据所设计的滑动窗口工作机制(Go-Back-N 或者选择重传)，推导出在无差错信道环境下分组层能获得的最大信道利用率；推导出在有误码条件下重传操作及时发生等理想情况下分组层能获得的最大信道利用率。给出理论推导过程。理论推导的目的是得到信道利用率的极限数据。为了简化有误码条件下的最大利用率推导过程，可以对问题模型进行简化，比如：假定超时重传的数据帧的回馈ACK 帧可以100%正确传输，但是简化问题分析的这些假设必须不会对整个结论产生较大的误差。**

* 选择重传：
* 由于需要携带帧头和帧尾，因此最大的信息利用率为.
* 由于信道的最大比特率为8000bps，可得出每传输一个字节耗时1ms，每帧的附加讯息固定为7，耗时7ms，若出现转义字符，则可能增加时间，现在假设信道上始终有数据需要传送，这样就可以简化模型。在误码率为10-5的信道上，算下来在这100000个比特中可以传送 个数据包，则可得出每48个数据包会有一个出错，假设每出错一次，在限定时间内可以重传该帧为正确帧。则每传送48个数据包需要传送48+1=49次，此时信道利用率.
* 实际上由于程序设计的原因，当一个数据包超时后，往往要重复多次传输给数据包造成信道的浪费，若有k次重传，则信道利用率为，在此程式中，平均要重传10次，因此信道利用率降为80.6%。
* 在ESC/FLAG模式中传输256个字符需要两倍的传输空间即极限值512，此时的信道利用率的极限值是。而在10-5的错误率的信道上，利用率为，若平均每个错帧重传10次，利用率将下降为。

**(5) 实验结果分析：你的程序运行实际达到了什么样的效率，比对理论推导给出的结论，有没有差距？给出原因。有没有改进的办法？如果没有时间把这些方法付诸编程实施，介绍你的方案。**

实验所得的结果于此大致相符： 在交替的发送的停等信道中，半停半发的端口的信道利用率自然为全利用率的50%，因此在第二行中A端的利用率约为B端的一半，这在意料之中。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 | 2636 | 51.97 | 94.57 | 通过测试 |
| 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” | 1423 | 49.50 | 91.27 | 通过测试，当超时时数 据帧会被重复发送 |
| 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 | 1450 | 96.41 | 96.41 | 通过测试 |
| 4 | datalink af  datalink bf | 站点A/B的分组层都洪水式产生分组 | 1406 | 90.94 | 92.43 | 通过测试，当超时时数 据帧会被重复多次发 送，占用通道资源 |
| 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 | 1442 | 35.26 | 32.18 | 通过测试，超时时重发 的帧重复过多,占用的资 源可观 |

**(6) 存在的问题：在“表3 性能测试记录表”中给出了几种测试方案，在测试中你的程序有没有失败，或者，虽未失败，但表现出来的性能仍有差距，你的程序中还存在哪些问题?**

该程序存在的主要问题就在于会反复重传未被确认的帧，造成了信道资源的浪费，这可以在实验的结果中看出，在进行较长时间的运行后，信道流量水平趋于平稳，此时的数据和理论之大致相当，说明所确认的问题确实是症结所在。只要能限制其反复重传，就能提高信道利用率。为了解决这个问题，尝试了对协议进行了改进，使用前帧记忆识别技术，使其在超时重传时不再重复发同一帧，并改善了出错丢帧后的接受逻辑，增加ACK的发送频率，合理改善了超时时限，经过这样的改动后，预期的效果应该有所改善，但实际的效果并不明显，在信道条件较好的情况下，可以提高到接近理论值的水平如在datalink afu0 datalink bfu0的信道下，信道利用率双方都提高到了95.46%，已经很接近理论的96.2%，但在信道条件下降的情况下有时会出现死锁的情况，因此健壮性反而下降，因为窗口大小最大只有64，这是硬性限制，在信道较差的情况下很容易就用光了，所以不适用。因此最后未被采用。

**四、研究和探索问题**

**1. CRC 校验能力**

* 尽管CRC校验理论上不可能校验出所有错误，但是CRC-32的生成函数G= x32+x26+x23+x22+x16+x12+x11+x10+x8+x7+x5+x4+x2+x1+1，除了其他优良特性外，该多项式还能检测到长度小于等于32的所有突发错误，以及影响到奇数位的全部突发性错误。所不能发现的错误的几率仅为0.0047%以下，非常低，因此系统能够基本实现无差错传送。一旦发生分组出错未被检验出来的情况，网络层也能够采取相应方式进行进一步的校验，综合起来能够实现无差错的传输。
* 信道误码率1e-5,校验出错概率为0.0047%,那么产生一个分组层错误的概率q=4.7e-10,
* 平均传n次出错，由公式1/(1-p)=1/q=212765957次传输，假设客户每天传输x次共需要582920/x 年才会发生一次分组层错误。
* 如果进一步降低误码事件发生的概率，最直接的方法增加校验位的长度，比如使用crc64，代价就是冗余位进一步增大，降低了信息传输的效率。

**2. CRC 校验和的计算方法**

* 通过查表法与模2取余数法是等效的，通过异或交换律 (a XOR b) XOR c = a XOR (b XOR c)可以证明。
* 算法中crc\_table[256]数组是通过生成多项式自已和自己作移位异或运算生成的，每次移动8位，一共有2^8=256种异或组合。
* 表中0x00000000项数据,CRC32生成多项式为0x04C11DB7，直接作异或得到0x00000000.
* 通过查表叠加的方式，每次可以移动一个字节（8bit），那么crc 32只需要移位4次，crc16需要移位2次，刚好花费双倍时间。
* pppfcs()中fcs参数是寄存器，cp是首地址，fcs寄存器存放上一轮计算的结果，共计算len次，三个参数缺一不可，但是作为调用者，不需要关心寄存器这种细节，因此只需要传入首地址和长度，对使用者屏蔽实现过程，封装整个函数，降低耦合度。