实验报告

2020212183　杨再俨

2020212181　王焕捷

2020212259　何奕骁

1.实验内容和实验环境描述

1.1 实验内容

利用所学的计算机网络课程中数据链路层原理，我们组队设计了一个滑动窗口协议，在仿真环境下编程实现有噪声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，信道传播时延为**270毫秒(ms)**，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(Byte)**。

1.2 实验环境

- Microsoft Windows
- Microsoft Visual Studio Community 2022（王焕捷）
- Microsoft Visual Studio Community 2019（何奕骁）
- CodeBlocks 20.03（杨再俨）

2.软件设计

2.1 数据结构

2.1.1 数据帧

作用：定义数据链路层之间发送的帧结构体

- KIND字段为帧类别标识
- ACK字段为ACK序列号
- SEQ字段为帧序列号
- DATA字段为封装的数据包
- CRC字段为CRC32校验码

结构描述 C 复制代码

```
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) |
    +=========+========+========+
```

代码描述 C 复制代码

```c
typedef struct {
    frame_kind kind;    // 1 byte
    seq_nr ack;         // 1 byte
    seq_nr seq;         // 1 byte
    packet info;        // PKT_LEN byte -- 256
    unsigned int crc32; // 4 byte -- crc32
}frame;
```

2.1.2 全局变量

DATA_TIMER

作用：设置数据定时器的时间间隔为2000ms

```c
#define DATA_TIMER  2000
```

ACK_TIMER

作用：设置ACK计时器的时间间隔为1100ms

```c
#define ACK_TIMER   1100
```

MAX_SEQ

作用：设置序列最大编号为MAX_SEQ

```c
#define MAX_SEQ 31
```

NR_BUFS

作用：发送窗口大小=接收窗口大小=16　根据公式NR_BUFS=(MAX_SEQ+1)/2

```c
#define NR_BUFS 16
```

inc函数

作用：使得序列号在[0,MAX_SEQ)范围内循环自增。

```c
#define inc(k) if (k<MAX_SEQ)   k = k + 1; else k = 0;
```

no_nak

作用：为了避免多次请求重传同一个丢失帧，接收方用no_nak变量记录下对于某一帧是否已经发送过NAK

```c
static unsigned char no_nak = 1;
```

2.1.3 数据类型

frame_kind

作用：限定帧类型，具体有数据帧、ACK帧和NAK帧三种类型的帧

```c
typedef unsigned char frame_kind;
```

seq_nr

作用：限定帧的序列号

```c
typedef unsigned char seq_nr;
```

packet

作用：定义数据包结构体类型，规定数据包长度固定为 256 字节

```c
typedef struct {
    unsigned char data[PKT_LEN];
}packet;
```

2.2 模块结构

2.2.1 between函数

作用：判断某一帧的序号是否落在发送/接收窗口内

参数：

- `a,c` 为队列长为MAX_SEQ+1的循环队列的首尾序号
- `b` 为某一帧的序号。
- 发送窗口中，a,c分别为 `ack_expected` 和 `frame_nr` ；接收窗口中，a,c分别为 `frame_expected` 和 `too_far`

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

2.2.2 put_frame函数

作用：给帧尾部添加校验和并传送给物理层

参数：

- `frame` 为某一帧的指针
- `len` 为帧当前的长度

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

}
```

2.2.3 send_data函数

作用：构造数据帧或ACK帧或NAK帧

参数：

- `fk` 为帧类型，可取值为FRAME_DATA, FRAME_ACK, FRAME_NAK
- `frame_nr` 为帧的序列号
- `frame_expected` 为希望收到的帧的序列号

- `buffer[]` 为存放从网络层传过来的数据包buffer缓冲区

```c
static void send_data(frame_kind fk, seq_nr frame_nr, seq_nr
frame_expected, packet buffer[])
{
}
```

2.2.4 main函数

作用：main函数为主程序入口，实现了选择重传协议

参数：

- `argc` 表示命令行参数的个数
- `argv` 是一个二维字符数组，表示输入参数的内容

```c
int main(int argc, char** argv)
```

2.2.5 程序调用关系图

本实验涉及了网络层、数据链路层以及物理层.我们从每一层的作用介绍,分层次的介绍算法.

首先物理层为数据链路层提供了8000bps，270ms传播延时,以及10^(−5)的误码率的字节流传输通道.为了实现该物理层模型,我们使用了"令牌桶"算法以限制发送速率为8000bps,同时在接收端利用伪随机数随机修改收到的数据,使得接收到的比特的误码率为10^(−5),将接收到的数据进行缓冲延时270ms再提交给数据链路层来仿真信道的传播时延.

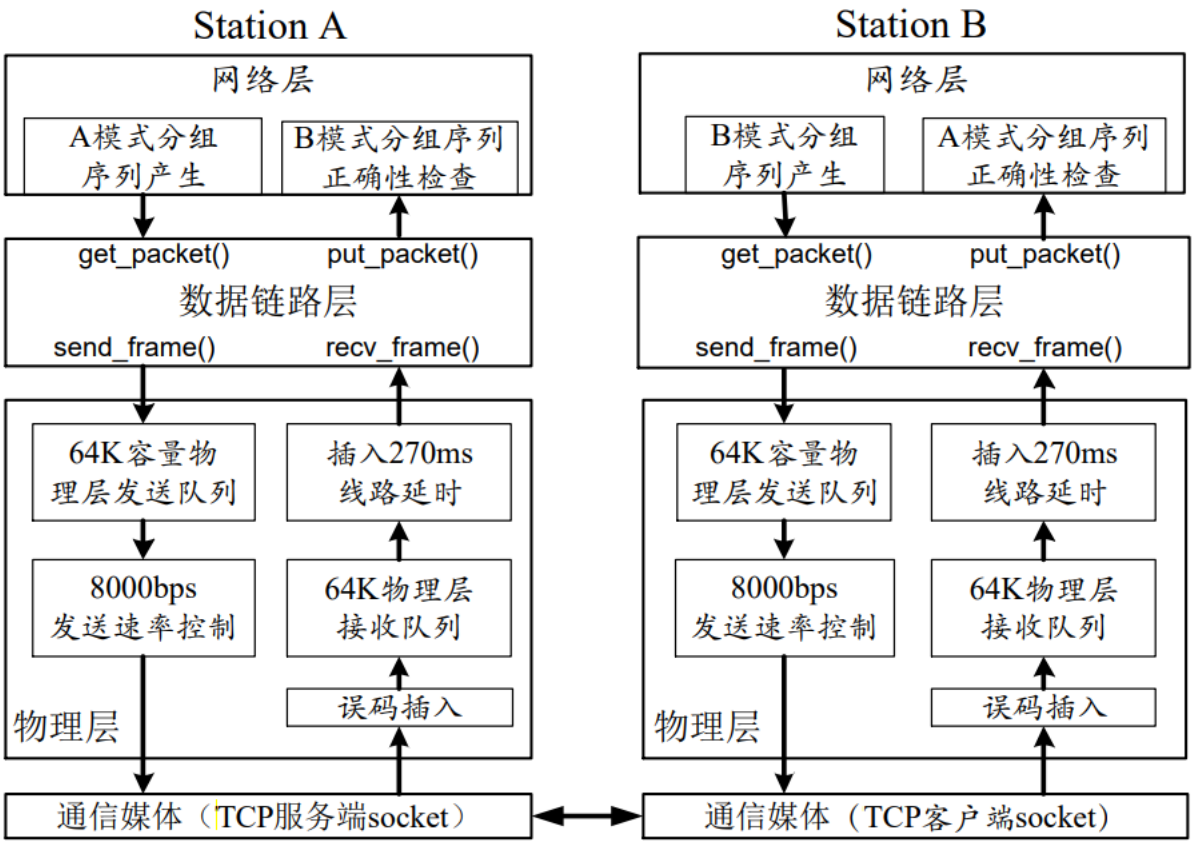
数据链路层使用接口函数 `send_frame()` 和 `recv_frame()` 来发送和接收一帧.因为是全双工通信,所以A站点和B站点都具备一个发送窗口和一个接收窗口.

对于发送窗口,当物理层READY且网络层有数据包需要发送时数据链路层进行帧的发送,收到ACK后发送窗口向右移动.若长时间未收到ACK信息则产生 `DATA_TIMEOUT` 事件,触发该事件后自

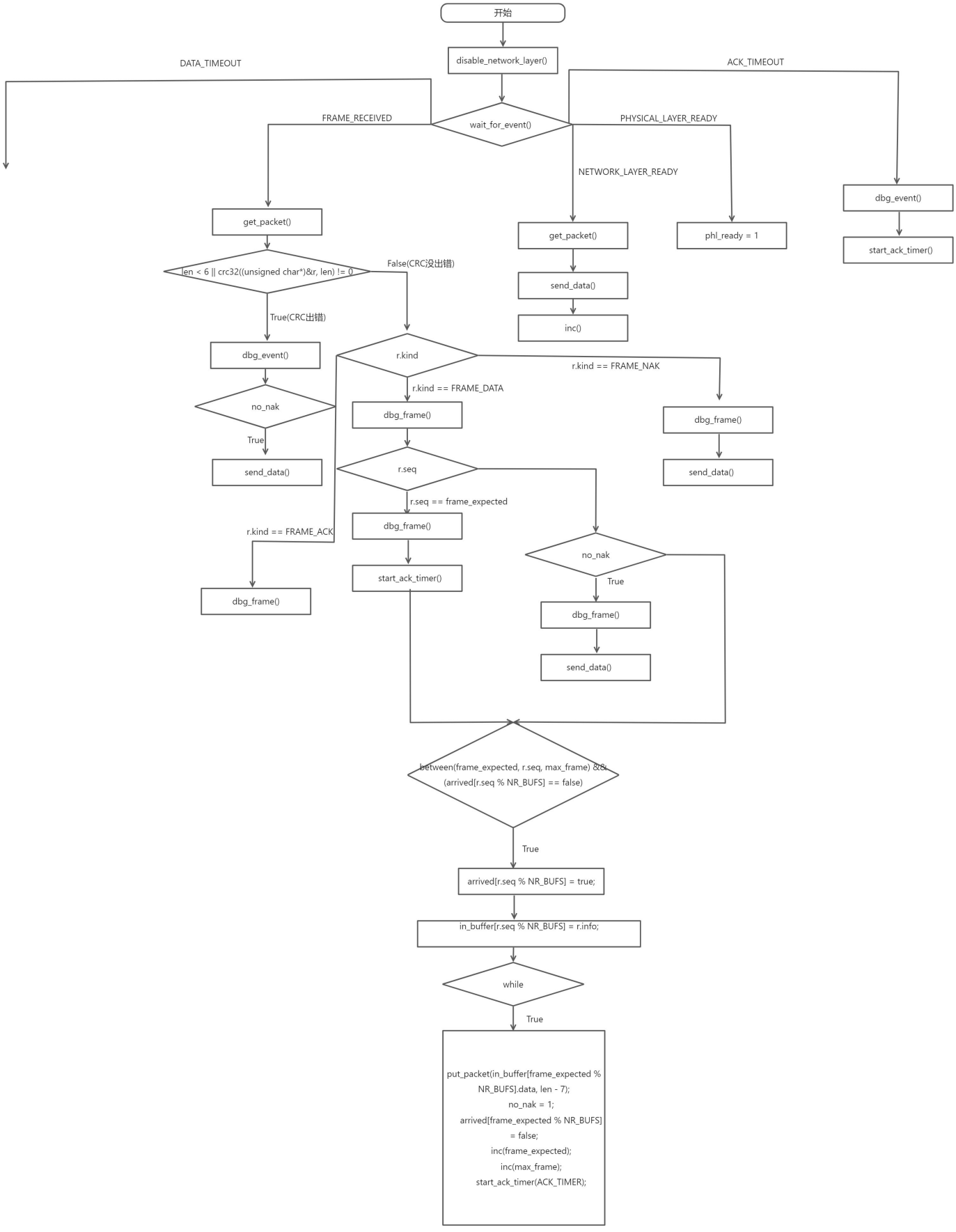
动重传未确认的帧.ACK帧使用了捎带确认的机制,同时也配备了ACK定时器,若长时间没有数据帧发送则ACK也会触发 `ACK_TIMEOUT` 进而发送ACK帧.

对于接收窗口,它会将收到的无误的数据帧暂存,直至满后将其按顺序提交给网络层,同时窗口向右移动.

网络层与数据链路层之间通过 `get_packet()` 和 `put_packet()` 接口函数进行数据包和帧之间的传输.



2.3 算法流程


开始
disable_network_layer()
wait_for_event()
DATA_TIMEOUT
ACK_TIMEOUT
FRAME_RECEIVED
PHYSICAL_LAYER_READY
NETWORK_LAYER_READY
dbg_event()
start_ack_timer()
get_packet()
get_packet()
phl_ready = 1
send_data()
inc()
len < 6 || crc32((unsigned char*)&r, len) != 0
False(CRC没出错)
True(CRC出错)
dbg_event()
r.kind
r.kind == FRAME_NAK
r.kind == FRAME_DATA
no_nak
dbg_frame()
dbg_frame()
True
send_data()
r.seq
send_data()
r.seq == frame_expected
dbg_frame()
r.kind == FRAME_ACK
no_nak
start_ack_timer()
True
dbg_frame()
dbg_frame()
send_data()
between(frame_expected, r.seq, max_frame) &&
(arrived[r.seq % NR_BUFS] == false)
True
arrived[r.seq % NR_BUFS] = true;
in_buffer[r.seq % NR_BUFS] = r.info;
while
True
put_packet(in_buffer[frame_expected %
NR_BUFS].data, len - 7);
no_nak = 1;
arrived[frame_expected % NR_BUFS]
= false;
inc(frame_expected);
inc(max_frame);
start_ack_timer(ACK_TIMER);

3.实验结果分析

3.1 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

本实验中我们实现了带NAK的**选择重传（SR）协议**，即实现了在有误码信道环境中的无差错传输功能。如果收到了损坏的帧，则该损坏的帧将被帧尾部的CRC校验码校验查出，丢弃并打印出错误信息。如果之前没有发送过NAK帧。则发送该NAK帧给发送方以加速重传；如果之前已经发送过NAK则不再发送，等待发送方定时器超时后自动重传。

3.2 程序的健壮性如何，能否可靠地长时间运行。

通过长时间的测试（至少20分钟以上），在我们三人的电脑上面，都没有出现过网络层因为收到错帧而崩溃的情况，所以因此我们认为我们的程序的健壮性比较好。

3.3 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。

3.3.1 滑动窗口的大小

根据所学的选择重传的协议的知识点并且为了充分利用序位号，我们可以定义选择重传的序号位的范围为[0,*MAX_SEQ*]，即[0,2^*n*−1]。然后我们设置发送窗口和接受窗口的大小均为2^(*n*−1)，用数据结构来表示即为(1/2)*(*MAX_SEQ+1*)，这样保证了空间的充足和避免了空间的浪费（根据所学知识这是选择重传的最大窗口）。我们设置的MAX_SEQ的大小为31，这个大小也刚刚好。

3.3.2 重传定时器的时限

重传定时器的时限的设定关乎数据帧在信道上传输的时间，重传的时间需要刚好比数据帧在信道上传输的时间长一点，设定的具体的时间需要合适，不能太长也不能太短，这样可以解决重传频繁（降低信道利用率）和降低发送速率（降低发送速率）等等问题。我们设置的为2000ms。这个时延效果较好。

3.3.3 ACK 搭载定时器的时限

ACK搭载定时器的时限的设定也是需要在一定的范围之内，如果该值选择较小，将会导致频繁重传ACK帧，这样会导致信道利用率的降低。如果我们将该值设置的过大，那么发送方将长时间接受不到ACK的确认信号，接受时间过长的话会导致重传，降低信道利用率。我们将ACK搭载定时器的设置为1000ms，效果较好。

3.4 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。

3.4.1 在无差错信道环境下分组层能获得的最大信道利用率

在无差错的信道上传输的时候，数据链路层提供的服务为8000bps，所以即为1000字节每秒，也就是说每1ms传输一个字节。所以在无差错的信道上面最大的信道利用 $\left(\frac{256}{1+1+1+256+4}\right)$ *100% = 96.97%。

3.4.2 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

现在考虑在物理层的误码率为10^(−5)的信道上面传输，在这个情况之下，我们假设有100000 bit，则有一个bit出错了，即为有一个帧出错了需要重传。所以整个信道利用率的公式就可以写为 $\left(\frac{\text{原来能够传输的帧} * 256}{(\text{原来能够传输的帧}+1+1) * 263}\right)$ *100%。而原来能够传输的帧 $\frac{100000}{263*8} \approx 47.5$ 帧。带入数据可以得到最后的信道利用率约为93.41%。

实际上在发送时，除了发送NAK导致的重传外，如果有连续的错帧，后续的错帧将不会单独发送NAK，而是等待发送方的定时器超时后重传，将会进一步降低信道利用率，但当误码率仍然较小时，对信道利用率的影响不显著

3.5 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。

可以发现，当使用无差错信道并连续发送时，信道利用率接近理论最大值96.97 %，因为误差是必然会发生的，而这些误差在可以接受的范围之内，所以我们可以认定，说明程序设计是成功的。

对窗口大小，DATE_TIMER和ACK_TIMER进行控制变量地修改然后得到新的程序，之后对这些程序进行运行可以得到不一样的信道利用率数据。我们认为，通过不断的控制变量，对三个参数进行修改再实验，不断地选择最合适的参数，可以不断提升信道利用率，最后逼近我们计算出来的理论信道利用率。

3.6 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?

我们的程序并未失败，但是其表现出来的性能距离理想的信道模型还是有一些差距，所以我们在充分检查了我们的整个程序与程序运行结果之后，发现了以下两点有关性能的问题。

① 在运行程序的时候如果电脑上有其他的软件正在运行，那么其他的软件会对程序的性能造成一定的影响，使得信道利用率发生下降。所以在运行这个程序的时候，需要尽量多去关闭我们电脑上正在运行的软件。

② 我们程序的信道不是很稳定，程序在刚开始运行的时候的信道利用率变化比较大，只有运行一段时间之后，信道的利用率才会在一个值附近波动。造成这样的原因主要是在于我们程序会反复重传未被确认的帧，这样造成了信道资源的浪费，这也可以从实验的结果里面看出。所以只要能限制其反复重传，那么我们应该就能提高信道的利用率。

4.研究和探索的问题

4.1 CRC 校验能力

CRC的检错能力很强，是(k, k + r)的编码，可以具体检测出如下的错误：

1. 全部的单个错。
2. 全部离散的二位错。
3. 全部奇数个错。
4. 长度 ⩽ k 的突发错。
5. 以 $1-(\frac{1}{2})^{k-1}$ 的概率检出长度为k + 1 k+1k+1位的突发错。

长度为k + r 的比特串总共可以编码的非零串共有 $2^{k+r}-1$ 个，其中有 2^k-1 种非零编码可通过CRC检测。

某个合法的码字发生变化，我们假设它变化成其他码字的概率都相等，则它变化后仍然能通过CRC检测的概率为：

$$\frac{2^k-1}{2^{k+r}-1}\approx 2^{-r}$$

我们假设我们这次实验的信道利用率接近100%，一直处于饱和全双工状态，且一天工作12h，数据吞吐率为8000bps，单帧约为300bytes。则：

双方信道交换帧的速率为：

$$2*\frac{8000bps}{300bytes/frame}\approx 6.6frames/sec$$

每天交换的帧数为：

$$6.6frames/sec*43200=285120frames$$

由上，发生一次误码所需要的时间为：

$$\frac{1}{285120*2^{-32}}\approx 15064\text{天}\approx 41\text{年}$$

实际上，码字变化为其他不同码字的概率肯定是不相等的，合法码字变化后仍然能通过CRC检测的概率远远小于 2^{-r} 。因此，实际上发生一次误码所需要的时间远远大于41年，几乎不可能出现误码。

4.2 程序设计方面的问题

4.2.1 get_ms()函数

程序库中获取时间坐标的函数 get_ms()如下：

get_ms()　　C　复制代码

```c
unsigned int get_ms(void) {
    struct _timeb tm;

    _ftime(&tm);

    return (unsigned int)(epoch ? (tm.time - epoch) * 1000 + tm.millitm : 0);
}
```

其调用了<sys/timeb.h>中的_ftime(struct __timeb32* _Time)函数来获取当前时间：

_ftime(struct __timeb32* _Time)函数　　C　复制代码

```c
struct   timeb {
    // 1970-01-01至今的秒数
    time_t   time;
    // 千分之一秒即毫秒
    unsigned   short   millitm;
    // 为目前时区和Greenwich相差的时间，单位为分钟
    short   timezonel;
    //  为日光节约时间的修正状态，如果为非0代表启用日光节约时间修正
    short   dstflag;
};
```

我们采用C标准库中time.h中的函数实现，如下所示：

```c
#include <time.h>


// epoch为开始通信时的起始绝对时间，单位是ms

// 使用time_t time()函数
unsigned get_ms() {
    unsigned int currentMs = clock() / (CLOCKS_PER_SRC / 1000);
    return (unsigned int)(epoch ? epoch - currentMs : 0);
}
```

4.2.2 printf风格函数的实现方式

printf风格函数最大的特点是传入参数的个数不定，其形参表里有省略号。

lprintf()

```c
int lprintf(const char *format,...)
{
    int n;
    va_list arg_ptr;

    va_start(arg_ptr, format);
    n = __v_lprintf(format, arg_ptr);
    va_end(arg_ptr);

    return n;
}

```

C语言使用宏定义实现不定参数：

```c
// 类型n的大小，这是考虑了字节对齐
#define _INTSIZEOF(n) ((sizeof(n)+sizeof(int)-1)&~(sizeof(int) - 1))
// ap指向第一个不定参数地址
#define va_start(ap,v) ( ap = (va_list)&v + _INTSIZEOF(v) )
// 下一个参数地址   返回当前ap指向的值,并且增加ap
#define va_arg(ap,t) ( *(t *)((ap += _INTSIZEOF(t)) - _INTSIZEOF(t)) )
// 将指针置为无效
#define va_end(ap) ( ap = (va_list)0 )

// 实际使用时
type va_arg(va_list argptr, type);
void va_end(va_list argptr);
void va_start(va_list argptr, last_parm);

```

4.2.3 两种定时器的设计原因

这两种定时器分别为发送的数据帧的计时器以及发送到ACK帧的计时器：

数据帧若超时，则发送方无法确定接收方是否成功收到的该数据，因此需确切的知道是哪一帧的定时器超时，即对发送窗口的每一个发送的帧都应该设置定时器；

而对于ACK帧，我们采用捎带应答的机制，并不需要知道具体是哪个ACK帧超时，故仅设置一个ACK定时器即可。

4.3 程序设计方面的问题

表3中列出了七种测试方案，设计这么多种测试方案的目的是检测此程序在不同信道条件下的传输性能。

–u是测试成帧方案的效率。无参数时模拟在实际线路中，不连续的收发时的传输性能。–f用于测试信道满负荷时的传输性能。–1,–e,–s用于测试在特殊情境下传输性能。–ber可以改变误码率，从而检验无差错传输的健壮性及性能。

5.实验总结和心得体会

Q:完成本次实验的实际上机调试时间是多少?

A:完成本次实验的实际上机调试时间约为15个小时，我们所花费的时间略微超过预估时间。主要原因在于编写代码这一过程,因为在使用C语言进行编程的过程中出现的一些语法的使用错误以及对项目整体的理解不够透彻，所以经常需要在编写的过程中翻阅数据以查看伪代码和相关知识点的复习。

Q: 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。

A:此次实验我们在Windows环境下进行,在安装IDE时并没有出现什么问题,但是在编写完代码后进行编译的过程中出现了一些错误,主要就是因为使用了网络编程相关的库,而Windows平台下的IDE可能都需要进行一些手动的链接库的操作,比如我使用的是CodeBlocks,需要手动在全局编译器设置中添加 `-lwsock32` 链接器选项。

Q: 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。

A:由于之前一直使用的是C++和Python语言,所以在要求使用C语言编写代码时会存在一些问题.尽管C++是C语言的增强版,但是在一些语法方面两者存在差异,这也是容易让人疑惑的地方.比如说C语言中常用的 `typedef` 关键字以及C语言的输入输出函数,在C++中我们只需要使用 `cout` 和 `cin` 就可以实现简单的输入输出流,但是在C中有各种各样的输入输出函数,其中最常用的是 `printf` 和 `scanf` .因为C语言是命令式编程语言,并不能像C++一样使用面向对象的思想封装对象,所以在编程过程中这种编程思维的切换也使得我们对C语言的操作不是那么的得心应手。

Q: 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。

A:本次实验使用的是基于TCP协议的全双工通信,因为我们并没有涉及传输层的编程所以在TCP协议这一块并没有什么问题.但是在数据链路层的设计过程中会因为条件的设置不妥或者出现漏掉 `break` 的地方,这将会导致严重的问题.在早期实验过程中就因为数据链路层协议的设计条件没有把握好,在程序运行了十分钟以后就出现了死锁的情况,程序会自动停下来,并不能达到要求的连续运行20分钟.关于参数的调整这一块,我们对实验参数如滑动窗口的大小，重传定时器的时限， ACK搭载定时器的时限等都进行了思考.对于这些参数的选取,可以进行定量的测试,在限定重传定时器的时限， ACK搭载定时器的时限的条件下修改MAX_SEQ,接着对程序运行结果进行分析;同理经过多次测试找到相对最优的DATA_TIMER和ACK_TIMER。

Q: 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。

A:在基于老师所给的框架下我们实现了选择重传协议来实现数据链路层之间数据帧的收发，关于库中的BUG这一点暂时还未发现.尽管库并没有问题，但是我知道一个好的C程序一定是在不断的优化过程中得

到的,这一点在库函数中也可以体现,针对某些数据结构我们可以使用更优的结构来尝试,同时像模拟物理层的"令牌桶"算法也可以使用时间复杂度和空间复杂度更优秀的算法来尝试。

Q: 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?

A:在本次的实验过程之中，我们遇到了比较多的困难，实际编程调试的时间超出了我们的预期，主要原因是在于我们对于选择重传协议的理解还是不够深刻，不够透彻。然后我们在写完我们的代码之后，对程序的代码进行调试，解决我们程序的bug这方面也花费了很长的时间。但是我们在这次的实际编程过程之中，我们采用了由简到繁，逐步推进的解决策略，从书上的协议1伪代码开始看起，慢慢地看到了协议6伪代码，我们在这个过程之中对学会了很多，对C语言的代码和编程思想有了更深一步的了解，同时也接触到了C语言里面有关网络编程的知识，对计算机网络的知识有了更深一步的巩固，在计算机网络的理论学习之中更进了一步。同时因为我们是三个人一起合作完成的这次任务，所以我们在这次实验里面了解到了更多的有关如何去合作做一个项目，如何去与他人完成对接的知识，对协议软件，对软件工程等等方面有了更多的经验和更深的理解。