计算机网络实验报告

实验名称：数据链路层通信协议的设计与实现

学　　院：计算机科学与技术

班　　级：07409

姓　　名：李劼 张志琳

学　　号：071202 071205

1、 实验内容和实验环境描述

a）实验类别

程序设计型

b）实验内容和实验目的

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，网络层分组长度在 240~256 字节范围。

总结而言，模拟信道特征如下：

1) 8000bps 全双工卫星信道
2) 单向传播时延 270 毫秒
3) 默认信道误码率为 10-5（可手动另设置）
4) 物理层接口：提供帧传输服务，帧间有 1ms 帧边界
5) 网络层属性：分组长度固定 256 字节

c）实验目的

通过该实验，进一步巩固和深刻理解数据链路层的字节填充方式的成帧技术，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。

d）实验环境

Linux Kernel 2.6.31-1

gcc 4.4.1

gdb 7.0.1

gvim 7.1

2、 协议设计

2. 协议的基本目的

本协议的目的是就数据链路层上实现仿真环境下两站点之间的无差错传输，具有简单的检错重传、流量控制等功能。

在此 GoBackN 协议中，采用了 ACK 捎带应答的方法来实现对某一帧的确认；利用搭载 ACK 计时器在一站点网络层无信息传输时，触发 ACT_TIMEOUT 使站点强发纯 ACK 帧。，在此 goback n 协议我们采用最大发送窗口数，MAX_SEQ = 7，相对单一等-停方式有效地提高了信道利用率；同时发送方在发送窗口的限制下按序发送数据，每发送一帧数据就启动一个计时器，并继续发送其他数据帧，对于接收方而言，接受窗口为 1，当判断接收到的是一个错误的帧时，丢弃后面来的帧，发送方在时间范围内收到该帧所对应的 ack 应答，则继续正常发送，如果某一帧超时或者传输错误，则从超时的那帧开始重传。

a）成帧方案

（1）数据帧

在数据帧的处理中，我们进行成帧操作的对象为以下一段字符串：

kind	seq	ack	data	checksum
1 byte	1 byte	1 byte	256 bytes	4 bytes

kind ： 指明帧的类型，1 字节

seq ： 发送窗口队列，1 字节

ack ： ACK 序号，1 字节

data ： 从网络层接收的分组内容 256 字节

checksum ： 前面所有域的 CRC 校验码，4 字节

（2）ACK 帧

帧内无数据内容，仅有 kind 和 ack 有效。分组为 5 长度的一个字符串，其中 SEQ 值不影响最终结果，在此设为 0。

最后发送的帧如下

kind	ack	checksum
1 byte	1 byte	4 bytes

b）计算 CRC 校验和的多项式定义及校验方案

（1）多项式如下：

$$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x^{1}+1$$

调用 crc32()便可返回校验码，返回值为一个 4 字节的无符号整型数据。保存在 checksum 域上。

（2）校验方案

在此协议中，在发送帧时，在帧的末尾添加了前面所有有效位的 CRC 校验码，所以在接收时，只需要计算整个帧的 CRC 校验码，如果为 0 则正确，否则错误。

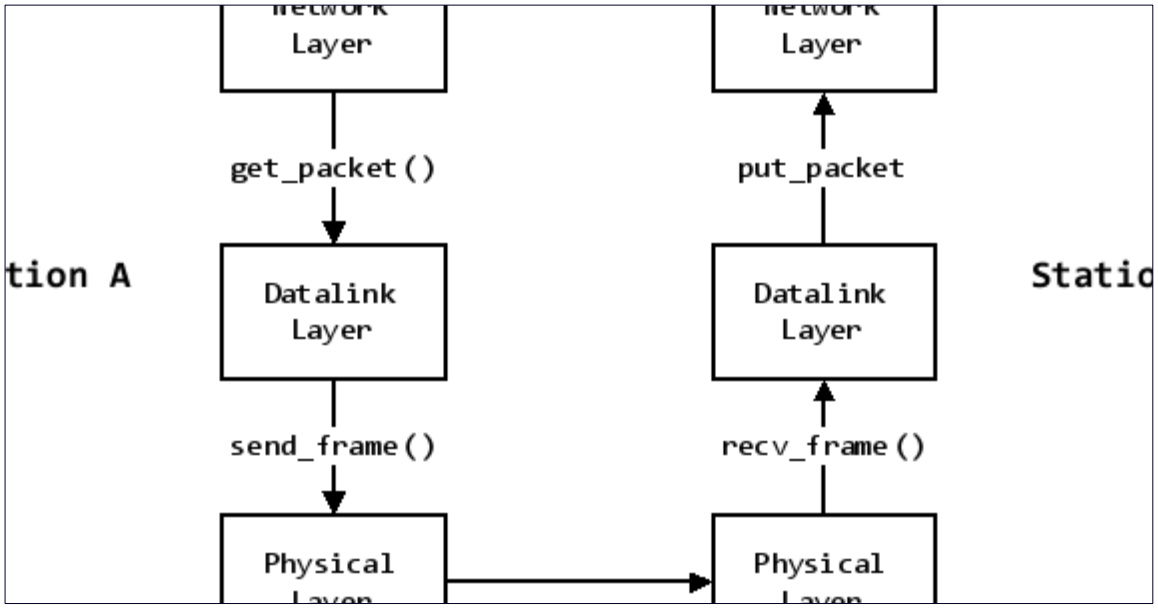
c）具体的协议传输控制：

在 phl_ready 为 1 且已用发送窗口小于最大窗口数时，执行 enable_network_layer();

否则，执行 disable_network_layer()，让网络层不接收数据。

3、 程序设计

3. 仿真环境接口



get_packet() ： 从网络层得到一个待发送的包
put_packet() ： 将一个接收到的传给网络层
send_frame() ： 将一个帧送到物理层
recv_frame() ： 从物理层接收一个帧
wait_for_evemt() ： 等待网络层、物理层和定时器事件的发生
crc32() ： 计算给定数据的 CRC 校验码

GoBackN 协议

4. 数据结构：

（1）宏定义

```
#define MAX_SEQ 15                    //最大发送窗口值为 15
#define DATA_TIMEOUT 3000                    //数据发送的超时时间
```

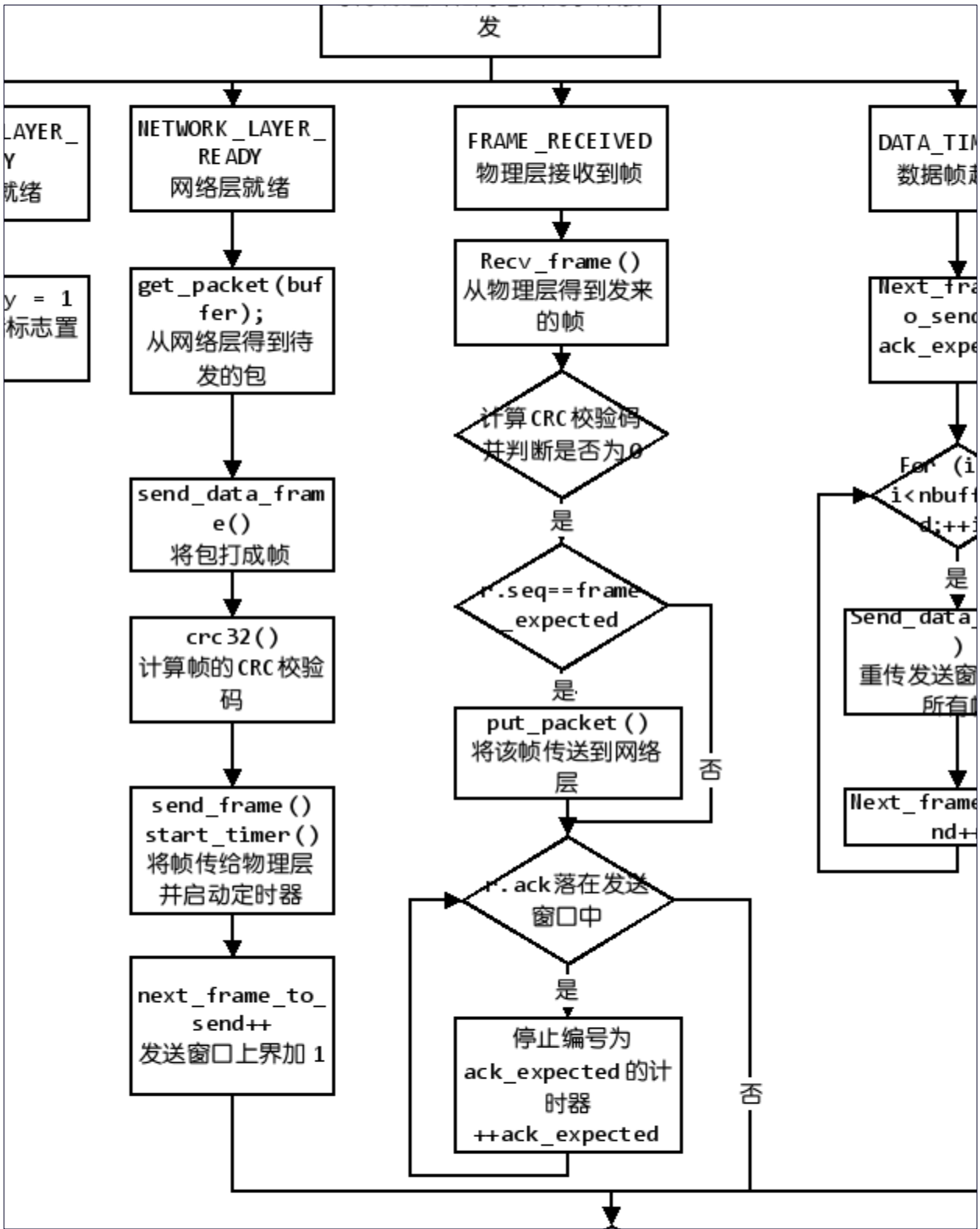
（2）frame 结构体

```
typedef unsigned char seq_nr;

    struct frame{
    unsigned char fk;
    seq_nr seq;
    seq_nr ack;
    unsigned char data[PKT_LEN]; //
    unsigned int padding;              // CRC code
};
```

（3）全局变量

static int phl_ready: 物理层就绪状态标记，初始化为 0
seq_nr next_frame_to_send: 下一次发送的 frame，初始化为 0
seq_nr ack_expected: 期望接收的 ack，初始化为 0
seq_nr frame_expected: 下一个接收的期望窗口，初始化为 0
char buffer[MAX_SEQ + 1][PKT_LEN]: 窗口缓冲区，共 8 个
int nbuffered: 已用发送窗口数，初始化为 0



5. 内部函数说明

1) send_frame(unsigned char * s, len)
计算 s 指针指向的长度为 len 的内存的 crc 码并附在 s 后，发送给物理层

2) send_data_frame(seq_nr frame_nr)
将序列号为 frame_nr 的数据封装成帧，发送给物理层

3) send_ack_frame()
发送一个 ack 为 frame_expected 的 ACK 帧

4) send_nak_frame()
发送一个 ack 为 frame_expected 的 NAK 帧

a）算法流程图：

选择性重传协议

6. 数据结构：

（1）宏定义

```
#define MAX_SEQ 15
#define NR_BUFS (MAX_SEQ + 1)/2            //窗口值为 7
#define DATA_TIMEOUT 3000                  //数据发送的超时时间
#define ACK_TIMEOUT      700          //搭载 ACK 计时器
```

（2）frame 结构体

```
typedef unsigned char seq_nr;

struct frame{
    unsigned char fk;
    seq_nr seq;
    seq_nr ack;
    unsigned char data[PKT_LEN]; //
    unsigned int padding;              // CRC code
};
```

（3）全局变量

static int phl_ready : 物理层就绪状态标记，初始化为 0
seq_nr ack_expected : 发送窗口下界，初始化为 0
seq_nr next_frame_to_send : 发送窗口上界
seq_nr too_far : 接收窗口上界，初始化为 NR_BUFS
seq_nr frame_expected : 接收窗口上界，初始化为 0
char in_buf[MAX_SEQ + 1][PKT_LEN]: 窗口缓冲区
char out_bur[MAX_SEQ + 1][PKT_LEN]: 窗口缓冲区

int nbuffered: 已用发送窗口数，初始化为 0

a）内部函数说明

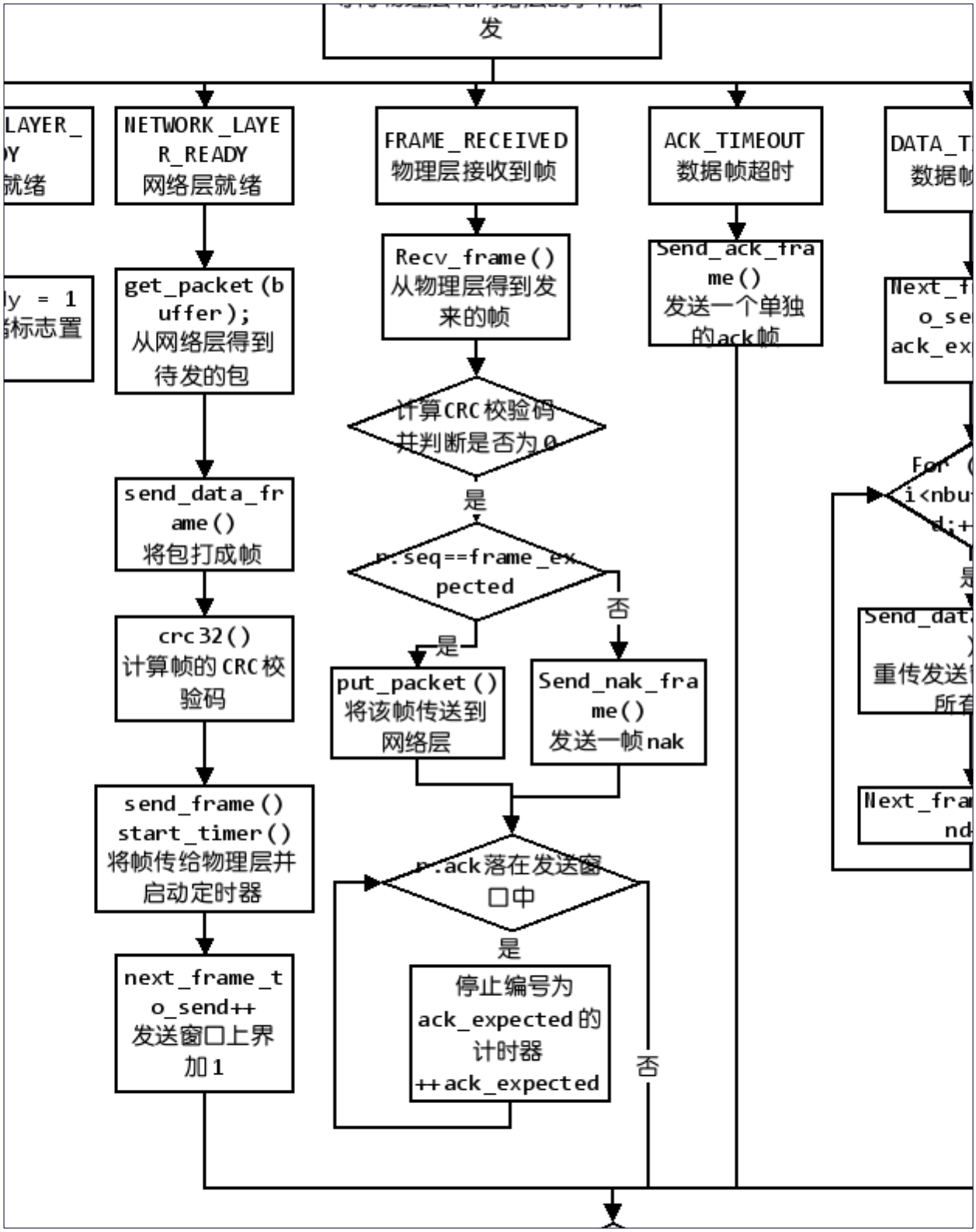
5) send_frame(unsigned char * s, len)
计算 s 指针指向的长度为 len 的内存的 crc 码并附在 s 后，发送给物理层

6) send_data_frame(seq_nr frame_nr)
将序列号为 frame_nr 的数据封装成帧，发送给物理层

7) send_ack_frame()
发送一个 ack 为 frame_expected 的 ACK 帧

8) send_nak_frame()
发送一个 ack 为 frame_expected 的 NAK 帧

b）算法流程图


发
LAYER_
Y
就绪
NETWORK_LAYER_READY
网络层就绪
FRAME_RECEIVED
物理层接收到帧
ACK_TIMEOUT
数据帧超时
DATA_T
数据帧
y = 1
标志置
get_packet(buffer);
从网络层得到待发的包
Recv_frame()
从物理层得到发来的帧
Send_ack_frame()
发送一个单独的ack帧
Next_f
o_se
ack_ex
计算CRC校验码并判断是否为0
For (
i<nbu
d;+
是
send_data_frame()
将包打成帧
P.seq==frame_expected
否
是
Send_dat
)
重传发送
所有
crc32()
计算帧的CRC校验码
put_packet()
将该帧传送到网络层
Send_nak_frame()
发送一帧nak
Next_fra
nd
send_frame()
start_timer()
将帧传给物理层并启动定时器
P.ack落在发送窗口中
是
否
next_frame_to_send++
发送窗口上界加1
停止编号为ack_expected的计时器
++ack_expected

4、

5、 实验结果分析

7. 协议的实现

在数据链路层滑动窗口协议设计实验里，我们实现的是超时重传的 GobackN 协议；相较 NAK 实现重传的 GobackN 协议，效率和实现难度小些。

本协议在无误码信道中，通信双方实现较理想的通信，信道利用率高达 96%以上。

在有误码的信道环境中，效率较参考用例低不少，并不能很好地完成无差错传输。详看性能测试记录表。

a）程序的健壮性

在无误码的信道环境下，有很好的健壮性；

然而，由于本协议中帧定位和检错精度较低，在有误码的信道中，有一定的机率会发生致命的错误；如在帧定位中，数据 FLAG 之间的 ESC 受到噪声干扰，这样帧便在此 FLAG 上定位，而下次遇到的 FLAG 也有可能是数据内容，如此帧的定位完全给打乱，容易产生长度不合规格的分组。

b）协议参数的计算和选取

1) **DATA_TIMEOUT—数据帧超时时间**

在一个数据发送帧完全发送出去时开始计时，在收到对方相应的 ACK 值时停止计时。

设通信延延迟时间为 Tp，一帧通过信道传送的时间为 Ts，那么发送方发送一帧到接收方接受到的时间间隔为 T，那么 DATA_TIMEOUT 必须大于 2T ，即

$$\text{DATA\_TIMEOUT} > 2(Tp + Ts)$$

其中 Tp 经测试大约为 300ms，而 Ts 取决于成帧后的发送帧大小，每一帧大小固定为：

$$(256 + 4 + 3) * 8 = 2104 \text{ bit}$$

信道传输速度为 8000bps，此时

$$Ts = (2104/8000\text{bps}) * 1000 = 263\text{ms};$$

故：

$$\text{DATA\_TIMEOUT} > 2(263 + 300) = 1126 \text{ ms.}$$

考虑到采用捎带确认机制，接受方不一定马上有反向数据帧，所以 DATA_TIMEOUT 的实际值应该大于计算值，大约在 2000ms 左右。

2) **ACK_TIMEOUT—搭载 ACK 记时器参数**

引用 DATA_TIMEOUT 参数估算中的数值，令此参数为 ACK_TIMEOUT，在理想情况下有

$$Tack > Tc2 + Tf$$

即从网络层等待分组并成帧时间加上发帧时间，故 Tack 在理想情况可以在（450，750）取值。

c）理论分析

选择重传协议，无差错时，由于窗口选择很大， 64×256>>(519+270)×2，故信道一直处于满的状态。影响利用率的只有转义字符，头和校验位。故极限利用率为：

256/(256+3+4)=97.338%。

在有差错情况下，则要考虑 NAK 和重传开销。现假设一次重传不会再误码，则重传开销为(256+7)×1E-5,故总极限利用率为

256/(0.00263+256+7)=97.337%

d）实验结果分析

1) **GoBackN 协议**

在无误码信道中，运行 20 分钟后的信道利用率基本与参考数据相符，其中 u 参数下，A 站效率为 49.2%，B 站为 96.67%。

在有误码信道中(误码率 10^{-5})，基本能保证运行健壮性，但效率比参考数据低了大约有 10%，偶尔会发生收到长度不合规格的规格。其中默认参数下，A 站效率为 40.7%，B 站效率为 79.5%。

而在高误码率下，由于 GoBackN 协议每次出错需要重传最多 N 个 frame，使得两个站点都在不停地重发帧，无法向前推进。

2) **选择性重传协议**

在无误码信道中，运行结果和 GoBackN 协议相同，运行 20 分钟后的信道利用率，A 站效率为 49.2%，B 站为 96.67%。

在有误码信道中(误码率 10^{-5})，A 站效率为 45.8%，B 站效率为 85.5%，比 GoBackN 略高。

高误码率下（10^{-4}），信道利用率大概维持在 30%左右。

e）存在的问题：

1) 随着误码率的提高，发生致命错误的机率会上升，但发生帧定位混乱后不一定会导致程序崩溃，所以可能把错帧上发给网络层。
2) 误码率高的情况下有可能出现不断地重传导致死锁
3) 在有误码率的信道传输中，效率较参考数据相比较低；
4) 如果要提高效率，应该从优化帧定位的算法、实现 NAK 的 gobackN、或选择重发协议上入手。

6、 研究和探索的问题

8. CRC32 的校验能力

根据分析，长度大于 r + 1 的突发差错或几个较短的突发差错发生后，坏帧被接收的概率为 0.5^r，即 0.5^{32}。假设信道传送速率为 8000bps，其平均上亿年才能出错一次，这个结果已经能够让人满意了。如果还要提高，可能对已校验结果进行二次校验，或增加校验位长度。

a）start_timer()和 start_ack_timer()的设置机制

start_timer()为每个数据帧设置单独的时钟计算该帧是否超时，参数是一个序号，当该数据帧超时时，从返回的定时器序号判断是哪一个 frame 超时并根据协议进行重发。而 ack 却不需要为每个数据帧设置单独的时钟来计算。当发生 ack 超时时，必定是接收窗口的下界所对应的数据帧的 ack 超时，因为它是所有窗口对应的数据帧中最早起动 ack 时钟的帧，因此不需要为每个 ack 时钟也携带上序列号。相应的，stop_ack_timer 停止的也是唯一的 ack 时钟。

b）协议测试问题

参数	功能
-u	在程序遇到接收错误时，可以使用无误码信道进行测试，如果在无误码信道中协议能正常工作，说明协议的接收部分没有出错，而是其他部分出错。若无误码时也出现同样的错误，则可以考虑是否发送错误。
-i	B 站网络层 100 秒发 100 秒停，进行对 ACK 应答能力的测试。
-f	产生洪水式分组用来测试协议的健壮性，当数据链路层从网络层接到一个分组，但没有正确地放入缓冲区，或放入缓冲区后相应变量设置不正确（如缓冲区满时没有执行 disable_network_layer 函数），或没有考虑到物理层的承载能力一味地向物理层发送数据，遇到洪水式的分组时就会出现如数组越界或后面的分组内容覆盖了前面的数据内容等错误。
-ber	**-1e-4** 或 **1e-5**：提高信道的误码率。测试在高误码率的情况下协议是否能正常工作

c）协议改进问题

GobackN 协议添加超时 ACK 和 NAK 机制

把算法改成选择重发协议算法

7、 实验验总结和心得体会

9. 通过动手编码，了解了协议实际工作方式

从课本上对数据链路层几个协议的描述，很难对数据链路层有一个感性的认识。而通过这次实验、以及测试数据，我们深刻理解了每个协议的工作流程，比如

1） 捎带确认是如何实现的

2） NAK、ACK 到底对信道利用率有多大影响

3） 在不同误码率下，GoBackN 和选择性重传协议的信道利用率的变化

4） 数据超时时间和 ACK 超时时间应该如何选择

5） 窗口大小对信道利用率的影响

等问题。

a）熟悉 Linux 环境下 gcc、gdb 工具链的使用

在 Linux 下一般用文本编辑器写代码，gdb 进行调试，而不是像 windows 那样所有工作都在 IDE 里完成。命令行工具链的使用，是在 Linux 完成实验必须的技能。

通过这次实验，学习了如何让 gcc 编译的时候加入.a 的动态链接库文件,以及 Makefile 的编写方式。

b）锻炼了打印调试能力

由于仿真环境有一个动态链接库提供，里面不包含调试信息，所以不能进行单步调试。但是环境里提供了很多打印调试信息的函数和参数，使得打印调试变得非常方便。

比如在实验过程中遇到了“Network layer received a bad packet。”错误。于是在每次向 p 网络层 put_packet 前，加入一条打印语句，将 packet 的内容打印出来。发现错误的产生是由于向网络层传送了一个以前已经传送过的包。找到了错误的原因，改正就比较简单了。

8、 源程序清单

源程序的清单见附件。

9、 测试结果

见附件