

[计算机网络 实验报告]

[实验一 数据链路层滑动窗口协议的设计与实现]



* 捎带传输的GoBackN协议实现
* 选择重传selective协议实现

姓名：裴子祥 胡晓妍

学号：2015211921 2015212029

班级：2015211307 2015211306

学院：计算机学院

2017-5-15

[北京邮电大学]

Ctrl+单击访问链接

**目录**

[一、 实验内容与实验环境描述 2](#_Toc483232854)

[1. 实验任务与内容 2](#_Toc483232855)

[2. 实验环境 2](#_Toc483232856)

[二、 软件设计 2](#_Toc483232857)

[1. 数据结构 2](#_Toc483232858)

[i. GoBackN协议 2](#_Toc483232859)

[ii. Selective协议 4](#_Toc483232860)

[2. 模块结构 5](#_Toc483232861)

[i. GoBackN协议 5](#_Toc483232862)

[ii. Selective协议 6](#_Toc483232863)

[3. 算法流程 8](#_Toc483232864)

[i. GoBackN协议 8](#_Toc483232865)

[ii. Selective协议 9](#_Toc483232866)

[三、 实验结果分析 10](#_Toc483232867)

[1. 功能实现 10](#_Toc483232868)

[i. GoBackN协议 10](#_Toc483232869)

[ii. Selective协议 10](#_Toc483232870)

[2. 程序健壮性 10](#_Toc483232871)

[3. 协议参数的选取 11](#_Toc483232872)

[i. GoBackN协议 11](#_Toc483232873)

[ii. Selective协议 11](#_Toc483232874)

[4. 理论分析 11](#_Toc483232875)

[i. GoBackN协议 11](#_Toc483232876)

[ii. Selective协议 12](#_Toc483232877)

[5. 实验结果分析 12](#_Toc483232878)

[i. GoBackN协议 12](#_Toc483232879)

[ii. Selective协议 13](#_Toc483232880)

[6. 存在的问题 14](#_Toc483232881)

[四、 研究和探索的问题 14](#_Toc483232882)

[1. CRC效验能力 14](#_Toc483232883)

[2. get\_ms()和 log\_printf 的实现 15](#_Toc483232884)

[3. 软件测试问题 15](#_Toc483232885)

[4. 对等协议实体之间的流量控制 15](#_Toc483232886)

[五、 实验总结与心得体会 16](#_Toc483232887)

[六、 源程序清单 17](#_Toc483232888)

[1. GoBackN协议 17](#_Toc483232889)

[2. Selective协议 20](#_Toc483232890)

# 实验内容与实验环境描述

## 实验任务与内容

利用所学数据链路层原理，设计滑动窗口协议（“回退N 步（Go-Back-N）”协议和“选择重传”（Selective）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。相关数据如下：

* 1. 信道模型为 8000bps 全双工卫星信道
  2. 信道传 播时延 270 毫秒
  3. 信道误码率为 10-5
  4. 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。

## 实验环境

Windows 10 环境下的 PC 机，使用 Microsoft Visual Studio 2015 集成化开发环境。

# 软件设计

## 数据结构

### GoBackN协议

结构定义

DATA Frame//帧格式

+=========+========+========+===============+========+

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

+=========+========+========+===============+========+

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 帧类别1位 | 帧序号1位 | 捎带确认1位 | 有效数据  256位 | CRC校检  32位 |

/\* FRAME kind \*/

#define FRAME\_DATA 1

#define FRAME\_ACK 2

#define FRAME\_NAK 3

#define DATA\_TIMER 2000 //Data帧定时

#define MAX\_SEQ 7 //帧序号空间,此时为窗口0~7

#define inc(k) if(k<MAX\_SEQ) k=k+1; else k=0//模MAX\_SEQ加1

**typedef** enum**{**FALSE**,**TRUE**}**boolean**;**

**typedef** unsigned char seq\_nr**;**//帧序号或ack序号

**typedef** struct **{**

unsigned char data**[**PKT\_LEN**];**

**}**packet**;**//数据包结构

**typedef** struct FRAME **{**

unsigned char kind**;** /\* FRAME\_DATA ,ack,nak\*/

seq\_nr ack**;** //帧种类

seq\_nr seq**;** //本帧序号

packet info**;** //数据

unsigned int padding**;** //填充字段

**}**frame**;**

**typedef** int event\_type**;//定义事件**

静态全局变量

static int phl\_ready **=** 0**;**//物理层是否准备好

主函数内变量

seq\_nr next\_frame\_to\_send **=** 0**;**//发送窗口上限,初始期望到达的帧序号

seq\_nr ack\_expected **=** 0**;**//发送窗口下限,下一次期望到达的ack

seq\_nr frame\_expected **=** 0**;**//接受窗口

frame r**;**

packet buffer**[**MAX\_SEQ **+** 1**];**//数据缓存区

seq\_nr nbuffered **=** 0**;**//控制发送的数据帧<= 发送窗口最大值

//初始缓存区中无数据包

event\_type event**,**arg**;**//事件类型

int len **=** 0**;**//记录帧长

### Selective协议

结构定义

DATA Frame//帧格式

+=========+========+========+===============+========+

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

+=========+========+========+===============+========+

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 帧类别1位 | 帧序号1位 | 捎带确认1位 | 有效数据  256位 | CRC校检  32位 |

/\* FRAME kind \*/

#define FRAME\_DATA 1

#define FRAME\_ACK 2

#define FRAME\_NAK 3

#define DATA\_TIMER 3000 //Data帧定时

#define ACK\_TIMER 1000 //ACK帧定时

#define MAX\_SEQ 31 //帧序号空间,此时为窗口0~31

#define inc(k) if(k<MAX\_SEQ) k=k+1; else k=0//模MAX\_SEQ加1

**typedef** unsigned char seq\_nr**;**//帧序号或ack序号

**typedef** struct **{**

unsigned char data**[**PKT\_LEN**];**

**}**packet**;**//数据包结构

**typedef** struct FRAME **{**

unsigned char kind**;** /\* FRAME\_DATA ,ack,nak\*/

seq\_nr ack**;** //帧种类

seq\_nr seq**;** //本帧序号

packet info**;** //数据

unsigned int padding**;** //填充字段

**}**frame**;**

**typedef** int event\_type**;//定义事件**

静态全局变量

static int phl\_ready **=** 0**;**//物理层是否准备好

主函数内变量

seq\_nr next\_frame\_to\_send **;**//发送窗口上限,初始期望到达的帧序号

seq\_nr ack\_expected**;**//发送窗口下限,下一次期望到达的ack

seq\_nr frame\_expected**;**//接受窗口下界

seq\_nr too\_far**;**//接受窗口上界

frame r**;**

packet buffer**[**MAX\_SEQ **+** 1**];**//数据缓存区

seq\_nr nbuffered **=** 0**;**//控制发送的数据帧<= 发送窗口最大值

//初始缓存区中无数据包

event\_type event**,**arg**;**//事件类型

int len **=** 0**;**//记录帧长

## 模块结构

### GoBackN协议

自定义模块

static boolean between**(**seq\_nr a**,** seq\_nr b**,** seq\_nr c**)**

发送窗口是否滑动的判别函数,若b 在 a 和 c 组成的窗口之间，则返回 true，否则返回 false,调用时between( ack.expected, r.ack, next\_frame\_to\_send );

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

将 frame 做 CRC 校验后附校验码一起发送到物理层

参数：unsigned char \* frame：要发送到物理层的帧。int len：帧的长度

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

//发送函数send\_data\_frame (next\_frame\_to\_send, frame\_expected, buffer[])

protocol.h头文件中模块

unsigned int crc32(unsigned char \*buf, int len)//crc32效验

void send\_frame(unsigned char \*frame, int len)//发送一帧给物理层

int recv\_frame(unsigned char \*buf, int size)//从物理层接受一帧

void start\_timer(unsigned int nr, unsigned int ms)//数据帧定时器

void stop\_timer(unsigned int nr)//关闭定时器

get\_packet(unsigned char \*buf)//从网络层获得一数据包

put\_packet(unsigned char \*buf,int size)//上传给网络层一包



### Selective协议

自定义模块

static boolean between**(**seq\_nr a**,** seq\_nr b**,** seq\_nr c**)**

发送窗口是否滑动的判别函数,若b 在 a 和 c 组成的窗口之间，则返回 true，否则返回 false,调用时between( ack.expected, r.ack, next\_frame\_to\_send );

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

将 frame 做 CRC 校验后附校验码一起发送到物理层

参数：unsigned char \* frame：要发送到物理层的帧。int len：帧的长度

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

//发送函数send\_data\_frame (帧的种类fk，,frame\_nr, frame\_expected, buffer[])

protocol.h头文件中模块

unsigned int crc32(unsigned char \*buf, int len)//crc32效验

void send\_frame(unsigned char \*frame, int len)//发送一帧给物理层

int recv\_frame(unsigned char \*buf, int size)//从物理层接受一帧

void start\_timer(unsigned int nr, unsigned int ms)//数据帧定时器

void stop\_timer(unsigned int nr)//关闭定时器

get\_packet(unsigned char \*buf)//从网络层获得一数据包

put\_packet(unsigned char \*buf,int size)//上传给网络层一包



## 算法流程

### GoBackN协议



### Selective协议



# 实验结果分析

## 功能实现

### GoBackN协议

采用了CRC校验和重传技术使错误得以被发现和纠正，实现了协议中帧在无误码、有误码信道环境中的无差错传输。

GoBackN程序编写的假设: 总是有反向数据帧可以捎带回去,所以并没有使用到ACK\_TIMEOUT事件，导致在站点B周期性交替“发送100秒,停发100秒”的情况下，接收方线路利用率与期望值相差较大，其余情况协议实现效率较为温和，而这点改进将在selective选择协议程序中进行改进。

### Selective协议

采用了CRC校验和重传技术使错误得以被发现和纠正，实现了协议中帧在无误码、有误码信道环境中的无差错传输。经过较长时间的运行程序,程序在无误码、有误码、高负载、低负载的信道环境中都能够实现较为理想的效率，与预期较为吻合（这是令人开心的）。

对Go Back N的算法改进：

. 去掉了Go Back N中总是有反向数据帧可以捎带回去的假设；

. 利用start\_ack\_timer启动一个辅助的定时器，如果在定时器超时之前，没有出现反向的流量，则发送一个单独的确认帧。

. 使用NAK，加快对错误帧的处理；在发生错误时，会发送nak帧，请求重传出错的帧直到对了为止

. 加大接收窗口，对于正确帧进行缓存，待收集齐整个窗口的帧后，有序上交网络层。

## 程序健壮性

GoBackN协议实现程序与selective选择重传协议程序的健壮性都较好，在高负荷和高误码率等条件下均能工作。

在较低误码率的信道环境中，两程序运行平稳，没有出现任何差错，健壮性良好，在高误码率的信道环境中，程序运行有时会出现错误太多或接受帧太多而短暂的中断15秒，但大多数时候运行时间较长，所以本程序健壮性良好。

## 协议参数的选取

### GoBackN协议

GoBackN协议程序中：活动窗口大小(帧序号范围)MAX\_SEQ 取为7，数据帧重传定时器DATA\_TIMER取为3000ms。ACK重传定时器，因为协议假设总有反向数据帧可以捎带确认回去，所以ACK\_TIMER并没有设置。

要使W/(2+2a)>=1,t1为传播时延270ms,t2为发送时延,设发送1bit需要1ms，则发送完整数据帧需要(3+256+4)\*8/1=263ms;

a=t1/t2=270/263;

所以W>=(2+2a)>4,又W取值为2n-1,所以取MAX\_SEQ 取为7,序号为0~7.

DATA\_TMIER的设置需要>=2\*(t1+t2)=1066;但取值1066在有误码的信道上结果与期望相差太大，应作出相应调整，使其增加，经过多次修改反复实验，取值3000ms时能得出较好的结果。

### Selective协议

MAX\_SEQ:31

滑动窗口的大小：16

重传定时器的时限：3000ms

ACK 搭载定时器的时限：1000ms

分组长度是256字节，由公式 可知窗口大小为2^n-1，并且双方的窗口大小均为((MAX\_SEQ+1)/2)，这样的大小足够使用又不会有过于富余的空间浪费。若窗口大小大于它，数据传输率=100%，但浪费缓冲区；若窗口大小小于它，发送端需要等待应答而使发送停顿。

选择重传的接收窗口与发送窗口一样大.选择重传协议允许与接受窗口一样多的分组失序到达,并保存这些失序到达的分组,直到连续的一组分组被交付给应用层.因为发送窗口与接收窗口是相同的,所以发送出来的所有分组都可以失序到达,而且会被保留知道交付为止.在一个可靠的协议中,接收方永远不会把分组失序地交给应用层.在他们被交付给应用层之前,先要等待那些更早发出来的分组到达。

定时器参数是根据理论分析加实验验证得到的。实验中，在这组参数的条件下信道利用率最高。

## 理论分析

### GoBackN协议

1. **无误码信道：**

在数据帧中，每一帧263字节，帧头3个字节，帧尾CRC校验位4个字节，所以信道利用率为:

256/263x100%=97.3%

1. **有误码信道：**

假设设重传操作及时，重传的数据帧的回馈正确传输。

当误码率为10-5,即发送105个比特，出现1个错误，则平均发送

100000/(263x8)=47.53帧会出现一个错误。

此时信道利用率为：

97.3%x47.53/(47.53+1)=95.5%

当误码率为10-4，即发送104个比特，出现一个错误，则平均发送

10000/(263x8)=4.75帧会出现一个错误。

此时信道利用率为：

97.3%x4.75/(4.75+1)=80.38%

### Selective协议

**a) 无误码信道：**

在数据帧中，每一帧263字节，帧头3个字节，帧尾CRC校验位4个字节，所以信道利用率为:

256/263x100%=97.3%

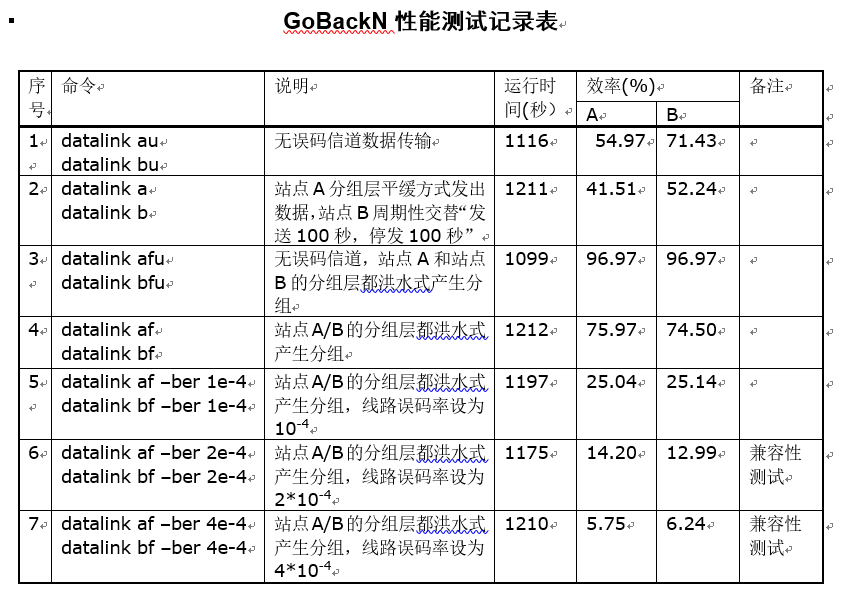
**b) 有误码信道：**

在误码率为 的信道上，100000个比特可发送 个数据包，即每传送48个数据包将有1个出错。则此时的信道利用率为97.3%x48/(48+1)=95.3%

若信道误码率为 ，则 ，即大约每5个帧就有一个出错，此时的信道利用率为97.3%x5/(5+1)=81.08%

## 实验结果分析

### GoBackN协议

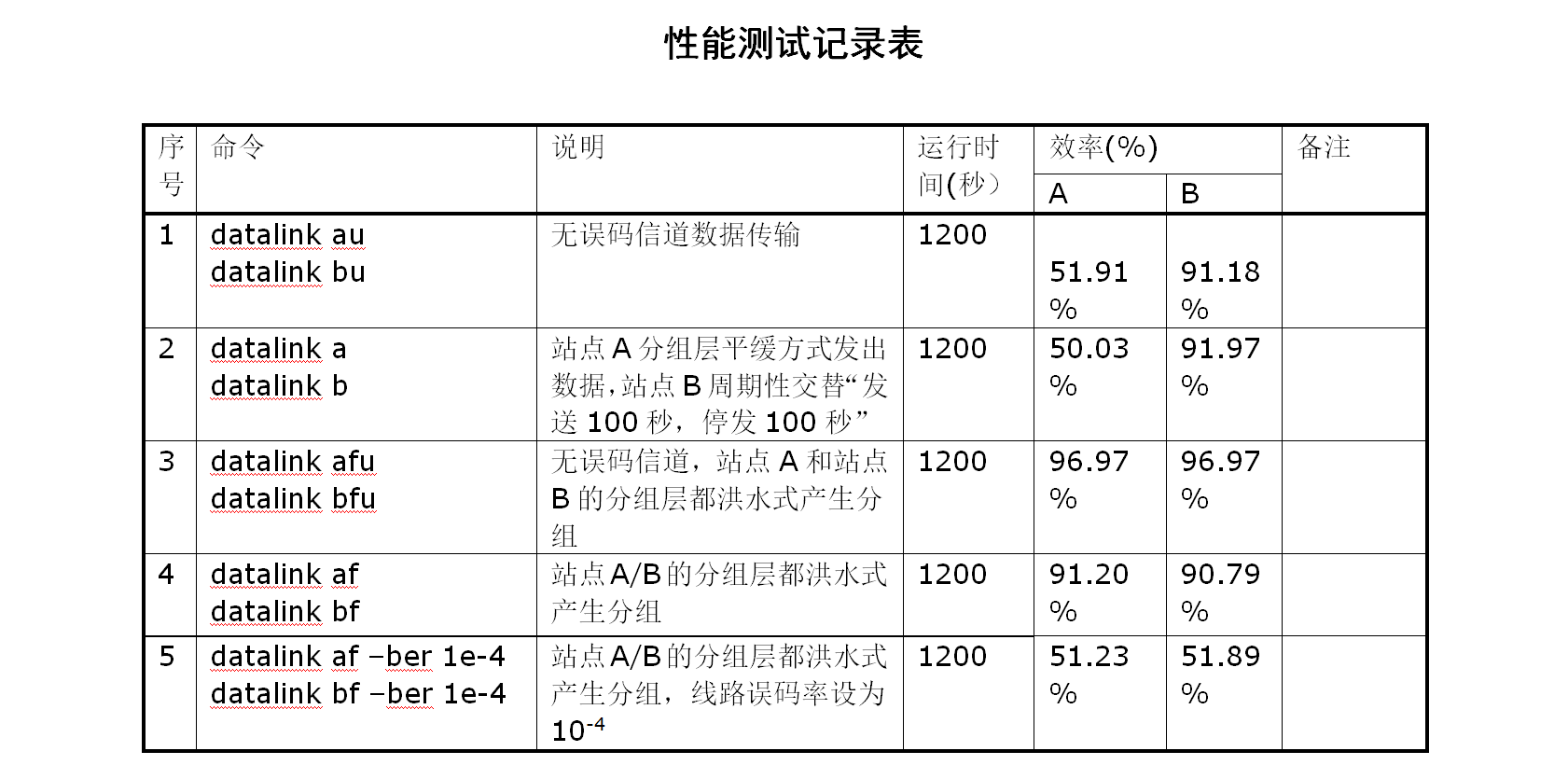


单看效率而言，无误码信道的b站，有无码信道上的效率都偏低，也进行了更多的兼容性测试，程序的稳定性与健壮性可以从多个长时间测试中看出。而这些效率也可以被接收，因为

总是有反向数据帧可以捎带回去的假设，在protocol库中并没有实现，所以导致偏低，我们可以看到后面selective协议对此的改进，也能追求所期望的效率。

### Selective协议

实验结果：



分析：

普通模式下，站点B的效率是站点A的一半，原因是B发送数据是周期性交替的，发送100s，停法100s。而洪泛模式下没有这种情况，A和B的效率大致相同。5当适当提高线路误码率时，A,B站的效率都有明显的下降。总的来说，实验结果比较接近老师给出的参考数据

## 存在的问题

GoBackN协议实验中，程序运行结果在周期交替发送数据的信道中，与所给的示例程序有所差距，但经过实验分析，是由于对GoBackN协议的前提假设导致效率的不足。但从易到难，逐步完善，在selective协议中进行相应的改进，使效率也得到显著提升。测试程序中时较为顺畅。

Selective实验中，MAX\_SEQ、DATA\_TIMER、ACK\_TIMER选取较为关键，在测试程序过程中，在之前会出现Network Layer received a bad packet from data link layer链路层工作失败的错误，但在对程序进行相应的修整，在之后的运行过程中不再出现。

# 研究和探索的问题

## CRC效验能力

CRC校验数：CRC校验数据由函数crc32()产生，函数crc32()返回一个32位整数为数据生成CRC-32校验和，并且把这 32比特校验和附在数据字节之后。

生成多项式为：x32+x26+x23+x22+x16+x12+x11+x10+x8+x7+x5+x4+x2+x1+1

校验和附加在数据帧尾部，接受方用带校验和的数据来逻辑除以生成多项式，余数为零则数据无误码，不为零则有误码等待发送方重传。

本次实验使用的 32 位的 CRC 校验码。在理论上，可以检测出：所有的奇数个数的错误，所有双比特错误，所有小于等于 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。

将错误发生事件的视为泊松分布，一帧(263\*8=2104bit)中,出现错误的位的个数的期望为：

λ=2104\*10-5=0.02104

一帧中出现33个以上错误的概率为：

P{33≤X≤2104}=

帧进一步分析，并不是长度大于32的突发错误都能够逃过检测，当突发错误后仍是G(X)的倍数时，也会被接受。此时无法检测出错误的概率为:

P{err but accept}=2-32P{33≤X≤2104}=1.20

帧概率极小，可以视为不可能事件，客户每天可以发送的帧的数目为

帧

发生一次分组误码事件发生需要1.42694×1094年。

## 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 字符串转化成最终的字符串，然后通过系统调用输出到屏幕上。

ACK计时器不能被刷新，是因为在selective选择重传协议中我们所关注的是第一个成功接收到，但没能及时发出数据帧以捎带ACK返回的帧。所以ACK计时器不能够被覆盖。

DATA计时器，关注的是每个已经发出的帧，以及ACK的帧是否及时到达，如果超时，发送方要重发计时器对应的数据帧，计时器与数据帧一一对应。而刚发出的新帧，原计时器残余时间毫无意义，需清零重新计算。

## 软件测试问题

设置多种测试方案的目的在于测试程序能否在正常环境下运行，通过多种测试方案

来模拟现实中的多种情况：

1. **无误码信道：**

测试协议最大吞吐能力。效率与帧格式设计有关，若控制字段较短，则在此信道上性能越好。

1. **发送/停发模式：**

测试协议在不同流量模式下性能。考察协议对ACK发送时机、NAK的设计。对于特殊假设条件（总是有反向数据帧可以捎带回去）的GoBackN协议，所得性能结果将是不尽如人意，但selective选择重传中对此有所改进，徐徐渐进。

1. **洪水模式：**

测试协议的抗压能力与最大输出能力。综合考察协议对ACK的发送时机,NAK的设计。

1. **高误码率信道：**

测试协议在高误码率的情况下的性能。考虑协议的重传机制的不同带来性能上的差异。

## 对等协议实体之间的流量控制

GoBackN协议中接受窗口为1，接受的数据帧必须要在接受窗口中才能成功被接受。选择重传协议有发送方窗口和接收方窗口限制，当发送方流量过大时接收方窗口将拒绝接收数据进而使发送方计时器超时而选择重传，同时发送方受发送方窗口大小限制不能突发发送大量数据。因此可以认为对等协议实体之间有流量控制。

通过enable\_network\_layer()和disable\_network\_layer()来控制网络层发来的数据包。

接收方可以把接受窗口大小设置为0以拒绝发送方发来的所有包。

# 实验总结与心得体会

如果一切100%顺利，编辑的程序一次编译就通过，运行一次就正确，那么完成本次实验的代码编写和调试工作大约需要4~6 个小时。你花的时间超过了这个预测吗？描述在调试过程中都遇到了哪些问题和解决的过程。

1. 完成本次实验的实际上机调试时间是多少？

实验用时:5天大约18个小时。时间主要花在示例代码的理解，书本上理论的分析，还有参数调整中的程序测试。

1. 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。

实验环境是在windows10下的VS2015中进行，两台PC，其中一台VS兼容性较好，可以直接允许示例程序然后在其中修改。另一台PC出现/Gy和/Zi不兼容的问题，经过网上提示教程，也不能解决问题。最后通过，新建项目，再将protocol.lib，protocol.h文件加入项目中，使得项目能够编译运行。

1. 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。

我们对C语言的掌握较为熟练，再者有示例程序和书上协议代码的参考，在编写程序过程中较为顺利，数据帧是一个结构体，采用数组形式缓存函数调用为传引用调用，传送的是数组首地址。

1. 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。

一开始，测试程序时，一直纠结于GoBackN协议程序效率相比selective协议程序效率与示例相差较大，最终发现是前提假设带来的问题。Selective程序在最开始会出现传包出错而导致程序停止。对MAX\_SEQ大小的选取，也会对效率有较大影响，selective协议中对7，15，31值进行测试，前两者效率远小于31。

1. 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。

对于库中模块，理解函数功能后之直接进行调用，暂未发现其库函数中的BUG。模块之间的功能界限较为清晰。

1. 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？

这次实验过程中遇到了较多困难，首先是对GoBackN协议于selective协议的理解不足。导致开始时都是照猫画虎地进行编程调试，主要参考书本上两个协议的代码，再进行修改使其适应protocol库，并对函数名称进行对应，对窗口数目和缓冲区数目这两个容易混淆的概念，一开始没有很好的明确算法及相应流程，后来从GoBackN到selective对协议的理解逐步深入，使实验到达最开始的两个目的：实现有噪音信道环境下的无差错传输，充分利用传输信道的带宽。代码逐步化简，并对代码进行注释处理，使得每一步都较为清晰，也便于队友间相互合作，一起讨论分析。了解到GoBackN协议的不足之处，也分析到selective协议的改进。

此次实验收获较大，我们对selective和GoBackN协议的运作有了深刻的理解。

# 源程序清单

## GoBackN协议

Datalink.h//头文件

#define FRAME\_DATA 1

#define FRAME\_ACK 2

#define FRAME\_NAK 3

#define DATA\_TIMER 2000 //Data帧定时

#define MAX\_SEQ 7 //帧序号空间,此时为窗口0~7

#define inc(k) if(k<MAX\_SEQ) k=k+1; else k=0//模MAX\_SEQ加1

**typedef** enum**{**FALSE**,**TRUE**}**boolean**;**

**typedef** unsigned char seq\_nr**;**//帧序号或ack序号

**typedef** struct **{**

unsigned char data**[**PKT\_LEN**];**

**}**packet**;**//数据包结构

**typedef** struct FRAME **{**

unsigned char kind**;** /\* FRAME\_DATA ,ack,nak\*/

seq\_nr ack**;** //帧种类

seq\_nr seq**;** //本帧序号

packet info**;** //数据

unsigned int padding**;** //填充字段

**}**frame**;**

**typedef** int event\_type**;**

GoBack.cpp//cpp文件主程序

#include <stdio.h>

#include <string.h>

#include "protocol.h"

#include "datalink.h"

static int phl\_ready **=** 0**;**//物理层是否准备好

//发送窗口是否滑动的判别函数

//between( ack.expected, r.ack, next\_frame\_to\_send );

static boolean between**(**seq\_nr a**,** seq\_nr b**,** seq\_nr c**)**

**{**

**if** **(((**a **<=** b**)** **&&** **(**b **<** c**))** **||** **((**c **<** a**)** **&&** **(**a **<=** b**))** **||** **((**b **<** c**)** **&&** **(**c **<** a**)))**

**return** TRUE**;**

**else**

**return** FALSE**;**

**}**

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

**{**

**\*(**unsigned int **\*)(**frame **+** len**)** **=** crc32**(**frame**,** len**);**

send\_frame**(**frame**,** len **+** 4**);**

phl\_ready **=** 0**;**//将数据发送到物理层后将物理层置为忙

**}**

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

**{**//发送函数Send (next\_frame\_to\_send, frame\_expected, buffer[])

frame s**;**

s**.**kind **=** FRAME\_DATA**;**

s**.**info **=** buffer**[**frame\_nr**];**//插入数据包进帧中

s**.**seq **=** frame\_nr**;**//插入序号进帧

s**.**ack **=** **(**frame\_expected **+** MAX\_SEQ**)** **%** **(**MAX\_SEQ **+** 1**);**//捎带ack

dbg\_frame**(**"Send DATA %d %d, ID %d, Windows %d, PHL Queue %d\n"**,** s**.**seq**,** s**.**ack**,** **\*(**short **\*)**s**.**info**.**data**,** MAX\_SEQ**,** phl\_sq\_len**());**

//输出帧信息

put\_frame**((**unsigned char **\*)&**s**,** 3 **+** PKT\_LEN**);**

start\_timer**(**frame\_nr**,** DATA\_TIMER**);**

**}**

int main**(**int argc**,** char **\*\***argv**)**

**{**

seq\_nr next\_frame\_to\_send **=** 0**;**//发送窗口上限,初始期望到达的帧序号

seq\_nr ack\_expected **=** 0**;**//发送窗口下限,下一次期望到达的ack

seq\_nr frame\_expected **=** 0**;**//接受窗口

frame r**;**

packet buffer**[**MAX\_SEQ **+** 1**];**//数据缓存区

seq\_nr nbuffered **=** 0**;**//控制发送的数据帧<= 发送窗口最大值

//初始缓存区中无数据包

event\_type event**,**arg**;**//事件类型

int len **=** 0**;**//记录帧长

protocol\_init**(**argc**,** argv**);**//协议初始化

lprintf**(**"Designed by Pei Zixiang, build: " \_\_TIME\_\_ " " \_\_DATE\_\_" \n"**);**

disable\_network\_layer**();**

**while** **(**1**)**

**{**

event **=** wait\_for\_event**(&**arg**);**

**switch** **(**event**)**

**{**

**case** NETWORK\_LAYER\_READY**:**//网络层准备就绪,有待发送的分组

get\_packet**(**buffer**[**next\_frame\_to\_send**].**data**);**//从网络层获得新包

nbuffered**++;**

send\_data\_frame**(**next\_frame\_to\_send**,** frame\_expected**,** buffer**);**

inc**(**next\_frame\_to\_send**);**//发送窗口上限前移

**break;**

**case** PHYSICAL\_LAYER\_READY**:**//物理层准备就绪

phl\_ready **=** 1**;**//物理层状态置1

**break;**

**case** FRAME\_RECEIVED**:**//物理层接受到一帧

len **=** recv\_frame**((**unsigned char **\*)&**r**,** **sizeof(**r**));**

//计算接受到的帧长,从物理层获得一帧

**if** **(**len **<** 5 **||** crc32**((**unsigned char **\*)&**r**,** len**)** **!=** 0**)**

**{**//帧被破坏,或CRC32校检失败,丢弃坏帧

dbg\_event**(**"\*\*\*\* Receiver Error, Bad CRC Checksum\n"**);**

**break;**

**}**

dbg\_frame**(**"Recv DATA %d %d, ID %d\n"**,** r**.**seq**,** r**.**ack**,** **\*(**short **\*)**r**.**info**.**data**);**

**if** **(**r**.**seq **==** frame\_expected**)**

**{**//帧按序到达

put\_packet**(**r**.**info**.**data**,** len**-**7**);**

inc**(**frame\_expected**);**

**}**

**while** **(**between**(**ack\_expected**,** r**.**ack**,** next\_frame\_to\_send**))**

**{**

nbuffered**--;**

stop\_timer**(**ack\_expected**);**

inc**(**ack\_expected**);**

**}**

**break;**

**case** DATA\_TIMEOUT**:**

dbg\_event**(**"---- DATA %d timeout\n"**,** arg**);**

next\_frame\_to\_send **=** ack\_expected**;**

**for** **(**int i **=** 1**;** i **<=** nbuffered**;** i**++)**

**{**//重发错帧之后所有帧

send\_data\_frame**(**next\_frame\_to\_send**,** frame\_expected**,** buffer**);**

inc**(**next\_frame\_to\_send**);**

**}**

**break;**

**}**

**if** **(**nbuffered **<** MAX\_SEQ **&&** phl\_ready**)**//物理层ready且缓存未满

enable\_network\_layer**();**

**else**

disable\_network\_layer**();**

**}**

**}**

## Selective协议

Datalink.h//头文件

/\* FRAME kind \*/

#define FRAME\_DATA 1

#define FRAME\_ACK 2

#define FRAME\_NAK 3

Selective.cpp//cpp文件主程序

#include <stdio.h>

#include <string.h>

#include<stdbool.h>

//改了c++常规调试信息格式最后一个

#include "protocol.h"

#include "datalink.h"

#define MAX\_SEQ 31

#define NR\_BUFS ((MAX\_SEQ+1)/2)

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

**typedef** unsigned char seq\_nr**;**

**typedef** struct **{** unsigned char data**[**PKT\_LEN**];** **}**packet**;**

bool no\_nak **=** true**;**

seq\_nr oldest\_frame **=** MAX\_SEQ **+** 1**;**

seq\_nr frame\_expected**;**

**typedef** struct **{**

unsigned char kind**;** // FRAME\_DATA

seq\_nr seq**;**

seq\_nr ack**;**

packet info**;**

unsigned int padding**;**//Stack of s was currupted调试的时候出错，加上这个就没错了

**}**frame**;**

#define DATA\_TIMER 3000

#define ACK\_TIMER 1000

static int phl\_ready **=** 0**;**

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

**{**

**\*(**unsigned int **\*)(**frame **+** len**)** **=** crc32**(**frame**,** len**);**

send\_frame**(**frame**,** len **+** 4**);**

phl\_ready **=** 0**;**

**}**

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

**{**

**return((**a **<=** b**)** **&&** **(**b **<** c**))** **||** **((**c **<** a**)** **&&** **(**a **<=** b**))** **||** **((**b **<** c**)** **&&** **(**c **<** a**));**

**}**

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

**{**

frame s**;**

s**.**kind **=** fk**;**

s**.**seq **=** frame\_nr**;**

s**.**ack **=** **(**frame\_expected **+** MAX\_SEQ**)** **%** **(**MAX\_SEQ **+** 1**);**

**if** **(**fk **==** FRAME\_DATA**)** **{**

//s.info = buffer[frame\_nr % NR\_BUFS];//??

memcpy**(**s**.**info**.**data**,** **&**buffer**[**frame\_nr **%** NR\_BUFS**].**data**,** PKT\_LEN**);**

dbg\_frame**(**"Send DATA %d %d, ID %d\n"**,** s**.**seq**,** s**.**ack**,** **\*(**short **\*)**s**.**info**.**data**);**

put\_frame**((**unsigned char **\*)&**s**,** 3 **+** PKT\_LEN**);**

start\_timer**(**frame\_nr**%**NR\_BUFS**,** DATA\_TIMER**);**

**}**

**if** **(**fk **==** FRAME\_NAK**)** **{**

no\_nak **=** false**;**

dbg\_frame**(**"Send NAK %d\n"**,** s**.**ack**);**

put\_frame**((**unsigned char **\*)&**s**,** 3 **+** PKT\_LEN**);**

**}**

**if** **(**fk **==** FRAME\_ACK**)**

**{**

dbg\_frame**(**"Send ACK %d\n"**,** s**.**ack**);**

put\_frame**((**unsigned char **\*)&**s**,** 3 **+** PKT\_LEN**);**

**}**

phl\_ready **=** 0**;**

stop\_ack\_timer**();**

**}**

int main**(**int argc**,** char **\*\***argv**)**

**{**

int arg**,** len **=** 0**;**

protocol\_init**(**argc**,** argv**);**

lprintf**(**"Designed by Hu Xiaoyan, build: " \_\_DATE\_\_" "\_\_TIME\_\_"\n"**);**

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**;**//接受窗口下界

seq\_nr too\_far**;**//接受窗口上界

int i**;**

frame r**;**

packet out\_buf**[**NR\_BUFS**];**

packet in\_buf**[**NR\_BUFS**];**

bool arrived**[**NR\_BUFS**];**

seq\_nr nbuffered**;**

int event**;**

ack\_expected **=** 0**;**

next\_frame\_to\_send **=** 0**;**

frame\_expected **=** 0**;**

too\_far **=** NR\_BUFS**;**

nbuffered **=** 0**;**

**for** **(**i **=** 0**;** i **<** NR\_BUFS**;** i**++)** arrived**[**i**]** **=** false**;**

enable\_network\_layer**();**

**while** **(**true**)**

**{**

event **=** wait\_for\_event**(&**arg**);**

**switch** **(**event**)**

**{**

**case** NETWORK\_LAYER\_READY**:**//允许网络层发送数据帧

nbuffered**++;**//缓冲区数据帧数加一

get\_packet**(**out\_buf**[**next\_frame\_to\_send**%**NR\_BUFS**].**data**);**//从网络层接收数据

send\_data**(**FRAME\_DATA**,** next\_frame\_to\_send**,** frame\_expected**,** out\_buf**);**//发送数据帧

inc**(**next\_frame\_to\_send**);**//发送窗口上界下移

dbg\_frame**(**"Send DATA %d %d, ID %d\n"**,** r**.**seq**,** r**.**ack**,** r**.**info**);**

**break;**

**case** PHYSICAL\_LAYER\_READY**:**//物理层空闲

phl\_ready **=** 1**;**

**break;**

**case** FRAME\_RECEIVED**:**//接受方收到一个帧

len **=** recv\_frame**((**unsigned char **\*)&**r**,** **sizeof** r**);**//帧长

**if** **(**len **<** 5 **||** crc32**((**unsigned char **\*)&**r**,** len**)** **!=** 0**)** **{**

dbg\_event**(**"\*\*\*\* Receiver Error, Bad CRC Checksum\n"**);**

**if** **(**no\_nak**)** send\_data**(**FRAME\_NAK**,** 0**,** frame\_expected**,** out\_buf**);**

**break;**

**}**

**if** **(**r**.**kind **==** FRAME\_DATA**)**

**{**

dbg\_frame**(**"Recv DATA %d %d, ID %d\n"**,** r**.**seq**,** r**.**ack**,** r**.**info**);**

**if** **((**r**.**seq **!=** frame\_expected**)** **&&** no\_nak**)**//未按顺序到达

send\_data**(**FRAME\_NAK**,** 0**,** frame\_expected**,** out\_buf**);**//返回nak

**else** start\_ack\_timer**(**ACK\_TIMER**);**//开启ack定时器

**if** **(**between**(**frame\_expected**,** r**.**seq**,** too\_far**)** **&&** **(**arrived**[**r**.**seq**%**NR\_BUFS**]** **==** false**))** **{**

arrived**[**r**.**seq**%** NR\_BUFS**]** **=** true**;**

in\_buf**[**r**.**seq**%** NR\_BUFS**]** **=** r**.**info**;**

**while** **(**arrived**[**frame\_expected**%**NR\_BUFS**])** **{**//如果到达的帧落在接收窗上

put\_packet**(**in\_buf**[**frame\_expected**%**NR\_BUFS**].**data**,** len **-** 7**);**//将到达的数据帧发送到网络层

no\_nak **=** true**;**//表示对于这一帧没有发送过否定确认

arrived**[**frame\_expected**%**NR\_BUFS**]** **=** false**;**//该处缓冲空间重新记为空

inc**(**frame\_expected**);**//接收窗口后移一位

inc**(**too\_far**);**

start\_ack\_timer**(**ACK\_TIMER**);**

**}**

**}**

**}**

**if** **((**r**.**kind **==** FRAME\_NAK**)** **&&** between**(**ack\_expected**,** **(**r**.**ack **+** 1**)** **%** **(**MAX\_SEQ **+** 1**),** next\_frame\_to\_send**))** **{**

send\_data**(**FRAME\_DATA**,** **(**r**.**ack **+** 1**)** **%** **(**MAX\_SEQ **+** 1**),** frame\_expected**,** out\_buf**);**

//dbg\_frame("Recv NAK %d\n", r.ack);

**}**

**while** **(**between**(**ack\_expected**,** r**.**ack**,** next\_frame\_to\_send**))** **{**

nbuffered**--;**//缓冲区数据帧数量减一

stop\_timer**(**ack\_expected**%**NR\_BUFS**);**//停止ack计时器

inc**(**ack\_expected**);**

**}**

**break;**

**case** DATA\_TIMEOUT**:**//数据帧超时。重发

dbg\_event**(**"---- DATA %d timeout\n"**,** arg**);**

send\_data**(**FRAME\_DATA**,** ack\_expected**,** frame\_expected**,** out\_buf**);**

**break;**

**case** ACK\_TIMEOUT**:**//确认帧超时，发送确认帧

dbg\_event**(**"---- ACK %d timeout\n"**,** arg**);**

send\_data**(**FRAME\_ACK**,** 0**,** frame\_expected**,** out\_buf**);**

**break;**

**}**

**if** **(**nbuffered **<** NR\_BUFS **&&** phl\_ready**)** enable\_network\_layer**();**//接收方缓冲区未满且物理层空闲，则开启网络层

**else** disable\_network\_layer**();**//否则关闭网络层

**}**

**}**