**北京邮电大学《计算机网络》课程实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **实验**  **名称** | 数据链路层滑动窗口协议的设计与实现 | | **学 院** | 计算机 | **指导教师** | 王晓茹 |
| **班 级** | **班内序号** | **学 号** | | **学生姓名** | **成绩** | |
| **2018211318** | **02** | **2018210074** | | **熊宇** |  | |
|  |  |  | |  |  | |
|  |  |  | |  |  | |
| **实**  **验**  **内**  **容** | 本次实验选用的滑动窗口协议为选择重传协议，利用所学数据链路层原理，自行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，信道提供帧传输服务，网络层分组长度固定为256字节。  本次实验选用的滑动窗口协议为选择重传协议，并且使用了NAK通知机制。 | | | | | |
| **学生实验报告** | （详见“实验报告和源程序”册） | | | | | |
| **课**  **程**  **设**  **计**  **成**  **绩**  **评**  **定** | **评语**:  **成绩**:  指导教师签名：  年 月 日 | | | | | |

注：评语要体现每个学生的工作情况，可以加页。

**北京邮电大学《计算机网络》课程实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **实验**  **名称** | 数据链路层滑动窗口协议的设计与实现 | | **学 院** | 计算机 | **指导教师** | 王晓茹 |
| **班 级** | **班内序号** | **学 号** | | **学生姓名** | **成绩** | |
| **2018211318** | **02** | **2018210074** | | **熊宇** |  | |
|  |  |  | |  |  | |
|  |  |  | |  |  | |
| **实**  **验**  **内**  **容** | 本次实验选用的滑动窗口协议为：   1. 选择重传   是否选用了NAK机制：   1. 使用了NAK通知机制。 | | | | | |
| **学生实验报告** | 实验报告是否上交？  已上交。 | | | | | |
| **课**  **程**  **设**  **计**  **成**  **绩**  **评**  **定** | **评语**:   1. 程序是否存在BUG？   编写过程中出现BUG，修正后目前无BUG。   1. 20分钟后，发送效率性能情况？峰值？   详见性能测试表。   1. 性能表与实际测试性能是否相符？   相符。   1. 程序中是否使用了合理的数据结构？如数据帧的设计？   使用了。详见实验报告【实验设计】数据结构部分。   1. 在数据帧头部中，是否使用了字节计数域？   并未使用。   1. 如何设定最大窗口？（写出公式和具体值）   NR\_BUFS = ((MAX\_SEQ + 1) / 2)，其中SEQ用 5 bit表示，因此MAX\_SEQ = (2^5) – 1 = 31，因此选取NR\_BUFS=16。   1. 如何设定超时重传时间？（给出计算公式）是否合理？   ack\_timer和data\_timer的选取要综合考虑，ack\_timer要远小于data\_timer，NR\_BUFS保持不变。  最终选取ack\_timer=280,data\_timer=5000   1. 如何设定ACK超时时间？（给出公式）   ack\_timer和data\_timer的选取要综合考虑，ack\_timer要远小于data\_timer，NR\_BUFS保持不变。  最终选取ack\_timer=280,data\_timer=5000   1. 程序的书写风格是否符合实验报告的要求？   符合。   1. 实验分工是否合理？   单人完成。 | | | | | |

目录

**实验报告封面1**

**实验报告评分表2**

**实验报告正文5**



**计算机网络实验报告**



**题目：选择重传协议**

**班 级： 2018211318班\_\_\_**

**学 号： 2018210074\_\_\_\_\_**

**姓 名： 熊宇\_\_\_\_\_\_\_\_\_\_\_**

**学 院： 计算机学院\_\_\_\_\_**

**2020年 5 月 4 日**

**实验报告**

**一 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。

**二 实验设备环境**

Windows 环境 PC 机，Microsoft Visual Studio 2013+集成化开发环境。

**三 实验设计**

1. 数据结构：
2. 帧

/\* 定义帧类型 \*/

typedef struct FRAME

{

unsigned char kind; /\* 帧的种类 \*/

unsigned char ack; /\* ACK号 \*/

unsigned char seq; /\* 序列号 \*/

packet data; /\* 需要被转发的数据 \*/

unsigned int padding;

}FRAME;

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) |

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

KIND：表示帧的类别 ACK：ACK序列号 SEQ：帧序列号 CRC：校验和

② 全局变量解释

/\* 发送窗口上限 \*/

static unsigned char next\_frame\_to\_send;

/\* 发送窗口下限 \*/

static unsigned char ack\_expected;

/\* 接收窗口上限 \*/

static unsigned char too\_far;

/\* 接收窗口下限 \*/

static unsigned char frame\_expected;

/\* 计数，发了多少个 \*/

static unsigned char nbuffered;

/\* 数据帧超时机制 \*/

#define DATA\_TIMER 5000

/\* ACK超时机制 \*/

#define ACK\_TIMER 280

/\* 帧的最大长度 \*/

#define MAX\_SEQ 31

/\* 窗口大小 \*/

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

/\* 实现k的按模 +1 \*/

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

/\* phl\_ready = 0 意味着物理层还没有准备好 \*/

static int phl\_ready = 0;

/\* no\_nak = true 意味着还没有发送过一个NAK \*/

bool no\_nak = true;

1. 主程序中变量解释

/\* 事件，在protocol.h中已完成定义 \*/

#define NETWORK\_LAYER\_READY 0

#define PHYSICAL\_LAYER\_READY 1

#define FRAME\_RECEIVED 2

#define DATA\_TIMEOUT 3

#define ACK\_TIMEOUT 4

int event, arg;

FRAME f;

int len = 0;

int i;

/\* 发送窗口的“椅子” \*/

packet out\_buf[NR\_BUFS];

/\* 接受窗口的“椅子” \*/

packet in\_buf[NR\_BUFS];

/\* 收到的数据帧该坐的“椅子”当前是否空闲 \*/

bool arrived[NR\_BUFS];

/\* 初始化 \*/

/\* 接收端初始化时，enable\_network\_layer() 被抑制 \*/

enable\_network\_layer();

ack\_expected = 0;

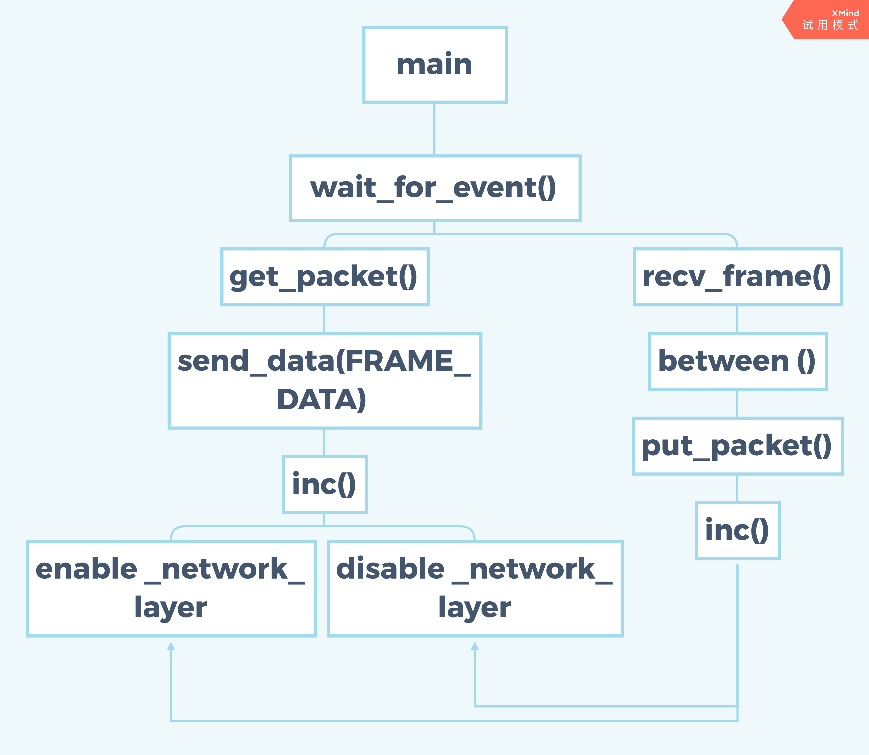
next\_frame\_to\_send = 0;

frame\_expected = 0;

too\_far = NR\_BUFS;

nbuffered = 0;

1. 模块结构



static int between(unsigned char a, unsigned char b, unsigned char c)

**作用**：判断数据帧是否位于接受窗口内，来判断是否暂存

**参数**：a,b,c，均为字节类型，其中两个分别为窗口的上、下界，一个为帧的编号。其中，发送窗口的上界和下界分别为next\_frame\_to\_send和ack\_expected，接收窗口的上界和下界分别为too\_far和frame\_expected，均定义在main函数中。

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

**作用**：向网络层上交数据帧

**参数**：frame，字节数组，由除padding域之外的帧内容转换而来；len，整型，为帧的当前长度。

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

**作用**：发出帧（此处按帧的类型用if语句进行不同的操作，所以我将三个函数合在一起）

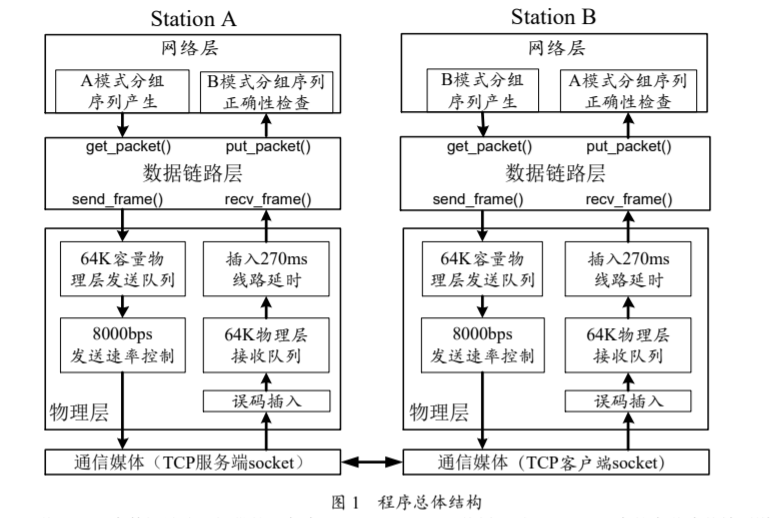
**参数**：fk，字节类型，为帧的内容；frame\_nr，字节类型，为帧的编号；frame\_expected，字节类型，为希望收到的帧的编号；buffer[]，二维字节数组，为缓冲区。

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

**作用**：主程式，包含选择重传协议的算法流程。

**参数**：argc，整型，表示命令行参数的个数；argv，二维字符数组，表示参数内容。

1. 算法流程



**物理层**：为数据链路层提供的服务为8000bps，270ms传播延时10-5误码率的字节流传输通道。为了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完成两个站点之间的通信。由于同一台计算机上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”算法限制发送速率以仿真8000bps线路；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，以仿真信道的传播时延特性。为了简化程序，省略了“成帧”功能，数据链路层利用接口函数send\_data()和recv\_frame()发送和接收一帧。

**数据链路层**：程序由同学自己设计完成，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()函数从网络层得到一个分组；当数据链路层成功接收到一个分组后，通过put\_packet()函数提交给网络层。

**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之间交换长度固定为256字节的数据分组。网络层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。

**四 实验分析**

1. 实验结果简介：

此次实验，本小组采用的选择重传协议，能够较长时间地在有误码信道环境中无差错传输，同时，在传输过程中，如果出现错误，丢失的帧，则会发出警报，在接受方会得到提示，因此能够可靠地实现通信。

1. 协议参数的选取：

NR\_BUFS的选取

NR\_BUFS = ((MAX\_SEQ + 1) / 2)

因此选取NR\_BUFS=16

ack\_timer和data\_timer的选取

要综合考虑，ack\_timer要远小于data\_timer，NR\_BUFS保持不变

最终选取ack\_timer=280,data\_timer=5000

1. 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；推导出在有差错条件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。为了简化有差错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：假定出现超时后一次重传可以 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。

**分析**：由于需要携带帧讯息，因此最大的信息利用率为约为（256-3）/256≈98.8%。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；每帧的附加信息固定为10，耗时10ms；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。在有10-5的错误率的信道上，算下来在这100000个比特中可以传送100000/（256\*8）≈48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使其成为正确帧。则每传送48个数据包需要传送48+1+1=50次，此时信道利用率≈94.9%，而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（（256-3）/256）\*（48/（48+1+k）），在此程式中，平均要重传10次，因此信道利用率降为80.4%。在ESC/FLAG模式中传输的平均250个字符需要两倍的传输空间即极限值500，此时的信道利用率的极限值是，而在10-5的错误率的信道上则利用率为47.5%，若平均每个错帧重传10次，利用率将下降为40.2%。

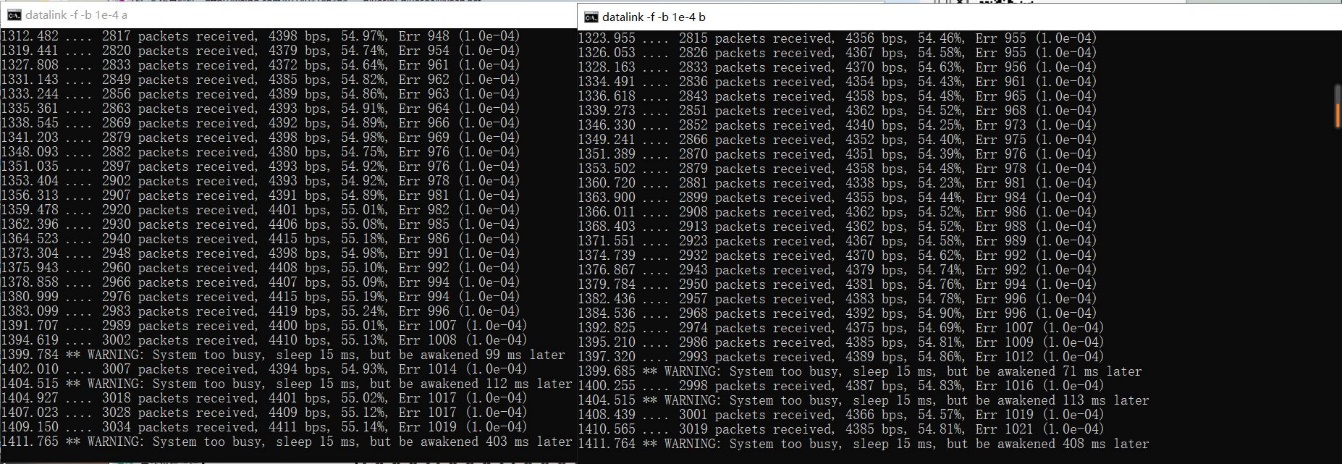
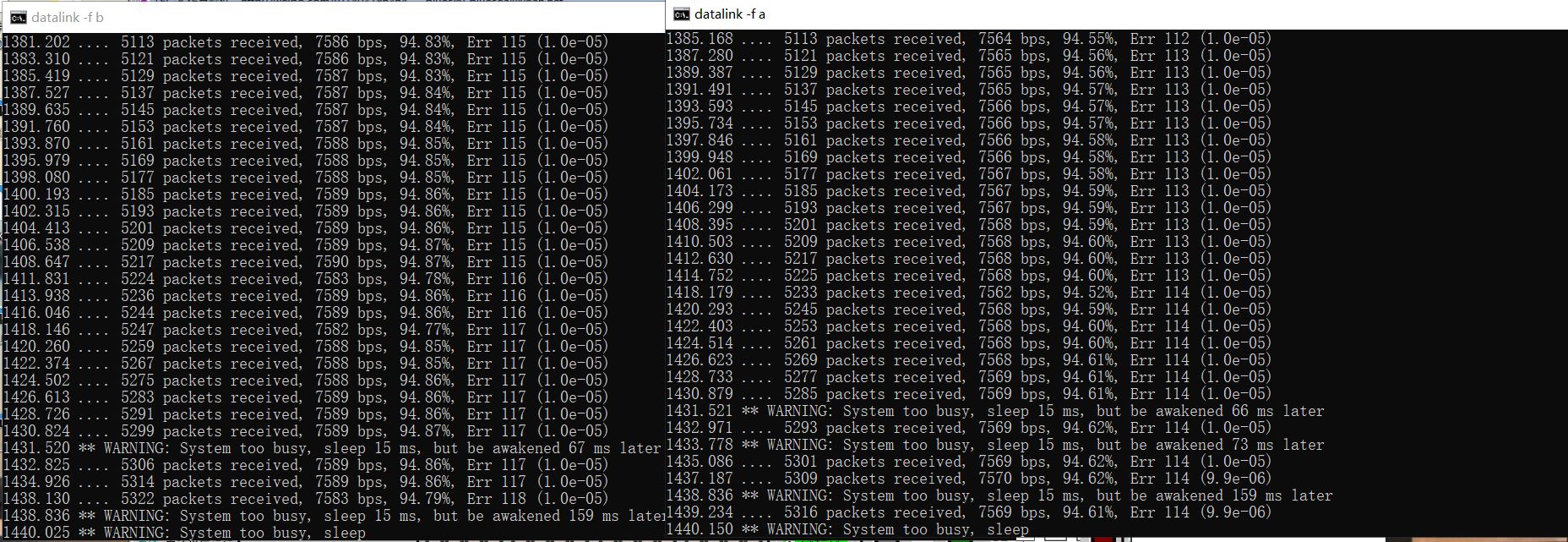
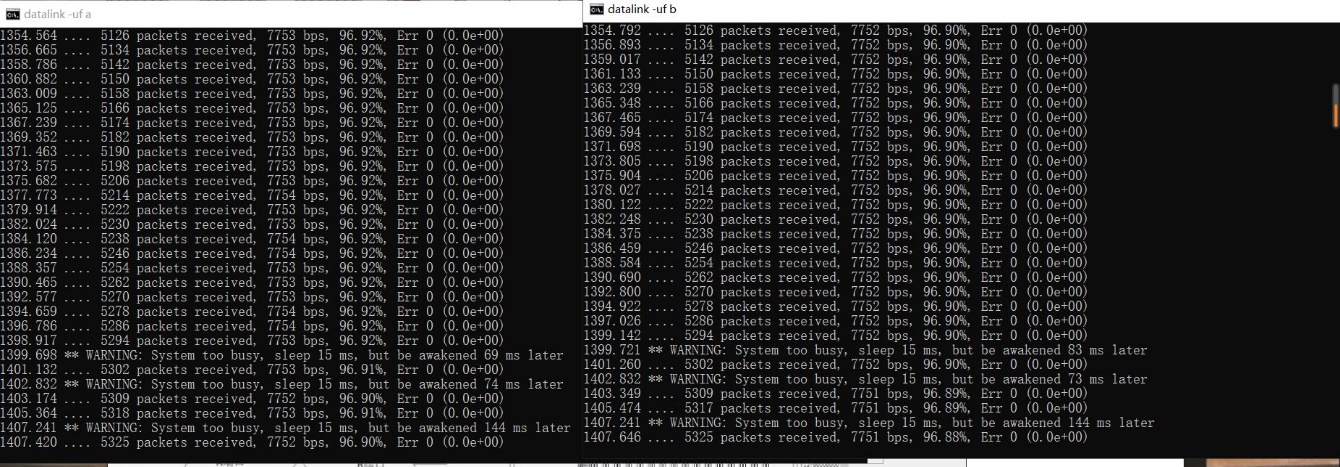
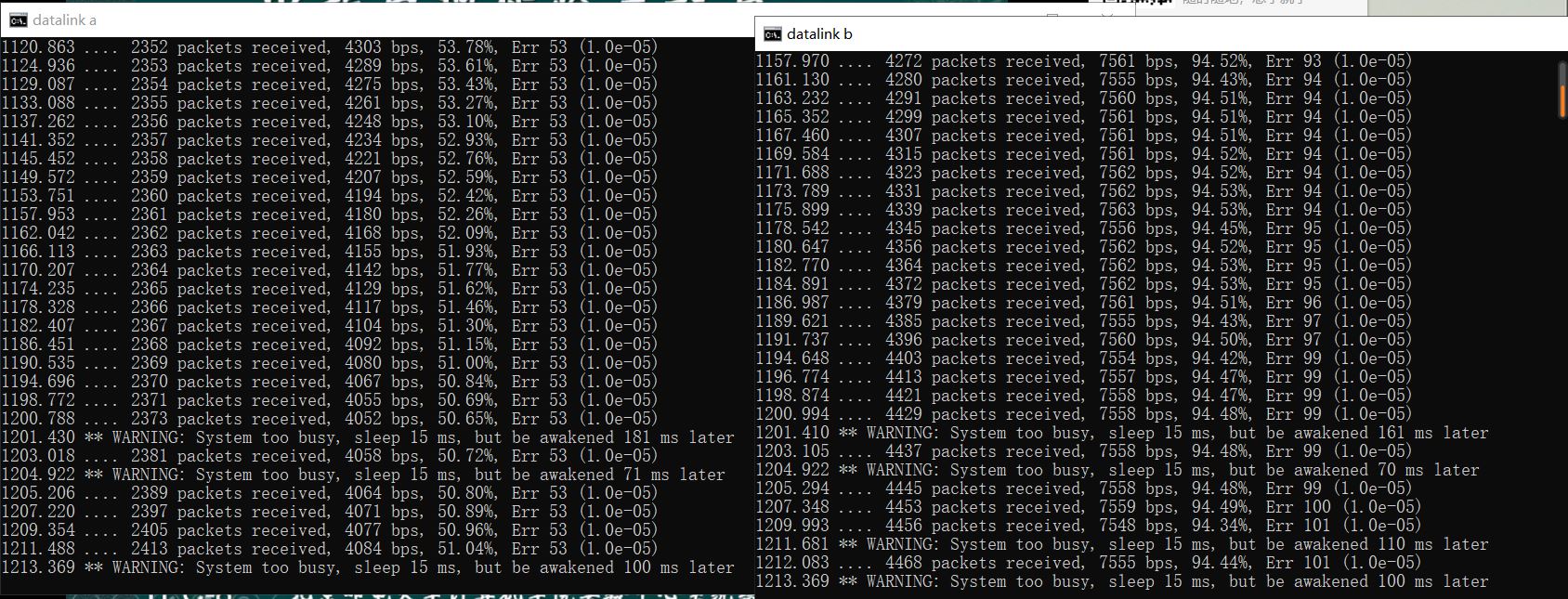
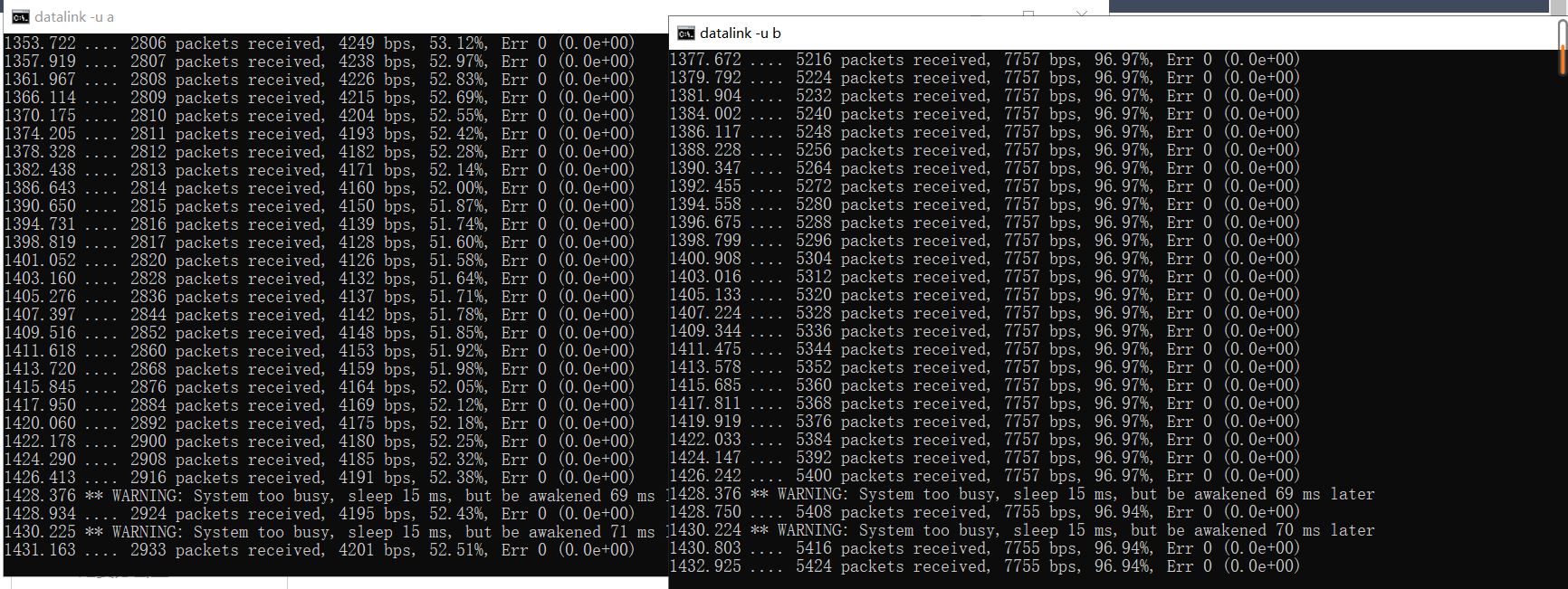
1. 实验测试：

滑动窗口机制的两个主要目标：实现有误码信道环境下的无差错传输；充分利用传输信道的带宽。完成无差错传输只是达到了基本目标，追求信道实际利用率是关键。测试以下几种情况下你所设计的软件能否正常工作20分钟，记录线路利用率。在实验报告中分析这些数据，给出实际线路利用率离极限效率之间尚有多大差距，并解释产生这样结果的原因。

**性能测试记录表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 命令选项 | 说明 | 运行时间(秒） | Selective算法  线路利用率(%) | |
| A | B |
| 1 | -–utopia | 无误码信道数据传输 | 1431 | 52.51 | 96.94 |
| 2 | 无 | 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” | 1213 | 51.04 | 94.44 |
| 3 | –-flood --utopia | 无误码信道，站点A和站点B的分组层都洪水式产生分组 | 1401 | 96.90 | 96.88 |
| 4 | –-flood | 站点A/B的分组层都洪水式产生分组 | 1460 | 94.61 | 94.79 |
| 5 | --flood -–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 | 1411 | 55.14 | 54.81 |

运行时间要求超过10分钟。（下面是实验截图）



**五 研究探索**

1. CRC 校验能力

CRC校验码具有以下检错能力：

检查出全部单个错；检查出全部离散的二位错；检查出全部奇数个错；检查出全部长度小于或等于K位的突发错。

1. 由于本次试验过程的误码信道是一个比较固定的误码率，而在实际生活当中的误码率不是稳定的，可能会因为传输环境的不同，使得他的误码率波动比较大的，例如，下雨天和晴天，高噪声和低噪声的情况，传输的距离也是影响因素。对于这种动态的误码率的通信过程，可能需要其他的一些参数来控制基本参数值（窗口大小，重传时间等等）来完成。

3.对等协议实体之间的流量控制

在我们设计的协议当中，流量的控制主要通过接收窗口，发送窗口还有确认机制来实现。因为有窗口大小的限制，发送方不会一次性发送过多信息导致接收方被信息洪流所淹没，导致信息丢失。这样可能会导致信道的利用率降低，但是如果合理的设计窗口大小，依然可以达到较高的信号利用率。

**六 实验总结**

1. 问题解决

完成本次实验代码的编写花了大约8个小时左右，选择的时间是在周三下午和晚上，其中有很大一部分时间用于调试Debug。在编程语言上遇到了对结构类型的调用问题（还C++没好好学的债）。

1. 心得体会

通过这次实验，我对完整的通信过程有了更深刻的理解。在必须动手的实验中，为了实现通信而更加细致地了解了其完整而详尽的过程；此外，在编程的过程中，对代码有了详细的分析，不管是对各个变量，还是函数中的各个参数，都能够清楚知道它所代表的含义及所起的作用。因为是单人完成，本次实验也锻炼了我统筹兼顾和编写代码的能力，使我受益匪浅。

**七 源程序**

见附页

#include <stdio.h>

#include <string.h>

#include "protocol.h"

#include "datalink.h"

typedef enum

{

false, true

}bool;

/\* 定义包类型 \*/

typedef struct

{

unsigned char info[PKT\_LEN];

}packet;

/\* 定义帧类型 \*/

typedef struct FRAME

{

unsigned char kind; /\* 帧的种类 \*/

unsigned char ack; /\* ACK号 \*/

unsigned char seq; /\* 序列号 \*/

packet data; /\* 需要被转发的数据 \*/

unsigned int padding;

}FRAME;

/\* 发送窗口上限 \*/

static unsigned char next\_frame\_to\_send;

/\* 发送窗口下限 \*/

static unsigned char ack\_expected;

/\* 接收窗口上限 \*/

static unsigned char too\_far;

/\* 接收窗口下限 \*/

static unsigned char frame\_expected;

/\* 计数，发了多少个 \*/

static unsigned char nbuffered;

/\* 数据帧超时机制 \*/

#define DATA\_TIMER 5000

/\* ACK超时机制 \*/

#define ACK\_TIMER 280

/\* 帧的最大长度 \*/

#define MAX\_SEQ 31

/\* 窗口大小 \*/

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

/\* 实现k的按模 +1 \*/

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

/\* phl\_ready = 0 意味着物理层还没有准备好 \*/

static int phl\_ready = 0;

/\* no\_nak = true 意味着还没有发送过一个NAK \*/

bool no\_nak = true;

/\* 判断数据帧是否位于接受窗口内，来判断是否暂存 \*/

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));

}

/\* 向网络层上交数据帧 \*/

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

{

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

send\_frame(frame, len + 4);

phl\_ready = 0;

}

/\* 发出帧，此处按帧的类型用if语句进行不同的操作，所以我将三个函数合在一起 \*/

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

{

/\* 创建一个空白新帧备用 \*/

FRAME s;

s.kind = fk;

/\* s.seq 只对数据帧有意义 \*/

s.seq = frame\_nr;

/\* 累积确认，表示最后一个正确收到的 \*/

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

/\* 如果是数据帧 \*/

if (fk == FRAME\_DATA)

{

/\* 将数据帧复制到 s 中去 \*/

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

/\* 打印信息 \*/

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

/\* 将数据帧下发给物理层 \*/

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

/\* 发送数据帧时，启动数据帧计时器 \*/

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

}

/\* 如果是NAK帧 \*/

if (fk == FRAME\_NAK)

{

/\* 已经发了一个NAK，no\_nak置false \*/

no\_nak = false;

/\* 将NAK帧下发给物理层 \*/

put\_frame((unsigned char\*)&s, 2);

}

/\* 如果是ACK帧 \*/

if (fk == FRAME\_ACK)

{

/\* 打印 \*/

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

/\* 将ACK帧下发给物理层 \*/

put\_frame((unsigned char\*)&s, 2);

}

/\* 发送帧时停止 \*/

stop\_ack\_timer();

}

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

{

/\*

事件，在protocol.h中已完成定义

#define NETWORK\_LAYER\_READY 0

#define PHYSICAL\_LAYER\_READY 1

#define FRAME\_RECEIVED 2

#define DATA\_TIMEOUT 3

#define ACK\_TIMEOUT 4

\*/

int event, arg;

FRAME f;

int len = 0;

int i;

/\* 发送窗口的“椅子” \*/

packet out\_buf[NR\_BUFS];

/\* 接受窗口的“椅子” \*/

packet in\_buf[NR\_BUFS];

/\* 收到的数据帧该坐的“椅子”当前是否空闲 \*/

bool arrived[NR\_BUFS];

/\* 初始化 \*/

/\* 接收端初始化时，enable\_network\_layer() 被抑制 \*/

enable\_network\_layer();

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;

}

/\* Create Communication Sockets \*/

protocol\_init(argc, argv);

/\* 打印作者信息、时间 \*/

lprintf("Designed by 2018210074 - Xiong Yu, build: " \_\_DATE\_\_" "\_\_TIME\_\_"\n");

for (; ; )

{

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

switch (event)

{

/\* 对于发送端，网络层有信息要发送 \*/

case NETWORK\_LAYER\_READY:

/\* 为发新帧做准备 \*/

nbuffered++;

/\* 取数据 \*/

get\_packet(out\_buf[next\_frame\_to\_send % NR\_BUFS].info);

/\* 发送DATA型数据帧 \*/

send\_data(FRAME\_DATA, next\_frame\_to\_send, frame\_expected, out\_buf);

/\* 发送窗口上限 +1 \*/

inc(next\_frame\_to\_send);

break;

/\* 物理层有信息来到 \*/

case PHYSICAL\_LAYER\_READY:

phl\_ready = 1;

break;

/\* 接收到帧 \*/

case FRAME\_RECEIVED:

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

/\* 检查帧是否正常 \*/

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

{

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

/\* 如果之前没发过NAK，那么就要发一个NAK \*/

if (no\_nak)

{

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

}

break;

}

/\*if (f.kind == FRAME\_ACK)

dbg\_frame("Recv ACK %d\n", f.ack);\*/

/\*

如果是数据帧，则检查帧序号，落在接收窗口内接收，否则丢弃

如果不等于接收窗口下限，且没发过NAK，还得发一个NAK

\*/

if (f.kind == FRAME\_DATA)

{

if ((f.seq != frame\_expected) && no\_nak)

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

else

/\* 无法捎带，则单发ACK \*/

start\_ack\_timer(ACK\_TIMER);

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

{

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

/\* 这个“椅子”有人坐了，不再空闲 \*/

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

/\* 信息写入该椅子 \*/

in\_buf[f.seq % NR\_BUFS] = f.data;

/\* 确保按次序依次上交，当frame\_expected这把椅子为true时才上交 \*/

while (arrived[frame\_expected % NR\_BUFS])

{

put\_packet(in\_buf[frame\_expected % NR\_BUFS].info, len - 7);

no\_nak = true;

/\* 让接收端的该椅子仍然可以接受数据 \*/

arrived[frame\_expected % NR\_BUFS] = false;

/\* 接收窗口上限、下限 +1 \*/

inc(too\_far);

inc(frame\_expected);

start\_ack\_timer(ACK\_TIMER);

}

}

}

/\* 如果是NAK帧，则选择重传 \*/

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

{

dbg\_frame("Recv NAK DATA %d\n", f.ack);

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

}

/\* 检查确认序号，落在发送窗口内则移动发送窗口，否则不做处理 \*/

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

{

nbuffered--;

/\* 收到正确确认，停止计时器 \*/

stop\_timer(ack\_expected % NR\_BUFS);

inc(ack\_expected);

}

break;

/\* ACK超时，就单发一个ACK \*/

case ACK\_TIMEOUT:

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

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

break;

/\* 超时 选择重传\*/

case DATA\_TIMEOUT:

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

if (!between(ack\_expected, arg, next\_frame\_to\_send))

arg = arg + NR\_BUFS;

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

break;

}

/\* 内存没满，继续进行 \*/

if (nbuffered < NR\_BUFS && phl\_ready)

enable\_network\_layer();

/\* 内存满了，停止 \*/

else

disable\_network\_layer();

}

}