

实验一

数据链路层

滑动窗口协议的设计与实现

计算机学院

基本内容

■ 实验内容

- ◆ 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两站点间无差错双工通信

■ 信道模型

- ◆ 8000bps全双工卫星信道
- ◆ 单向传播时延270毫秒
- ◆ 信道误码率为 10^{-5}
- ◆ 物理层接口：提供帧传输服务，帧间有1ms帧边界
- ◆ 网络层属性：分组长度固定256字节

■ 实验组人数

- ◆ 1~3人

■ 实验设备环境

- ◆ WindowsXP, Microsoft Visual Studio 2013+
- ◆ Linux (Ubuntu, Fedora, RedHat)

实验步骤

■ 熟悉编程环境

- ◆ 安装好VC或兼容的更高版本的C语言编程环境
- ◆ 了解程序的主体运行框架
- ◆ 可利用的子程序

■ 协议设计和程序总体设计

- ◆ 设计好要实现的滑动窗口协议，定义帧字段，规划程序的总体结构，相关子程序的设置

■ 编码和调试

- ◆ 将所设计的协议编码实现并上机调试通过，实现数据链路层两个站点之间的通信。

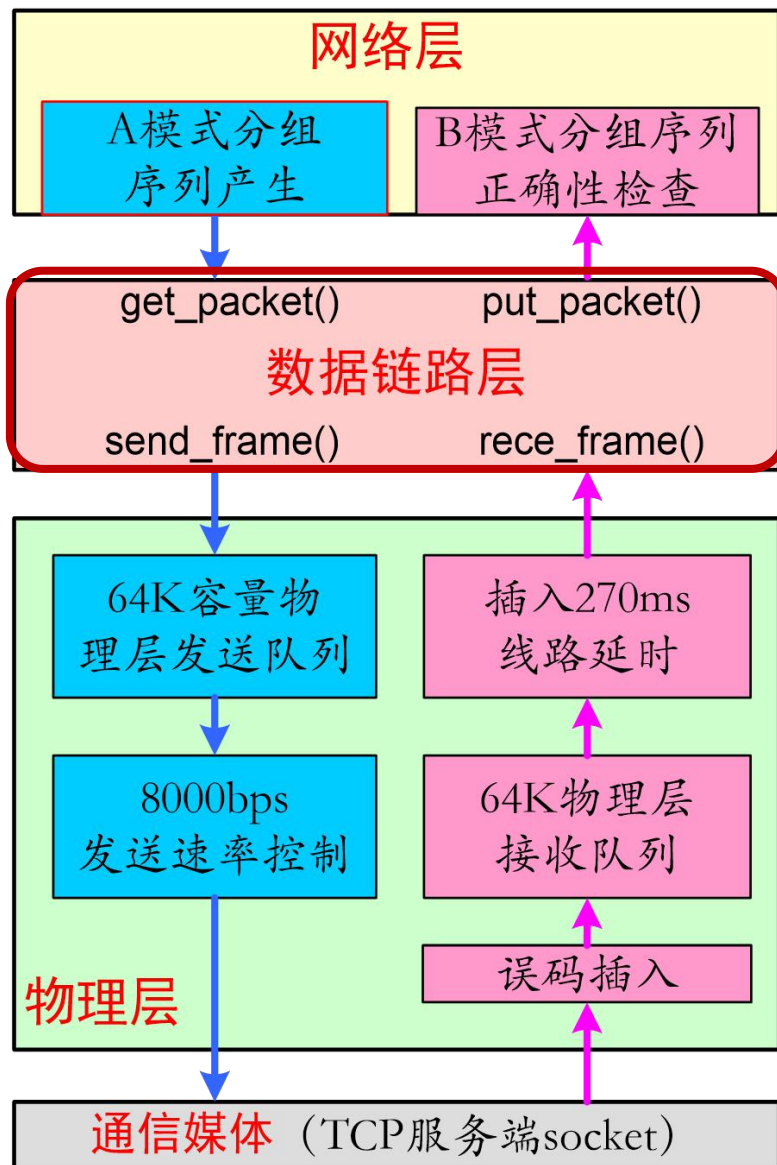
■ 软件测试和性能评价

- ◆ 在无误码信道环境下运行测试
- ◆ 有误码信道环境下的无差错传输
- ◆ 要求：稳定运行20分钟以上，效率不能太低

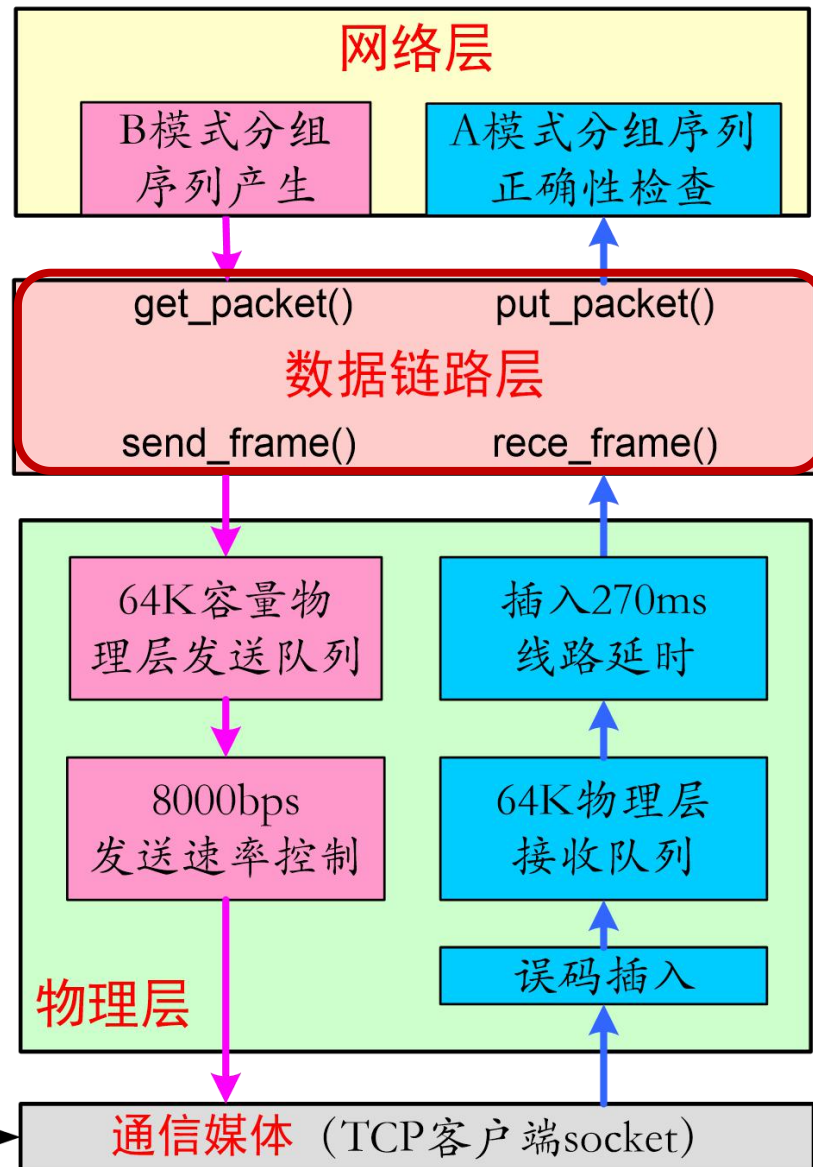
■ 实验报告及程序验收

总体结构

Station A



Station B



Windows环境编译和运行

Lab1-Windows

datalink.c

datalink.h

protocol.h

protocol.c
lprintf.c
lprintf.h
crc32.c
getopt.h
getopt.c

■ 编译

■ 程序运行（启动两个进程）

A站: stopwait.exe -d3 a

B站: stopwait.exe -d3 b

■ 产生的日志文件

stopwait-A.log

stopwait-B.log

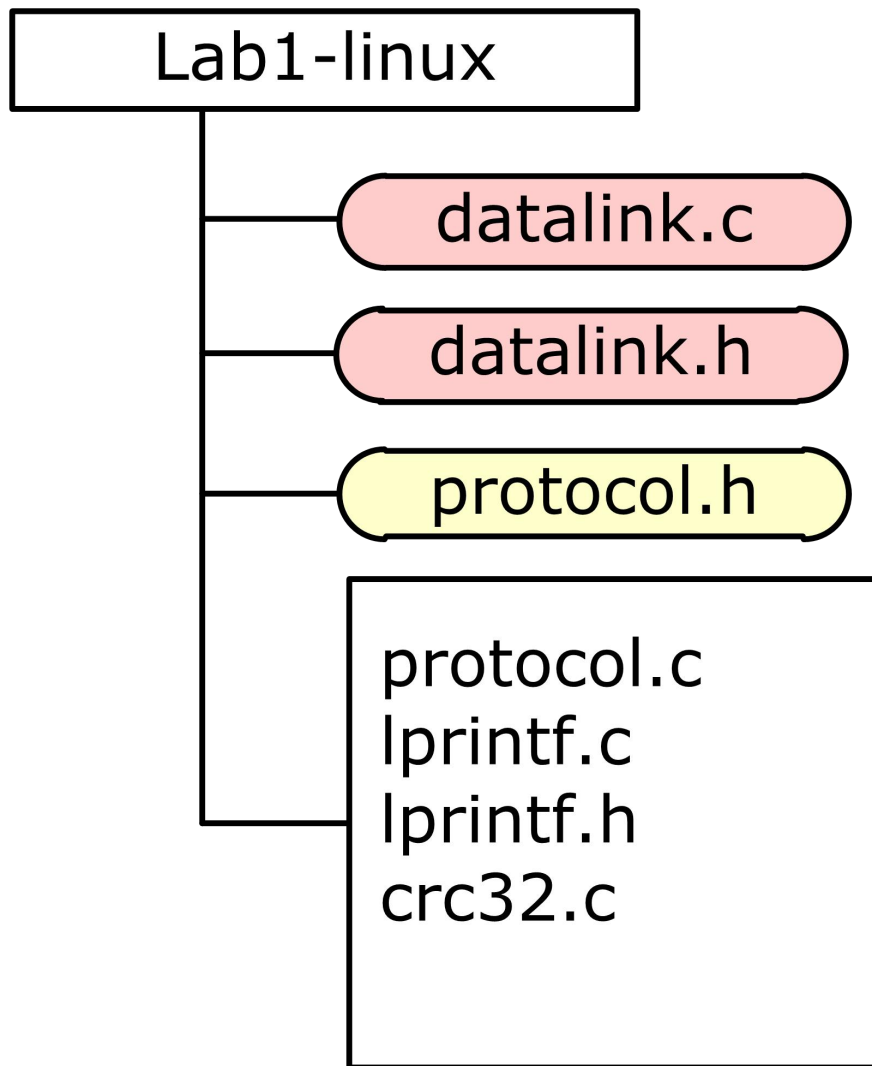
■ 目录Example下的可执行文件

stopwait.exe

gobackn.exe

selective.exe

Linux环境编译和运行



■ 操作系统

RedHat, Fedora
Ubuntu

■ 编译

make

■ 程序运行（启动两个进程）

A站: ./stopwait -d3 A

B站: ./stopwait -d3 B

■ 产生的日志文件

stopwait-A.log

stopwait-B.log

■ 目录Example下的可执行文件

stopwait

gobackn

selective

程序运行：命令行选项

- 启动执行EXE文件时，在命令行中附带一些选项对程序的执行进行控制

Usage:

`./stopwait <options> <station-name>`

Options :

- ?, --help : print this
- u, --utopia : utopia channel (an error-free channel)
- f, --flood : flood traffic
- i, --ibib : set station B layer3 sender mode as IDLE-BUSY-IDLE-BUSY-...
- n, --nolog : do not create log file
- d, --debug=<0-7>: debug mask (bit0:event, bit1:frame, bit2:warning)
- p, --port=<port#> : TCP port number (default: 59144)
- b, --ber=<ber> : Bit Error Rate (received data only)
- l, --log=<filename> : using assigned file as log file
- t, --ttl=<seconds> : set time-to-live

■ 命令行选项使用举例

`stopwait -fnd 3 -b 1.0e-6 -p 44944 A`

`stopwait -fn -debug=3 --ber=1.0e-6 --port=44944 A`

日志函数

■ 函数

```
extern void lprintf(char *fmt, ...);
```

■ 举例

```
lprintf("Received a frame, %d bytes\n", len);
```

（1）在每行最左端自动增加时间戳（ms级）

（2）把显示的信息同步存于日志文件中

上下层接口函数

■ 运行环境的初始化

```
void protocol_init(int argc, char **argv);
```

■ 与网络层模块的接口

```
#define PKT_LEN 256
```

```
void enable_network_layer(void);
```

```
void disable_network_layer(void);
```

```
int get_packet(unsigned char *packet);
```

```
void put_packet(unsigned char *packet, int len);
```

■ 与物理层模块的接口

```
int recv_frame(char *buf, int size);
```

```
void send_frame(char *buf, int len);
```

事件驱动函数

```
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
```

样例程序datalink.c

- 样例程序实现了简单的全双工“停-等”协议
 - ◆ 未设ACK定时器，收到数据就立刻回复ACK
 - ◆ 未实现NAK
- 编辑，编译
- 运行
 - ◆ 分别在两个DOS窗口运行stopwait a和stopwait b，那么会启动两个站运行。
 - ◆ 如果运行stopwait -d3 a和stopwait -d3 b，那么，会打印出协议运行信息。协议运行信息的输出，也是在datalink.c中设定的

CRC校验和的产生与验证

```
unsigned int crc32(unsigned char *buf, int len);
```

■ 校验和产生

char *p; 为p指向的缓冲区内243字节数据生成校验和，并把校验和附在243字节之后

```
*(unsigned int *)(p + 243) = crc32(p, 243);
```

p所指缓冲区必须至少有247字节有效空间以防内存访问越界

■ 验证校验和

◆ 针对对上面的例子，只需要判断`crc32(p, 243 + 4)`是否为0：校验和正确为0，否则不为0

定时器管理

■ 数据定时器

```
void start_timer(unsigned int nr, unsigned int ms);
```

```
void stop_timer(unsigned int nr);
```

- ◆ 定时器启动时刻不是当前时刻，而是将当前物理层发送队列的数据发送完毕后开始启动计时
- ◆ 重复设置同一个编号的计时器会导致重新按新调用计时
- ◆ “计时时间到”会产生DATA_TIMEOUT事件

■ ACK定时器

```
void start_ack_timer(unsigned int ms);
```

```
void stop_ack_timer(void);
```

- ◆ 定时器启动时刻为当前时刻
- ◆ 在先前启动的定时器未超时之前重新执行start_ack_timer()调用，定时器将依然按照先前的时间设置产生事件ACK_TIMEOUT

协议工作过程的跟踪和调试

■ 相关函数

- ◆ `extern void dbg_event(char *fmt, ...);`
- ◆ `extern void dbg_frame(char *fmt, ...);`
- ◆ `extern void dbg_warning(char *fmt, ...);`
- ◆ `char *station_name(void);`

■ 程序内部的输出控制开关debug_mask

- ◆ bit0为1时，打开dbg_event的输出，否则被忽略
- ◆ bit1为1时，打开dbg_frame的输出，否则被忽略
- ◆ bit2为1时，打开dbg_warning的输出，否则被忽略
- ◆ 命令行debug选项0~7为debug_mask赋值（默认0）

程序运行异常中止的错误信息

| 类别 | 错误信息及说明 |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 参数错 | Station name must be 'A' or 'B' |
| 链路层工作失败 | Network Layer: incorrect packet length Network Layer received a bad packet from data link layer |
| TCP通信故障 | Station A failed to bind TCP port Station B failed to connect station A |
| 物理层发送队列溢出 | Physical ayer Sending Queue overflow 物理层发送队列溢出（队列最多可以缓冲64K字节） |
| 操作系统环境问题 | WARNING: System too busy, sleep 15 ms, but be awakened 61 ms later 警告信息，不会导致程序中止运行，但可能影响算法的线路利用率统计指标(检测到系统忙碌：进程主动请求睡眠15ms，但是被唤醒后发现时间已逝去61ms。关闭系统中其它运行程序) |
| 函数调用错 | recv_byte(): Receiving Queue is empty 未产生FRAME_RECEIVED事件就执行recv_frame() get_packet(): Network layer is not ready for a new packet 未产生NETWORK_LAYER_READY事件就执行get_packet() start_timer(): timer No. must be 0~128 系统最多支持129个定时器，定时器编号太大 |

实验报告要求

■ 实验内容和实验环境描述

■ 协议设计

- ◆ 帧中各个字段的定义和编码
- ◆ 两个站点间信息交换的过程控制，尤其是误码条件下的控制方案

■ 软件设计

- ◆ 数据结构， 模块结构， 算法流程

■ 实验结果分析

- ◆ 理论分析
- ◆ “性能测试记录表”， 实验结果分析
- ◆ 存在的问题和改进思路

■ 研究和探索的问题

■ 实验总结和心得体会

- ◆ 上机调试时间， 编程语言方面， 协议方面

■ 源程序文件

关于验收

- 分组：1-3人一组，每组完成一个程序、提交一份实验报告
- 验收时间：返校后

Any Questions?