**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** |  |
| **班 级** | **班内序号** | **学 号** | | **学生姓名** | **成绩** | |
| **2017211311** | **8** | **2017211416** | | **张志博** |  | |
| **2017211311** | **9** | **2017211417** | | **田维翰** |  | |
| **2017211311** | **10** | **2017211418** | | **赵佳祺** |  | |
| **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，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。

1. 实验环境描述
   1. Windows 10 1809
      1. Microsoft Visual Studio 2019
   2. docker
      1. Ubuntu 18.04.1 LTS
      2. GNU Make 4.1
      3. gcc version 7.3.0 (Ubuntu 7.3.0-27ubuntu1~18.04)
2. 协议设计
   1. 帧中各个字段的定义和编码

typedef struct FRAME {

unsigned char kind;

unsigned char ack;

unsigned char seq;

unsigned char data[PKT\_LEN];

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) |

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

\*/

* + 1. kind 变量记录本数据帧的类型信息，分为数据帧、ACK桢、NAK桢三种，其中以宏常量 FRAME\_DATA 表示“数据帧”，以宏常量 FRAME\_ACK 表示“ACK桢”，以宏常量 FRAME\_NAK 表示“NAK桢”。
    2. ack 变量记录本数据帧的 ack 编号。
    3. seq 为本数据帧自身的编号。
    4. data 数组为来自网络层的数据包数据，其最大长度为 256，由 protocol.h 中的宏常量 PKT\_LEN 定义。
    5. padding 为 CRC 校验码数据，由 crc32.c 中的 unsigned int crc32(unsigned char \*buf, int len) 计算得。
  1. 两个站点间信息交换的过程控制
     1. 首先初始化物理层，当物理层初始化结束后，值为 `PHYSICAL\_LAYER\_READY` 的事件被捕捉，执行 `enable\_network\_layer();` 启用网络层，开始接收数据包；
     2. 当网络层被启用后，若网络层有数据要发送给接收方，首先增大已缓存包的计数器，再从网络层取得数据包存入将要送出的数据缓存区，缓存序号为将要送出的数据桢序号对缓冲区大小求余结果。然后调用本实验中设计的传输协议发送该数据包，停止 ack 计时器，数据包将被包装成“数据”类型的数据帧。最终增大“将要送出的数据桢”计数器数值。
     3. 若事件类型为“收到来自物理层的数据”，则开始对收到的数据帧进行处理。首先对数据帧进行校验，根据crc协议规定，如果校验结果非0，收到的数据桢出现了错误，此时立即丢弃该数据帧并发送 NAK 桢以示传输错误。如果crc校验正确，那么根据数据桢类型进行分支。
        1. 如果数据帧为“FRAME\_DATA”类型，即带有数据的数据帧，首先对帧序号进行判断，详细流程见二.3.1，否则，启动 ACK 计时器，解决当反向数据包过少时无法及时传达 ACK 信息的问题。当收到的帧序号落在协议合法范围内且这个桢以前并未收到过时，标记此桢缓冲区序号的状态为“已收到”，将数据桢缓存区中自下界到最后一个“已收到”桢的连续桢的数据内容发送给网络层，与此同时，标记已经发送给网络层的桢对应的缓冲区序号状态为“未收到”，向前移动接收窗口上下界，启动 ACK 计时器。
        2. 如果数据桢类型为"FRAME\_ACK"，打印 debug 数据后不作处理。
        3. 如果数据帧类型为"FRAME\_NAK"，进行 NAK 处理逻辑，详见二.3.2.
        4. 以上处理结束后，根据收到桢的 ack 序号，停止自应收 ack 序号下界到当前桢 ack 序号之间桢的 ack 计时器，并同时减小发送缓冲区计数器的值。
     4. 如果事件类型为“发送数据帧的计时器超时”，则向物理层重新发送超时的数据帧。
     5. 如果事件类型为“ACK 计时器超时”，则向物理层发送含对应 ack 序号的数据桢。
  2. 误码条件下的控制方案
     1. 对桢序号进行判断的流程如下，当非 nak 状态下接收到的数据桢序号不是本机应该收到的数据桢序号，则立即发送 NAK 桢。
     2. NAK 处理逻辑如下，若类型为 NAK 的桢的 ack 序号落在本站还未收到 ACK 的桢范围内，重新向物理层发送 ack 序号后1对应发送缓冲区中桢的数据。

1. 软件设计
   1. 数据结构
      1. 结构体

typedef struct FRAME {

unsigned char kind;

unsigned char ack;

unsigned char seq;

unsigned char data[PKT\_LEN];

unsigned int padding;

} FRAME;

1. 全局变量
   1. static bool no\_nak; // 还没有发送nak
   2. static bool phl\_ready;
   3. static unsigned char ack\_expected; // 发送窗口的下边缘
   4. static unsigned char next\_frame\_to\_send; // 发送窗口的上边缘+1
   5. static unsigned char frame\_expected; // 接收窗口的下边缘
   6. static unsigned char too\_far; // 接收窗口的上边缘+1
   7. static unsigned char out\_buf[NR\_BUFS][PKT\_LEN]; // 出站流的缓冲区
2. 主函数中的变量
   1. FRAME f;
   2. unsigned char in\_buf[NR\_BUFS][PKT\_LEN]; // 入站流的缓冲区
   3. bool arrived[NR\_BUFS]; // 入站位图
   4. unsigned char nbuffered; // 缓冲的数据包
   5. int len; // 从物理层获取传入帧的长度
   6. int arg; // event = wait\_for\_event(&arg)
   7. int event； // 5种事件：NETWORK\_LAYER\_READY、PHYSICAL\_LAYER\_READY、FRAME\_RECEIVED、DATA\_TIMEOUT、ACK\_TIMEOUT
   8. 模块结构
      1. 程序
         1. static bool between(unsigned char a, unsigned char b, unsigned char c)
            1. 作用：判断当前帧是否落在发送/接收窗口内
            2. 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\_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
         2. static unsigned char inc(unsigned char a)
            1. 作用：使一个字节在0-MAX\_SEQ的范围内循环自增
            2. 参数：a，字节类型
         3. static void put\_frame(unsigned char\* frame, int len)
            1. 作用：为一个帧做CRC校验，填充至帧的尾部并将其递交给网络层发送
            2. 参数：frame，字节数组，由除padding域之外的帧内容转换而来；len，整型，为帧当前的长度
         4. static void send\_frame\_k(unsigned char fk, unsigned char frame\_nr)
            1. 作用：构造并发送数据、ack或nak帧
            2. 参数：fk，字节类型，为帧的内容；next\_frame，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；out\_buf，二维字节数组，为缓冲区
         5. int main(int argc, char\*\* argv)
            1. 作用：主程式，包含选择重传协议的算法流程
            2. 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
      2. 程序调用关系图

main -> protocol\_init

-> disable\_network\_layer

-> enable\_network\_layer

-> wait\_for\_event

-> get\_packet

-> send\_frame\_k -> put\_frame -> send\_frame

-> start\_timer

-> stop\_ack\_timer

-> dbg\_frame

-> start\_ack\_timer

-> stop timer

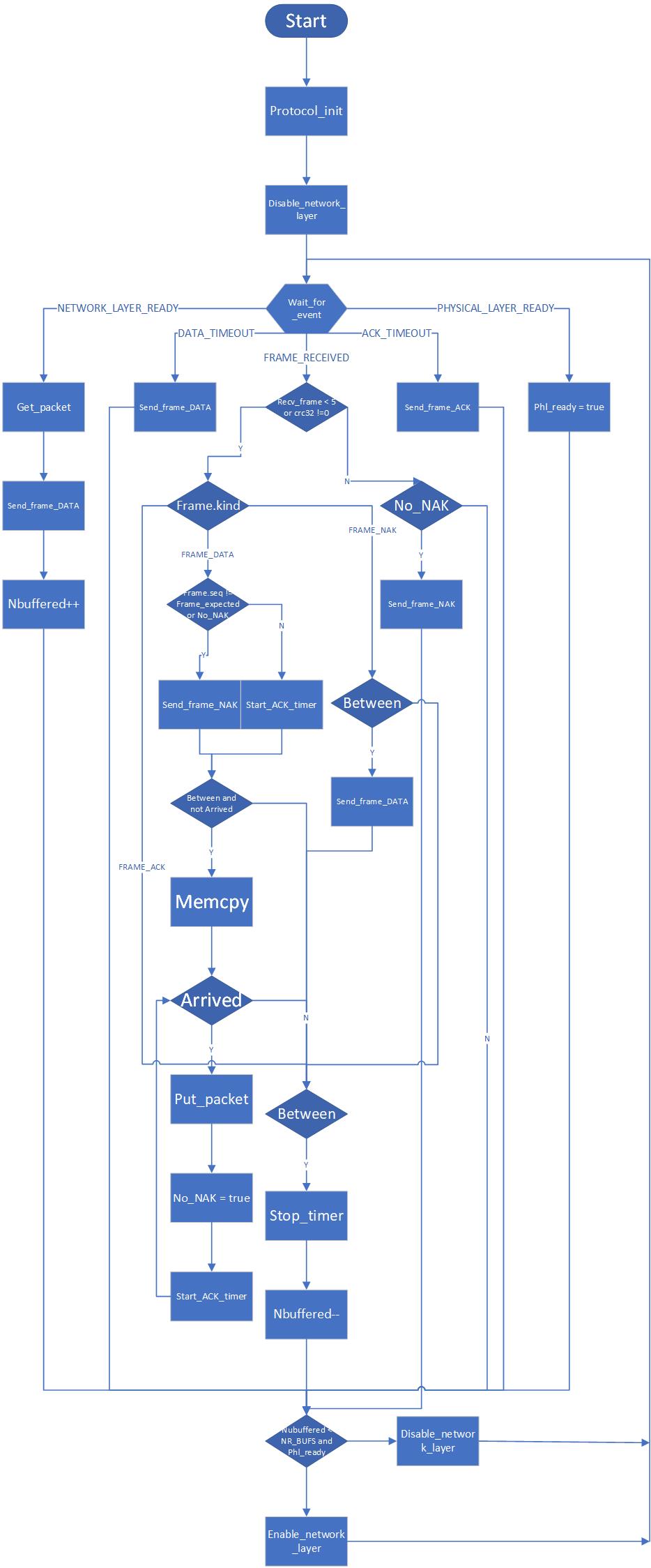
-> crc32

-> dbg\_event

-> between

-> inc

* 1. 算法流程



1. 实验结果分析
   1. 理论分析

由于需要携带帧信息，所以最大信息利用率为，由于数据链路层提供的服务为8000bps，所以每传输一个字节耗时1ms，每帧的附加信息固定为10，耗时10ms，若出现转义字符还将增加时间。

简化模型，假设信道上始终有数据需要传输。则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。假设在限定时间内可以重传的该帧为正确帧，则每传送48个数据包需传送48+1+1=50次。所以信道利用率为而由于程序设计原因，当一个数据包超时后，常常需要重传多次造成信道浪费。若重传k次，则信道利用率为，若重传10次，信道利用率为78.18%。若信道误码率为，则，即大约每5个帧就有一个出错，此时在ESC/FLAG模式下平均250个字符需要两倍的传输空间即极限值510，若平均每个错帧重传10次信道利用率的极值为

* 1. 性能测试记录表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 命令选项 | 说明 | 运行时间(秒） | Selective算法3000/1000/16  线路利用率(%) | |
| A | B |
| 1 | -–utopia | 无误码信道数据传输 | 1200+ | 57.9 | 97.0 |
| 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” | 1200+ | 56.3 | 94.5 |
| 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 | 1200+ | 97.0 | 97.0 |
| 4 | –-flood | 站点A/B的分组层都洪水式产生分组 | 1200+ | 93.6 | 92.6 |
| 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 | 1800+ | 59.7 | 57.5 |

* 1. 实验结果分析
     1. 实现的协议软件是实现了有误码信道环境中无差错传输功能。

如果收到一个损坏帧，在尚未发送否认确定的情况下，则捎带发送否定确认。并打印错误报告。如果已发送否定确认，则只打印错误报告，不重复发送否定确认。

* + 1. 程序的健壮性优秀，可以可靠地长时间运行。

程序成功连续运行半小时以上，并取得预定效果，具有足够的健壮性。

* + 1. 协议参数的选取

物理层提供的是字节流传输服务，使用字节填充技术成帧，分组长度为256字节。为了避免在有出错帧接收方要求重传时产生二义性，我们定义窗口大小为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。

滑动窗口的大小直接涉及到信道利用率和数据拥塞问题，若太大，数据发送过快将产生拥塞导致数据丢失，出错率增加，若太小则信道利用率降低，通过实验测试合适的窗口大小为16。重传定时器时限涉及到重传的响应时间，太小会导致频繁重传，太大则重传等待时间太久，经过我们的试验测试，选取重传定时器时限定为3000毫秒，ACK 搭载定时器的时限为1000毫秒最合适。

* 1. 存在的问题和改进思路

实验顺利进行，每次测试得到的数据都比较接近，但和理论值相比信道利用率较低，这可能与信道的负载有关。由实验结果可以看出，窗口大小是16时信道利用率最高，与理论最大值最接近。在洪泛模式下，信道利用率与理论值接近。而在其他情况下负载有时较轻，所以信道利用率相比理论值较低。

1. 研究和探索的问题
   1. CRC 校验能力

CRC校验码的检错能力很强，它除了能检查出离散错外，还能检查出突发错，CRC校验码具有以下检错能力：CRC校验码能检查出全部单个错；CRC校验码能检查出全部离散的二位错；CRC校验码能检查出全部奇数错误；CRC校验码能检查出全部长度小于或等于K位的突发错；CRC校验码能以[1-（1/2）K-1]的概率检查出长度为（K+1）位的突发错。

本实验采用 CRC32 ，对于 32 位及以下个数的误码可以检出，对于奇数个误码个数也可以检出。所以不妨假设误码率为 1e-4，那么发生超过 34 位误码的概率不会大于 2 \* (0.0001 \*\* (34 - 2))，信道速率为 8000 bps，约为每秒 40 桢，使用时间 50% 则约 10 \*\* 127 / 4 秒会出现未检出的错误，约为 8 \* (10 \*\* 119)年。

如果客户仍不满意，则可以使用改进的校验码算法，例如一种改进的旋转 ＣＲＣ数据校验设计方法。(<http://journal.nudt.edu.cn/publish_article/2011/6/201106008.pdf>)

* 1. CRC 校验和的计算方法

CRC查表法本质上是以牺牲空间复杂度来换取时间复杂度，也就是将一部分计算结果事先储存成表，然后在实际计算时对表查找，缩短计算时间。

表格实际上就是多次移位校验多项式后迭代异或的结果。

如果使用 CRC16，计算时间不会较 CRC32 增加一倍，采用分组的快速查表法，计算时间相似。

RFC1662 中给出的 pppfcs32 函数可以从给定的 fcs 继续进行运算，当 crc32 前有更多位数的快速校验算法时，crc32 作为末尾计算方式，可以从前者继承计算结果，从而继续计算。

* 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。
  2. 程序设计方面的问题
     1. get\_ms()如何实现

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。

* + 1. ack\_timer 与 timer

ack\_timer 无需针对特定的桢进行计时，而用于数据超时的 timer 则需要对缓冲区序号对应的桢进行超时计时。

* 1. 软件测试方面的问题
     1. 设计五种测试方案，目的在于测试程序在各种不同条件下的行为是否在书写者预期内，如果程序行为超出预期，则应针对发生的状况进行评估，调整。

| **序号** | **可能导致异常的程序问题** |
| --- | --- |
| 1 | 数据流串接错误，无法将原始数据传递到远程终端 |
| 2 | 未正确实现timer，未正确实现误码检测算法 |
| 3 | 流控不完善，导致信息丢失或buffer溢出 |
| 4 | 应对2与3共同作用的情况 |
| 5 | 应对更高的误码率，若检验算法性能不佳，窗口大小等超参效果不好，则会导致程序崩溃 |

* + 1. 不能模拟真实环境中会变化的超参情况，线路短时中断的情况。
    2. 组合多样性的硬件软件平台，对此软件进行多次测试。
    3. 可尝试采用 ci 自动构建测试方式，节省开发员构建时间。
  1. 对等协议实体之间的流量控制

在设计的协议当中，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。

1. 实验总结和心得体会
   1. 上机调试时间

完成本次试验代码的编写大约使用了2-3天，每天大约使用了2-3小时的时间，其中部分时间使用在程序框架的确定和DEBUG；在这之后又使用了几天于测试协议参数的选取等问题。

* 1. 编程语言方面

C语言是面向过程的语言，基本没有困难。

* 1. 协议方面

因为有PPT、PDF和书本等资料，除调试时出现了一些逻辑错误，没有遇到过复杂的问题。

通过这次试验，我们对数据链路层的选择重传协议的机制有了很深刻的了解。很多书本上理解不深的讲义，在我们进行试验的过程中，通过对模拟结果的分析与思考都逐渐的理解了。而且，此次试验是我们第一次模拟通信，让我们学会了windows下观察网络收发数据包模拟环境的搭建。

1. 源程序文件
   1. datalink.h

#define DATA\_TIMER 3000

#define ACK\_TIMER 1000

#define MAX\_SEQ 15 // should be 2^n-1

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

/\* FRAME kind \*/

#define FRAME\_DATA 1

#define FRAME\_ACK 2

#define FRAME\_NAK 3

/\*

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) |

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

\*/

typedef struct FRAME {

unsigned char kind;

unsigned char ack;

unsigned char seq;

unsigned char data[PKT\_LEN];

unsigned int padding;

} FRAME;

* 1. datalink.c

#include <stdio.h>

#include <string.h>

#include <stdbool.h>

#include "protocol.h"

#include "datalink.h"

static bool no\_nak = true; // no nak has been sent yet

static bool phl\_ready = false;

static unsigned char ack\_expected = 0; // lower edge of sender's window

static unsigned char next\_frame\_to\_send = 0; // upper edge of sender's window +1

static unsigned char frame\_expected = 0; // lower edge of receiver's window

static unsigned char too\_far = NR\_BUFS; // upper edge of receiver's window +1

static unsigned char out\_buf[NR\_BUFS][PKT\_LEN]; // buffers for the outbound stream

static bool between(unsigned char a, unsigned char b, unsigned char c) {

return a <= b && b < c || c < a && a <= b || b < c && c < a;

}

static unsigned char inc(unsigned char a) {

return (a + 1) % (MAX\_SEQ + 1);

}

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

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

send\_frame(frame, len + 4);

phl\_ready = false;

}

// construct and send a data, ack, or nak frame

static void send\_frame\_k(unsigned char fk, unsigned char frame\_nr) {

FRAME s; // scratch variable

s.kind = fk; // kind == data, ack, or nak

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

s.seq = frame\_nr; // only meaningful for data frames

switch (fk) {

case FRAME\_DATA:

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

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

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

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

break;

case FRAME\_ACK:

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

put\_frame((unsigned char\*)& s, 2); // transmit the frame

break;

case FRAME\_NAK:

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

put\_frame((unsigned char\*)& s, 2); // transmit the frame

no\_nak = false;

break;

}

phl\_ready = false; //

stop\_ack\_timer(); // no need for separte ack frame

}

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

FRAME f; // scratch variable

unsigned char in\_buf[NR\_BUFS][PKT\_LEN]; // buffers for the inbound stream

bool arrived[NR\_BUFS]; // inbound bit map

unsigned char nbuffered = 0; // initially no packets are buffered

int len;

int arg;

for (int i = 0; i < NR\_BUFS; ++i)

arrived[i] = false;

protocol\_init(argc, argv);

lprintf("Designed by Zhibo Zhang, build: " \_\_DATE\_\_" "\_\_TIME\_\_"\n");

disable\_network\_layer();

while (true) {

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

switch (event) {

case NETWORK\_LAYER\_READY: // accept, save, and transmit a new frame

++nbuffered; // expand the window

get\_packet(out\_buf[next\_frame\_to\_send % NR\_BUFS]); // fetch new packet

send\_frame\_k(FRAME\_DATA, next\_frame\_to\_send); // transmit the frame

next\_frame\_to\_send = inc(next\_frame\_to\_send); // advance upper window edge

break;

case PHYSICAL\_LAYER\_READY:

phl\_ready = true;

break;

case FRAME\_RECEIVED: // a data or control fraem has arrived

len = recv\_frame((unsigned char\*)& f, sizeof f); // fetch incoming frame from physical layer

if (len < 5 || crc32((unsigned char\*)& f, len)) {

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

if (no\_nak)

send\_frame\_k(FRAME\_NAK, 0); // damaged frame

break;

}

switch (f.kind) {

case FRAME\_DATA: // an undamaged frame has arrived

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

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

send\_frame\_k(FRAME\_NAK, 0);

else

start\_ack\_timer(ACK\_TIMER);

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

// frames may be accepted in any order

arrived[f.seq % NR\_BUFS] = true; // mark out\_buf as full

memcpy(in\_buf[f.seq % NR\_BUFS], f.data, len - 7); // insert data into out\_buf

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

// pass frames and advance window

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

no\_nak = true;

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

frame\_expected = inc(frame\_expected); // advance lower edge of receiver's window

too\_far = inc(too\_far); // advance upper edge of receiver's window

start\_ack\_timer(ACK\_TIMER); // to see if a separate ack is needed

}

}

break;

case FRAME\_ACK:

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

break;

case FRAME\_NAK:

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

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

send\_frame\_k(FRAME\_DATA, inc(f.ack));

break;

}

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

--nbuffered; // handle piggybacked ack

stop\_timer(ack\_expected % NR\_BUFS); // frame arrived intact

ack\_expected = inc(ack\_expected); // advance lower edge of sender's window

}

break;

case DATA\_TIMEOUT:

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

send\_frame\_k(FRAME\_DATA, ack\_expected); // time out

break;

case ACK\_TIMEOUT:

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

send\_frame\_k(FRAME\_ACK, 0); // ack timer expired, send ack

break;

}

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

enable\_network\_layer();

else

disable\_network\_layer();

}

}