北京邮电大学计算机网络

2023 年 5 月 16 日

数据链路层实验

目录

1 实验内容

1.1 实验内容

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。

1.2 实验目的

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。

1.3 实验要求

实验题目给出了物理层信道模型和分组层数据的大小，链路层协议的设计有很大的自由度。由组内同学共同讨论完成，包括帧控制字段的设计，滑动窗口的过程控制。从易到难，可选的协议类型为“不搭载 ACK 的 Go-Back-N 协议“，“使用搭载 ACK 技术的 Go-Back-N 协议”，“选择重传协议”，要求必须是全双工通信协议。本次实验中，选取了后两种协议进行实现。

2 实验环境

本实验提供的资料包可以在 Windows 以及 Linux 上运行, 本实验报告中选择的环境为运行于 Windows 10 搭载的 WSL 子系统上的 Arch Linux, 使用的 gcc 版本为 13.1.1, clang 版本为 15.0.7.

3 软件设计

3.1 数据结构

3.1.1 Go Back N 协议

```
#define DATA_TIMER 2000 // 重传计时器
#define ACK_TIMER 1100  // ACK计时器
#define MAX_SEQ 7       // 帧序号最大值
#define inc(k)                                                          \
  if (k < MAX_SEQ)                                                      \
    k++;                                                                \
  else                                                                  \
    k = 0;

typedef unsigned char frame_kind;      // 帧类型
typedef unsigned char seq_nr;          // 帧序号
typedef unsigned char packet[PKT_LEN]; // 数据包

typedef struct FRAME {  // 帧格式
  frame_kind kind;      // 帧类型
  seq_nr ack;           // ACK序号
  seq_nr seq;           // 帧序号
  packet info;          // 帧数据
  unsigned int padding; // CRC填充
} frame;

static int phl_ready = 0;        // 物理层状态
static unsigned char no_nak = 1; // ACK状态

static seq_nr next_frame_to_send;  // 发送窗口下一个发送
static seq_nr ack_expected;        // 发送窗口希望的ACK
static seq_nr frame_expected;      // 接收窗口希望的帧序号
static frame r;                    // 接受的帧
static packet buffer[MAX_SEQ + 1]; // 数据缓存
static seq_nr nbuffered;           // 已经缓存的帧数目
```

3.1.2 选择重传协议

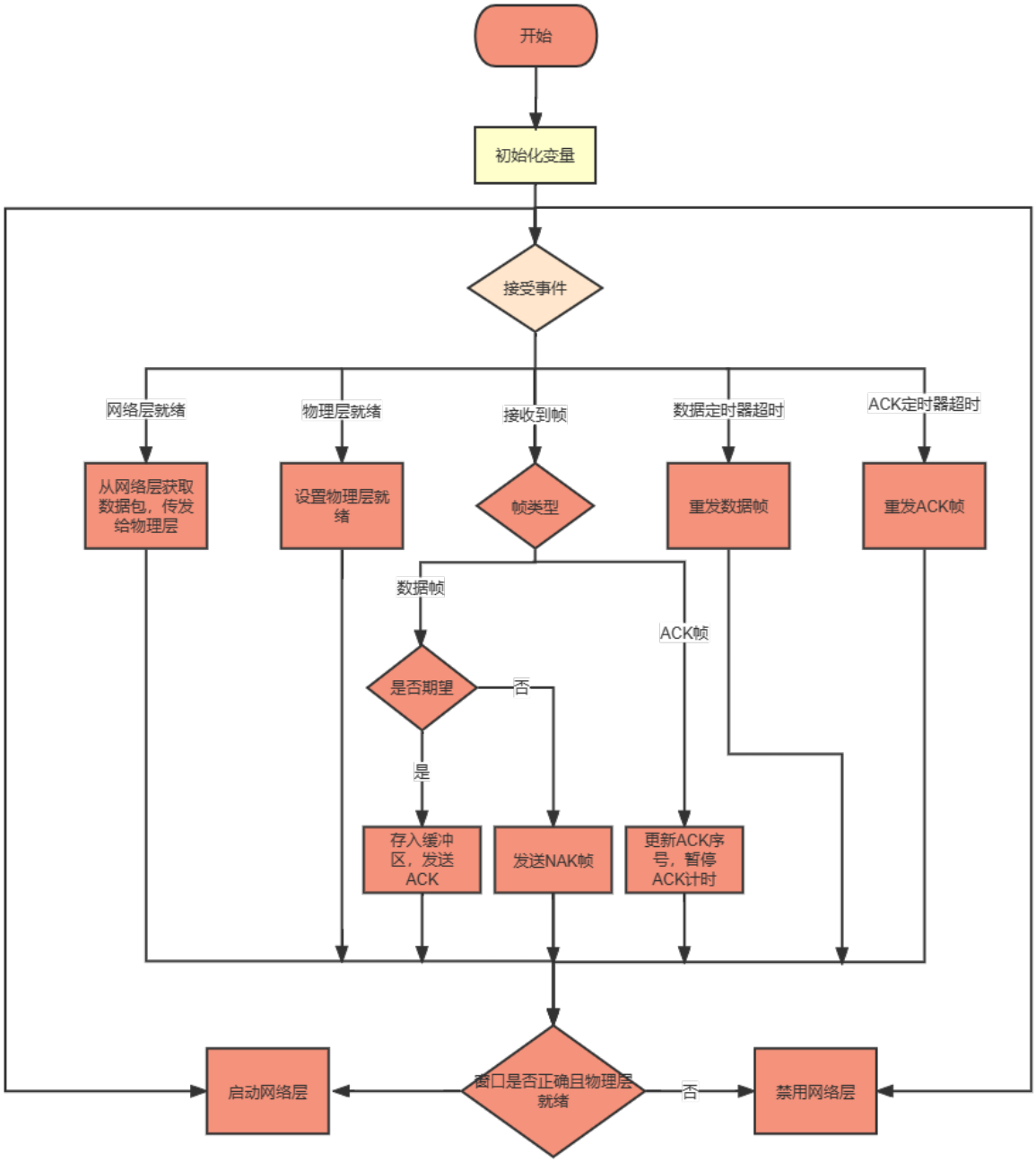
```
#define DATA_TIMER 6000             // 重传计时器
#define ACK_TIMER 5100              // ACK计时器
#define MAX_SEQ 43                  // 帧序号最大值
#define NR_BUFS ((MAX_SEQ + 1) / 2) // 窗口大小
#define inc(k)                                                          \
  if (k < MAX_SEQ)                                                      \
    k++;                                                                \
  else                                                                  \
    k = 0;

typedef unsigned char frame_kind;      // 帧类型
typedef unsigned char seq_nr;          // 帧序号
typedef unsigned char packet[PKT_LEN]; // 数据包

typedef struct FRAME {  // 帧格式
  frame_kind kind;      // 帧类型
  seq_nr ack;           // ACK序号
  seq_nr seq;           // 帧序号
  packet info;          // 帧数据
  unsigned int padding; // CRC填充
} frame;

static int phl_ready = 0;        // 物理层状态
```

```
24  static unsigned char no_nak = 1; // ACK状态
25
26  static seq_nr next_frame_to_send; // 发送窗口下一个发送
27  static seq_nr ack_expected;       // 发送窗口希望的ACK
28  static seq_nr frame_expected;     // 接收窗口希望的帧序号
29  static seq_nr too_far;            // 接收窗口右端点的下一个
30  static frame r;                   // 接收的帧
31  static packet in_buf[NR_BUFS];    // 接收窗口缓冲
32  static packet out_buf[NR_BUFS];   // 发送窗口缓冲
33  static bool arrived[NR_BUFS];     // 记录完成的帧
34  static seq_nr nbuffered;          // 已经缓存的帧数目
```

3.2 模块结构

```
1   /*
2    * 判断某个序号是否在窗口内
3    * a 左下标
4    * b 判断的值
5    * c 右下标的下一个
6    */
7   static unsigned char between(seq_nr a, seq_nr b, seq_nr c) {
8     return ((a <= b && b < c) || (c < a && b < c) || (c < a && a <= b));
9   }
10
11  /*
12   * 发送帧添加校验位给物理层
13   */
14  static void put_frame(unsigned char *frame, int len) {
15    *(unsigned int *)(frame + len) = crc32(frame, len);
16    send_frame(frame, len + 4);
17    phl_ready = 0;
18  }
19
20  /*
21   * 填充内容发送给物理层
22   */
23
24  static void send_data_frame(frame_kind fk, seq_nr frame_nr,
25                              seq_nr frame_expected, packet buffer[]) {
26    frame s;
27    s.kind = fk;
28    s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
29    stop_ack_timer();
30
31    if (fk == FRAME_DATA) {
32      s.seq = frame_nr;
33      memcpy(s.info, buffer[frame_nr % NR_BUFS], PKT_LEN);
34      dbg_frame("Send DATA %d %d, ID %d\n", s.seq, s.ack, *(short *)s.info);
35      put_frame((unsigned char *)&s, 3 + PKT_LEN);
36      start_timer(frame_nr, DATA_TIMER);
37    } else if (fk == FRAME_ACK) {
38      dbg_frame("Send ACK  %d\n", s.ack);
39      put_frame((unsigned char *)&s, 2);
40    } else if (fk == FRAME_NAK) {
41      dbg_frame("Send NAK  %d\n", (s.ack + 1) % (MAX_SEQ + 1));
42      no_nak = 0;
43      put_frame((unsigned char *)&s, 2);
44    }
45  }
```

3.3 算法流程


开始
初始化变量
接受事件
网络层就绪
物理层就绪
接收到帧
数据定时器超时
ACK定时器超时
从网络层获取数据包，传发给物理层
设置物理层就绪
帧类型
重发数据帧
重发ACK帧
数据帧
ACK帧
是否期望
否
是
存入缓冲区，发送ACK
发送NAK帧
更新ACK序号，暂停ACK计时
窗口是否正确且物理层就绪
否
启动网络层
禁用网络层

4 实验结果

4.1 简单描述

我所实现的协议软件实现了有误码信道中无差错传输功能

4.2 健壮性

在文档中给出的所有性能测试命令下, 程序均可正常运行超过 20 分钟, 且性能接近参考数据, 这证明了程序的正确性与健壮性, 实现了有误码信道环境中无差错传输功能.

4.3 协议参数的选取

考虑滑动窗口的大小, 重传定时器的时限, ACK 搭载定时器的时限. 根据信道特性数据, 分组层分组的大小, 以及的滑动窗口机制, 给出定量分析, 详细列举出选择这些参数值的具体原因.

信道带宽为 8000 bps, 单程传播时延为 270ms, 使用稍带确认

$$u = \frac{W}{2(1+\alpha)}$$

$$\alpha = \frac{t_{prop}}{t_{tran}}$$

$$p_{tran} = \frac{packet\_len}{transmission_rate}$$

解得, $u = W/4.1 \geqslant 100\% \Rightarrow W \geqslant 5$.

重传定时器设置为两倍传播时延加一个 ACK 定时器时延比较合适, ACK 定时器设置为 2000ms, 则重传定时器为 $2700 * 2 + 2000 = 7400$ms.

4.4 理论分析

4.4.1 无误码情况下最大信道利用率

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

$$\frac{256}{263} \times 100\% \approx 97.3\%$$

4.4.2 有误码情况下最大信道利用率

这里假设重传操作及时, 重传的数据帧的回馈, ACK 帧可以 100% 正确传输.

当误码率为 10^{-5}, 即发送 10^5 个比特, 出现一个错误, 则平均发送 $\frac{100\ 000}{263 \times 8} = 47.53$ 个帧会出现一个错误. 先假设重传的帧必然正确, 则发送 47.5 帧需要发送 $47.5 + 1 + 1 = 49.5$ 帧 (NAK 帧和重发的帧).

此时信道利用率为:

$$47.5/49.5 * 97.3\% = 95.3\%$$

当误码率为 10^{-4}, 即发送 10^4 个比特, 出现一个错误, 则信号利用率为:

$$\frac{5}{6} \times 97.3\% = 81.0\%$$

4.5 结果分析

序号	命令选项	说明	运行时间(秒)	GoBackN 算法线路利用率(%)		Selective 算法线路利用率(%)	
				A	B	A	B
1	-utopia	无误码信道数据传输	1200	53.71	95.56	54.19	96.97
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	46.75	81.88	52.25	94.48
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	96.97	96.97	96.97	96.97
4	-flood	站点 A/B 的分组层都洪水式产生分组	1200	85.11	84.12	94.92	94.25
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}	1200	31.75	32.51	56.58	59.68

数据结果记录表

总体来说, 结果和理论值 (参考数据) 很接近, 效果很好.

4.6 存在的问题

程序虽然经过了所有测试方案, 但在低误码率的情况下理论值与实际数据均有改进空间. 另一个是参数可能不是最优情况, 只是得到了一个较好的结果, 并没有继续向后测试.

5 研究和探索的问题

5.1 CRC 校验能力

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。而这一概率为 5.160×10^{-93} 可以认为这一事件为不可能事件, 客户每天实际可以发送的帧数目为 $94.6\% * 8000/256/8 * 60 * 60 * 24/2 = 159638$ 帧. 所以发生这一事件至少需要 $1.937 \times 10^{92}/159638/365 = 5.3091 \times 10^{91}$ 年.

5.2 软件测试方面的问题

不同的测试方案可以检测软件在不同状况下的稳定性与性能, 以模拟真实环境.

无误码信道数据传输可以用于验证程序基础功能的正确性, 这样可以先摆脱校验模块进行测试, 提高开发效率.

平缓方式和洪水式产生分组则是模拟真实网络环境中的空闲期与高峰期.

高误码率情况则可检测程序健壮性, 就网络而言, 一般来说人们愿意牺牲一定的性能来换来更高的稳定性.

此外, 认为测试还应该覆盖信道完全空闲, 无数据传输时的情况, 以检测程序是否表现正常.

5.3 对等协议实体之间的流量控制

鉴于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.

6 实验总结和心得体会

6.1 实验总结

本次实验在课下学习理解实验书用了大约 3-4 小时, 实际上机撰写代码并调试的时间为 6 小时.

由于使用的是自己平时一直在用的编程环境, 因此在编程环境上没有出现问题, 而编程过程中没有发现 C 语言方面的问题.

协议参数用了很长时间才测试出较为理想的结果, 而通过本次实验的代码编写对数据链路层各功能的实现有了形象的概念.

6.2 心得体会

SR 协议和 GBN 协议是网络传输中常用的协议, 它们各自具有不同的优缺点, 需要根据具体的网络环境选择合适的协议.

在实现这两种协议时, 需要深入了解其原理, 并使用合适的编程语言进行实现.

在测试过程中, 需要考虑不同的网络环境, 包括丢包率、带宽等因素, 以便更加准确地比较两种协议的性能.

在进行性能比较时, 需要考虑各种因素, 包括丢包率、延迟、吞吐量等, 以便更加全面地评估两种协议的优缺点.

在这次实验中, 通过实现和测试来深入了解这些协议的优缺点, 这对我们进一步学习和研究网络传输具有重要的意义.

本组实验只有一个人参加, 虽然工作量较大, 但是通过对整个实验过程的亲身经历, 对实验有了更加全面而细致的了解, 也深切理解到真正的协议制定及参数调整其不易之处.

7 源程序文件

见附件.