北京邮电大学《计算机网络》课程实验报告

<table>
<tr><td>实验
名称</td><td colspan="2">数据链路层滑动窗口
协议的设计与实现</td><td>学
院</td><td>计算机</td><td>指导教师</td><td>张雪松</td></tr>
<tr><td>班 级</td><td>班内序号</td><td colspan="2">学 号</td><td>学生姓名</td><td colspan="2">成绩</td></tr>
<tr><td>2023211211</td><td>4</td><td colspan="2">2023211196</td><td>马一民</td><td colspan="2"></td></tr>
<tr><td>2023211311</td><td>10</td><td colspan="2">2023211202</td><td>张鑫溢</td><td colspan="2"></td></tr>
<tr><td>2023211311</td><td>17</td><td colspan="2">2023211209</td><td>肖璨</td><td colspan="2"></td></tr>
<tr><td>实
验
内
容</td><td colspan="6">本次实验选用的滑动窗口协议为选择重传/回退 N 步，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。
本次实验，本小组分析了信道传输效率，优化了算法设计，通过调整多种参数，提高了信道利用率、带宽等多个方面。
本次实验中，马一民同学负责算法优化、程序调试、参数分析以及进一步分析问题优化改进问题等工作，张鑫溢同学负责数据结构的设计、程序编码、主函数框架设计等工作，肖璨同学负责协议设计、程序编码、数据测试、参数优化等工作，小组共同总结并完成了实验报告。</td></tr>
<tr><td>学生实验报告</td><td colspan="6">（详见“实验报告和源程序”册）</td></tr>
<tr><td>课程
设计
成绩
评定</td><td colspan="6">评语：

成绩：
指导教师签名：
年 月 日</td></tr>
</table>

注：评语要体现每个学生的工作情况，可以加页。

一、 实验内容和实验环境描述

1 实验内容及目的

本次实验要求学习并利用数据链路层原理，自行变成实现一个滑动窗口协议。选用的滑动窗口协议为选择重传协议，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。

实验目的在于深入理解并熟练运用数据链路层原理，通过实际编程实现滑动窗口协议，掌握选择重传/回退 N 步协议在有噪音信道环境下的运行机制。同时，探究 NAK 通知机制对协议性能的影响，对比使用与不使用该机制时数据传输的可靠性、传输效率等指标。

在实际应用中，可靠的双工通信是网络通信的基石，通过本实验优化滑动窗口协议，可有效提高数据在复杂信道环境下的传输质量，为卫星通信、无线通信等领域的数据传输提供技术参考与保障，对推动通信技术的发展具有重要的现实意义。

通过此次实验，有助于加深对数据链路层协议的认知，提升理论与实践结合的能力，切身学习、深入体会协议设计与调试，增强了对实际的分层协议结构的认知。

2 实验环境

系统环境为 Windows 11，使用 Visual Studio 2017+ 集成开发环境，使用 C 语言进行开发。

3 实验原理

3.1 数据链路层

在计算机网络体系中，分层结构是实现网络通信的核心架构，它将复杂的网络功能划分为多个层次，每个层次专注于特定的任务，通过层与层之间的协作完成整个网络通信过程。

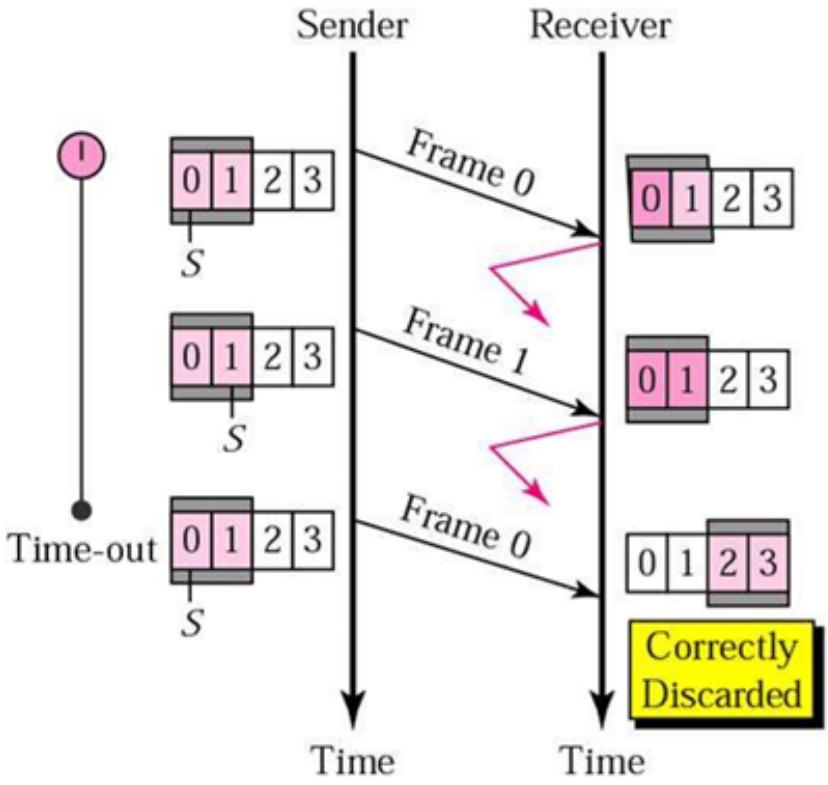
本次实验旨在深入学习理解数据链路层。数据链路层首先从网络层获取数据包，紧接着将数据包封装成帧，帧的结构一般包含帧头、数据包以及帧尾。帧头部分携带了如源地址、目的地址、控制信息等关键内容，用于指导数据的传输与处理；帧尾通常设置校验和字段，例如循环冗余校验（CRC）码，接收方通过重新计算校验和来判断帧在传输过程中是否出现错误。完成封装后，数据帧借助物理层进行传输，接收方的数据链路层接收到帧后，会进行解析，提取出数据包，并将其传递给接收方的网络层。

3.2 选择重传 / 回退 N 步协议原理

GBN 协议：发送方可以连续发送多个数据帧，接收方只按序接收数据帧。当接收方检测到某个帧出错或丢失时，丢弃该帧及其后续所有帧，并通过 ACK 确认序号告知发送方。发送方一旦超时或收到 ACK 序号小于当前已发送帧序号，就从出错帧开始，重新发送窗口内所有未被确认的帧。这种方式简单，但可能会导致大量不必要的重传，尤其是在误码率较高的信道中。

SR 协议：同样允许发送方连续发送多个数据帧，但接收方可以对每个正确接收的帧进行单独确认。当检测到某帧出错时，接收方暂存出错帧之后的正确帧，同时向发送方发送 NAK（若使用该

机制）或不发送确认，告知发送方重传出错帧。发送方仅重传出错的帧，而非像 GBN 那样重传多个帧，从而减少了信道资源的浪费，在高误码率信道中能显著提高传输效率。



3.3 帧结构

KIND	ACK	SEQ	DATA	CRC

KIND 是帧类型，ACK 为确认帧，SEQ 为帧序列号，DATA 为数据字段，CRC 为校验码。

3.4 CRC 校验

CRC，循环冗余校验是数据链路层广泛使用的一种差错检测技术，通过多项式除法生成校验码，用于检测数据传输过程中是否

发生比特错误。

CRC 校验基于模 2 运算(即异或运算,不考虑进位和借位),将待传输的数据视为一个二进制多项式，然后用一个生成多项式(固定的二进制序列)对其进行模 2 除法,得到的余数即为 CRC 校验码。接收方使用相同的生成多项式对收到的数据进行计算，若计算结果与接收到的 CRC 校验码一致,则认为数据传输正确。

二、 软件设计

1 数据结构设计

1.1 结构体

本程序中定义了一个核心结构体 FRAME 用于封装网络传输的基本单元——帧。该结构体包含几个关键成员：

kind 标识帧的类型，包括数据帧（FRAME_DATA）、确认帧（FRAME_ACK）或否定确认帧（FRAME_NAK）;

ack 作为确认号，在数据帧和 ACK 帧中用于捎带确认，表示发送方确认接收到的最后一个帧的序号;

seq 是帧的序列号，主要用于数据帧的标识，表示当前帧的序号;

data[PKT_LEN] 是一个字符数组，存储实际传输的数据，其长度由宏 PKT_LEN 定义;

padding 是填充字节，用于满足数据对齐要求或为 CRC 校验等信息预留空间。

```
27  struct FRAME
28  {
29      unsigned char kind;
30      seq_nr ack;
31      seq_nr seq;
32      unsigned char data[PKT_LEN];
33      unsigned int padding;
34  };
```

1.2 全局变量以及主函数变量

本程序定义了以下全局变量:

```
43  static seq_nr next_frame_to_send = 0; // 发送方下一个要发送的帧序号
44  static seq_nr ack_expected = 0;       // 发送方下一个要确认的帧序号
45  static seq_nr frame_expected = 0;     // 接收方下一个要接收的帧序号
46  static seq_nr too_far;                // 接收方下一个要接收的帧序号 (窗口上界)
47
48  static unsigned char out_buf[NR_BUFS][PKT_LEN]; // 发送方缓冲区
49  static unsigned char in_buf[NR_BUFS][PKT_LEN];  // 接收方缓冲区
50  static bool arrived[NR_BUFS];                   // 接收方缓冲区位图 (标记哪些槽已填充)
51
52  static int nbuffered = 0;  // 发送方缓冲区中已存放的帧数
53  static int phl_ready = 1;  // 物理层是否准备好接收数据
54  static bool no_nak = true; // 是否禁止连续发送 NAK
```

这些全局变量用于实现滑动窗口协议中发送方和接受方的运行状态。

发送方状态变量包括：next_frame_to_send（seq_nr）记录发送方下一个新数据帧的序号；ack_expected（seq_nr）代表发送方期望收到的最小确认号，标志着发送滑动窗口的下沿；以及nbuffered（int）统计发送方当前已缓存（已发送但未确认）的帧数；

接收方状态变量包括：frame_expected（seq_nr）指示接收方当前期望接收的数据帧序号，作为接收滑动窗口的下沿；too_far（seq_nr）定义了接收滑动窗口的上界（不含该值），接收方仅接

受序号在此窗口内的帧；以及 arrived[NR_BUFS] (bool 数组)，这是一个位图，标记接收缓冲区中哪些序号的帧已成功到达；

数据缓冲区方面：out_buf[NR_BUFS][PKT_LEN] (unsigned char 二维数组) 是发送方的循环缓冲区，用于存储从网络层获取但尚未收到确认的数据包； in_buf[NR_BUFS][PKT_LEN] (unsigned char 二维数组) 则是接收方的循环缓冲区，用于存储已正确接收但可能失序的数据帧，等待按序提交。

以及控制标志包括：phl_ready (int) 标识物理层状态，1 表示物理层空闲可接受数据，0 表示繁忙；no_nak (bool) 作为 NAK 连续发送标志。

另外在主函数中，还涉及以下变量，event (int) 用于存储由 wait_for_event 函数返回的事件类型，如网络层就绪、物理层就绪、帧到达或超时等；arg (seq_nr) 则携带与这些事件相关的特定参数，例如在数据帧超时事件中，它会保存超时的帧序号；而 len (int) 则记录了实际接收到的帧的长度，这个长度包括了帧头和数据部分，但不包含物理层可能添加的 CRC 校验码。

2 模块结构

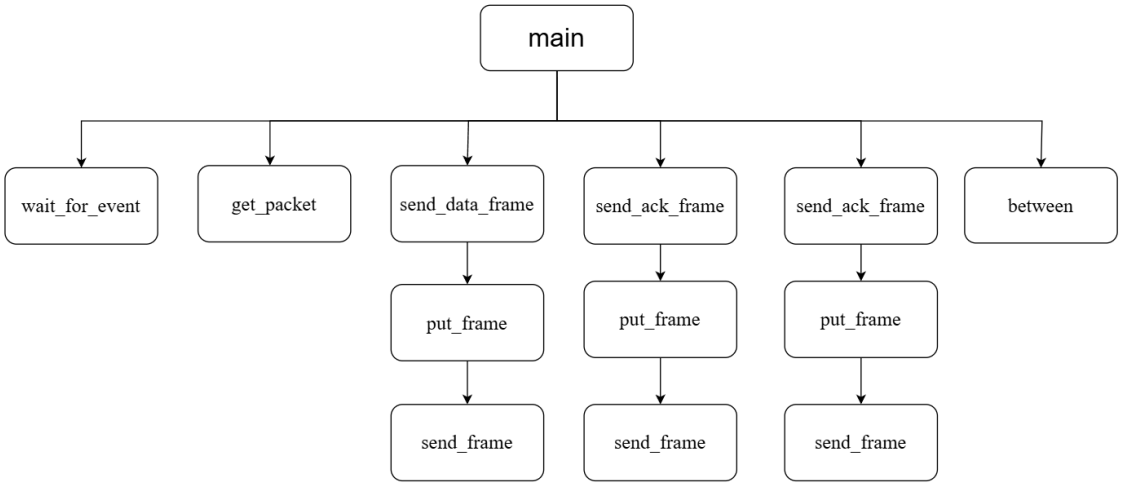
2.1 主要子程序

子程序名称	功能	参数	返回值
inc	将序号 k 加 1。如果 k 达到 MAX_SEQ（值为 31），则将其重置为 0，实现序号的循环使用。	k（seq_nr 类型），要递增的序号	无

between	判断序号 b 是否位于由 a（窗口下界，包含）和 c（窗口上界，不包含）定义的滑动窗口区间内。此函数正确处理了序号循环回绕的情况。	a（seq_nr 类型），窗口的起始序号 b（seq_nr 类型），需要判断是否在窗口内的目标序号 c（seq_nr 类型），窗口的结束序号	如果 b 在窗口 [a, c) 内，则返回 true；否则返回 false
show_received_buffer()	在控制台打印接收方缓冲区 arrived 的当前状态，显示哪些序号的帧已经接收并存入缓冲区。	无	无
put_frame	将构造好的帧发送到物理层。在发送前，它会计算帧数据（不包括 CRC 字段本身）的 CRC32 校验和，并将此校验和附加到帧的末尾。然后调用 send_frame（物理层接口函数）发送整个帧（数据 + CRC）。发送后，将 phl_ready 置为 0，表示物理层暂时不可用。	frame（unsigned char* 类型），指向包含帧头部和数据的内存区域的指针 len（int 类型），帧数据的长度	无
send_data_frame	构建并发送一个数据帧。	frame_nr（seq_nr 类型），要发送的数据帧的序号	无

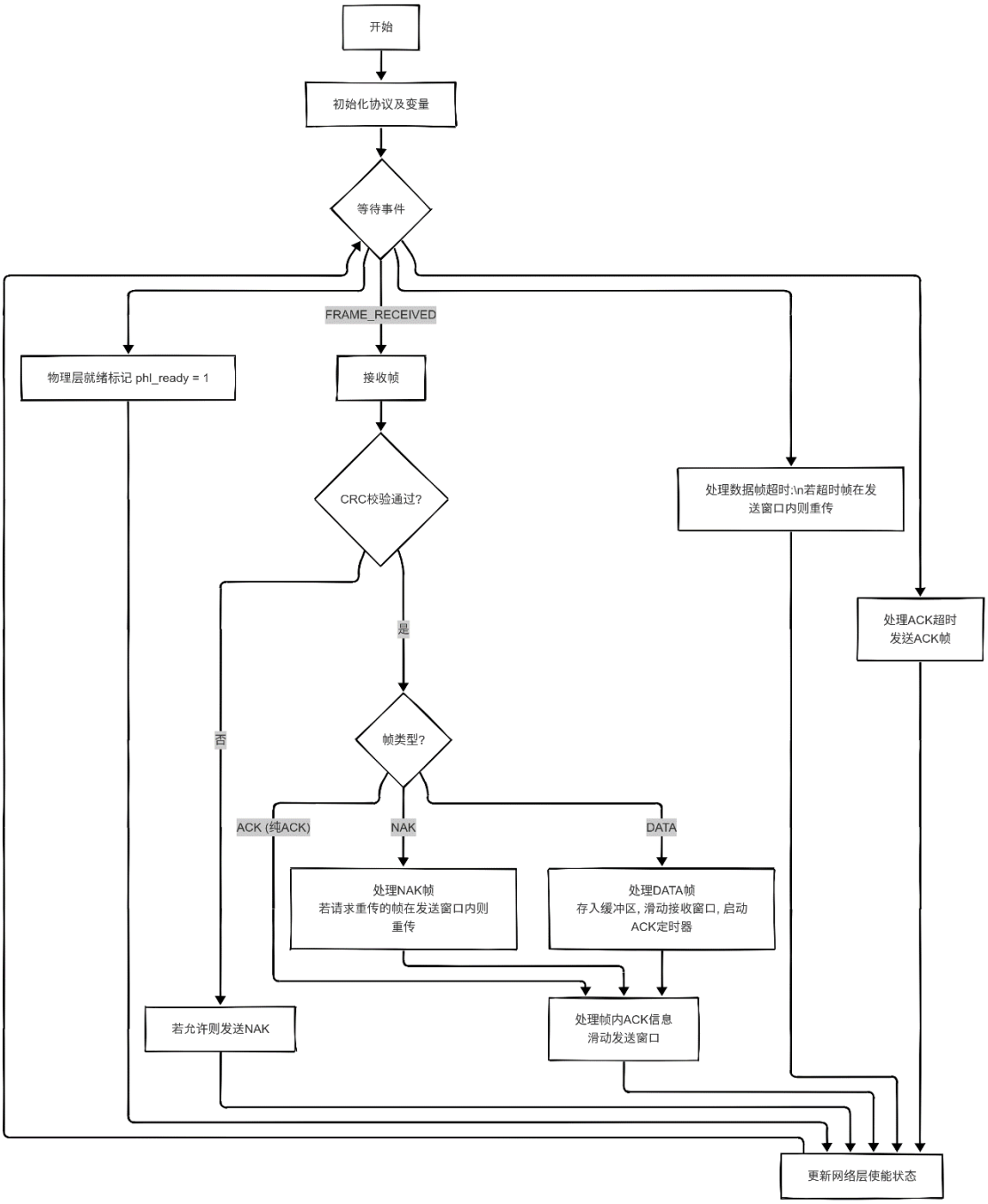
send_ack_frame	构建并发送一个独立的 ACK 帧。当接收方收到数据但暂时没有数据要回复时，会使用此函数发送确认。	无	无
send_nak_frame	构建并发送一个 NAK 帧。当接收方检测到帧丢失或损坏时（例如，收到 CRC 错误的帧，或收到的帧序号不连续），会发送 NAK 请求重传。	无	无
main	程序的主控制函数，实现了滑动窗口协议的核心逻辑。	argc（int 类型），命令行参数的个数 argv（char** 类型），指向命令行参数字符串数组的指针	

2.2 主要子程序调用关系图



3 算法流程

3.1 算法流程图



3.2 算法流程描述

主函数中首先进行初始化，包括全局变量、发送和接收窗口及

缓冲区的初始化，并将网络层就绪；接着进入循环，通过 wait_for_event() 等待事件，根据不同事件类型执行相应操作，如网络层准备好数据且发送窗口未满时获取数据包并发送数据帧，接收到帧时检查 CRC 校验校验通过后，依据帧的具体类型（数据帧或 ACK 帧）进行不同处理。若是数据帧，将其存入缓冲区，滑动接收窗口并启动 ACK 定时器；若是 ACK 帧，则根据帧内 ACK 信息滑动发送窗口、释放缓冲区。当数据帧超时事件，会将发送窗口内的所有帧进行重新传输，确保数据不丢失。而当 ACK 超时，会立即发送 ACK 帧，维持通信的可靠性。如此循环，直至所有数据传输完成。

三、 实验和结果分析

1 协议软件在有误码信道环境中无差错传输功能

本组实现的选择重传协议能够有效处理数据传输中的错误情况。任何 CRC 检测出错的帧将会拒绝接受，发送 NAK 立即要求重传或等待超时重传，丢包则需要等待超时重传，最后收到的包将会按顺序无差错上交网络层。

经过本组测试，该协议能够长时间工作而不出错，进一步验证了无差错传输功能的实现。

2 程序的健壮性

经过测试，该协议能够可靠地保持长时间运行。

3 理论分析

下面分析选择重传协议在无差错信道环境和有差错信道环境下能够获得的最大信道利用率。

假设超时重传的 ack 可以 100%正确传送，不考虑 ack 和 nak 占用的带宽。理论选择重传可以获得的的效率：

一次发送成功的概率：

$$P(\textit{一次发送成功}) = p_1 = (1-p)^{263*8} = (1-p)^{2104}$$

假设一帧要发送的次数为随机变量 X，

$$X\sim \textit{几何分布}$$

$$P(X=k) = (1-p_1)^{k-1}p_1$$

$$E(X) = \frac{1}{p_1} = \frac{1}{(1-p)^{2104}}$$

无误码条件下：

$$\eta = \frac{256}{263} = 97.34\%$$

误码率10^{-5}情况下：

$$E(X) = 1.02126, \qquad \eta = \frac{256}{263 * E(X)} = 95.31\%$$

误码率10^{-4}情况下：

$$E(X) = 1.23418, \qquad \eta = \frac{256}{263 * E(X)} = 78.87\%$$

4 各版本协议参数选取、实验结果、存在问题分析

本组实现了选择重传协议，且实现了三个版本，后两个版本与课程介绍的的选择重传协议不同，参数选取和结果也不同，故分版本分析。

4.1 教材中提供的选择重传协议的实现方法

4.1.1协议参数的选取分析

首先分析一下两种错误率意味着什么，这将会影响协议参数

的选取。

考虑较低误码率10^{-5}下出错的可能性：

发送一帧出错的概率是

$$p = 1 - (1 - 10^{-5})^{263*8} = 2.082\%$$

连续两帧出错的概率是

$$p_2 = p * p = 0.0433\%$$

类似的，计算高误码率的情况，得到如下表格：

		较低误码率10^{-5}	较高误码率10^{-4}
发送一帧出错	出错概率	2.082%	18.975%
	出现频率	约 12 秒一次	约 1.5 秒一次
连续出错两次	出错概率	0.0433%	3.601%
	出现频率	约 1247 秒(20 分钟)一次	约 14 秒一次

根据上述不同错误率的情况，下面逐个分析各个参数的选取

窗口大小：

根据所给的信道模型为 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}，网络层分组长度固定为 256 字节，均使用捎带应答（对于网络层新包较少，不适用捎带应答的情况，通过 ack 超时时长的调节可以做到和使用捎带应答的反应时间几乎一致），根据图片内容，可以做如下计算：

① 无误码的情况下：

$$\alpha = \frac{传播时延}{发送时延} = \frac{270}{(256+7)*\frac{8}{8000}*1000} = \frac{270}{263}$$

$$\frac{W}{2+2\alpha} \geq 1$$

$$解得 W \geq 4.05$$

② 倘若发生一次出错，此时由于 nak 的存在，发送方可以迅速响应，重发出错帧，此时需要

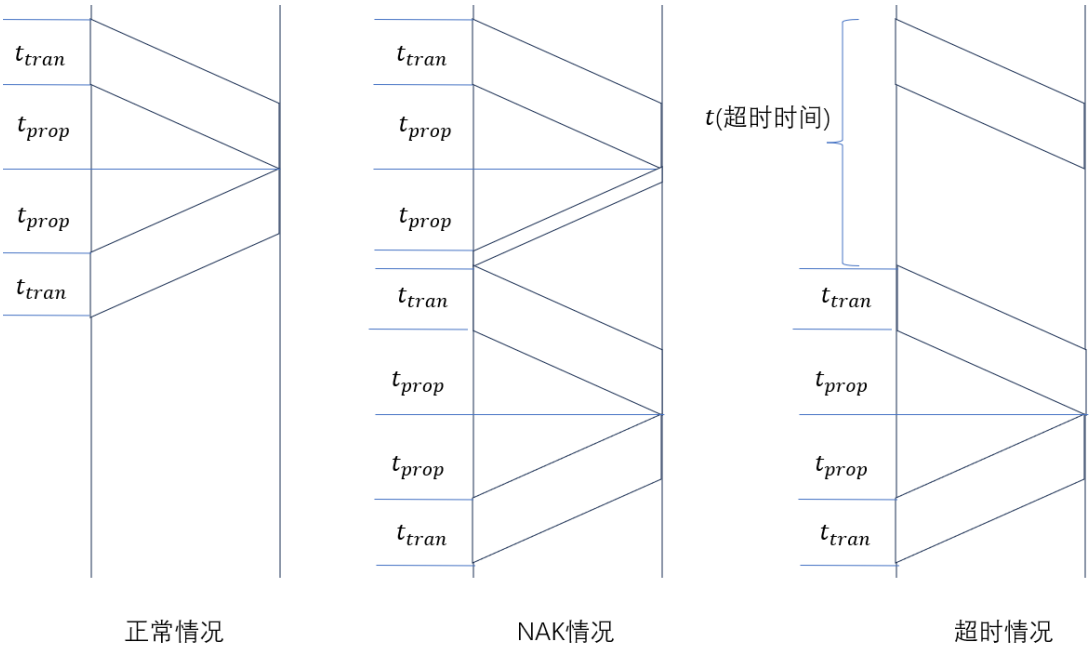
$$W \geq 2 + 4\alpha = 6.11$$

才能完全避免影响。

③ 假设误码率较高，出现了连续的错误，不得不使用超时重传。如果有一个数据需要等待超时重传，超时时间为 t，此时需要

$$W \geq 2\alpha + 1 + \frac{t}{263}$$

发送一帧所致的最长空闲时间就是超时重传所带来的时间，倘若 t 过大，则 W 会需要很大才能尽可能提高效率。



由此得到 W 的取值范围，为了确定 W 的取值，还需要分析一下

数据超时时间 t。

需要注意的是，W 大小的调节仅仅是为了使信道尽可能占满，仅仅能够提高由于信道空闲而导致的效率低下。由于数据出错率高导致的效率降低是不可能通过 W 的调节得到补偿的。

也就是说，窗口的大小在不占用过多资源的情况下，可以尽可能增大，以便将信道上充满发送的信息，同时因为物理层存在流量控制，故不必担心使得物理层缓冲区溢出。

为什么发送窗口个数等于接收窗口个数？发送窗口和接收窗口一定要$\leq 2^n$，否则发送和接受的窗口中均会出现重复序号；如果 ack 一直丢包，发送方一直发送旧帧，为了防止接收窗口滚动过一个周期，把旧帧当作新帧接受，必须令发送窗口和接收窗口之和$\leq 2^n$；发送窗口小于接收窗口是无意义的；发送窗口大于接收窗口，一旦出现坏帧，接收窗口停止滑动，发送窗口发送的大于接收窗口接受能力的帧都将无法接受。所以发送窗口最好等于接收窗口。

4.1.2 数据超时时间

数据的超时几乎只在高误码率情况下才会出现，单次错误可以通过 nak 弥补，连续的或短时间内的错误只得等待数据超时。

单次错误时，也就是低误码状态下：

超时时间首先不可过短，应该大于一个使用 nak 的包从发送到接收的时延，否则会出现大量的重发帧占用带宽。

考虑使用 nak 的一个包从发送到成功接收的时间，也就是 t

的最短时间。

$$263 * 3 + 270 * 4 + 6 = 1875ms$$

较多错误时，高误码状态下：

在 nak 发送之后，再次出错，无法发 nak，且捎带应答失效（因为只确认 nak 之前的帧），若是错误过多，此时无论 t 设置较大或较小，后来发送的数据总是会超时，并且容易连续大量帧出现超时，类似如下情况：

……

XXX.XXX 3 号帧超时，重传

XXX.XXX 4 号帧超时，重传

XXX.XXX 5 号帧超时，重传

XXX.XXX 6 号帧超时，重传

……

简单通过调节 t 的大小无法避免这种情况，所以 t 的设置只需要满足低误码状态下的最短时间即可。

通过上述分析，根据 t 和 W 之间的关系，可以令窗口为 16（窗口调节得更大没有好处，因为只要满足上述限定，带宽就可以基本占满，限制高误码率的效率原因不在于此，经过测试，高误码率下带宽占用率也可达到 99%），则$t \leq 3419ms$，超过该值则无法占满带宽，设置数据帧超时为 2000ms。

4.1.3 数据超时时间 t

该值则无法洪水模式下，认为两方总是有数据发送，此时由于存在物理层的流量控制和发送速率限制，使得一个 packet 发送到下一个 packet 发送需要一定时间，即是两帧发送的最短间隔时间，也是捎带应答的反应时间，如果 ack 超时时间短于这个时间，则会发送大量的 ack 占用带宽。如果 ack 较大，由于洪水模式下总是使用捎带应答，无影响。

非洪水模式下，数据发送较慢时，捎带应答较为少用，需要等待 ack 超时时间，此时应尽可能使得 ack 超时时间和正常的捎带应答反应时间相等。即稍大于 263ms，也就是发送一个 packet 需要等待物理层的时间：

$$t_1 = \frac{(256 + 7) * 8}{8000} = 263ms$$

由此可以得知 t_{ack}设置为稍大于 263ms 的值即可，本组设置为 270ms.

4.1.4 实验结果分析

实验结果如下

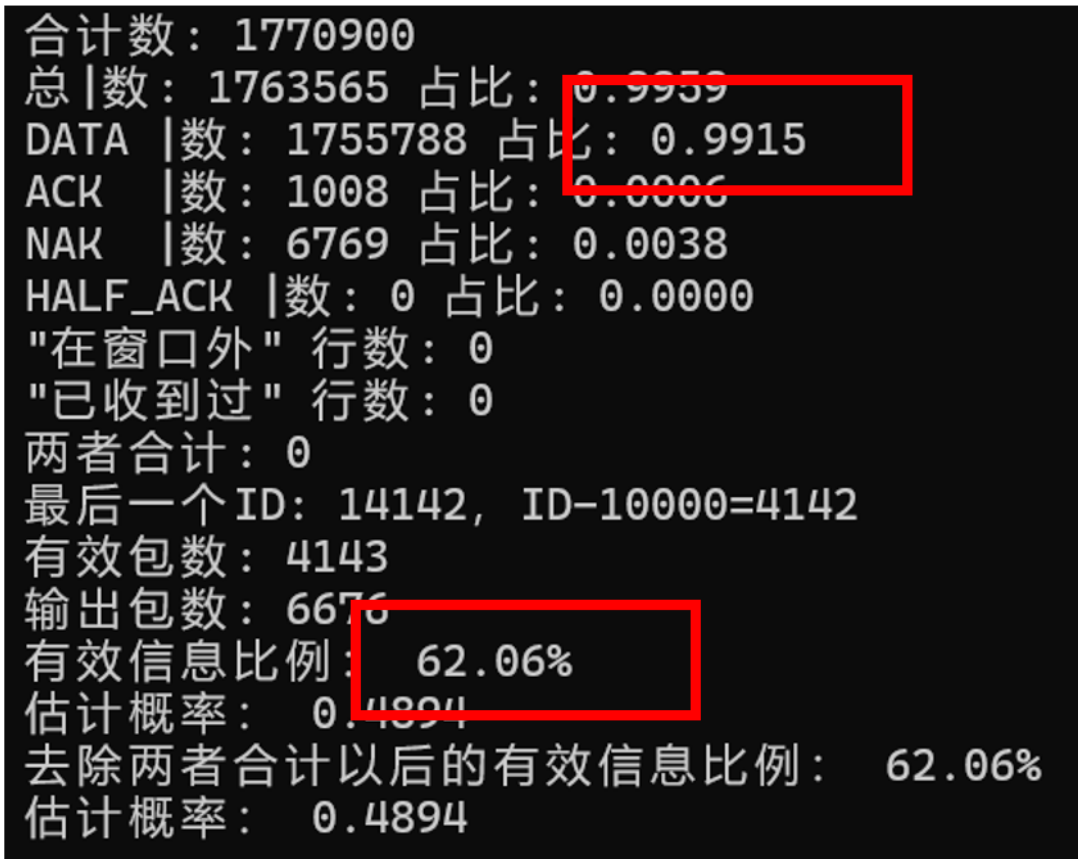
序号	命令选项	说明	运行时间(秒）	Selective 算法线路利用率(%)		存在的问题
				A	B	
1	--u	无误码信道数据传输	1800	53.54	96.96	
2	无	站点 A 分组层平缓方式发出数据，站点 B 周期性交替“发送 100 秒，停发 100 秒”	1800	51.46	95.17	
3	–f -u	无误码信道，站点 A 和站点 B 的分组层都洪水式产生分组	1400	96.96	96.96	
4	--f	站点 A/B 的分组层都洪水式产生分组	1800	95.23	95.18	
5	-f –b 1e-4	站点 A/B 的分组层都洪水式产生分组，线路误码率设为 10-4	1800	56.19	53.57	与理论值 78.87 相差较大

4.1.5 存在的问题

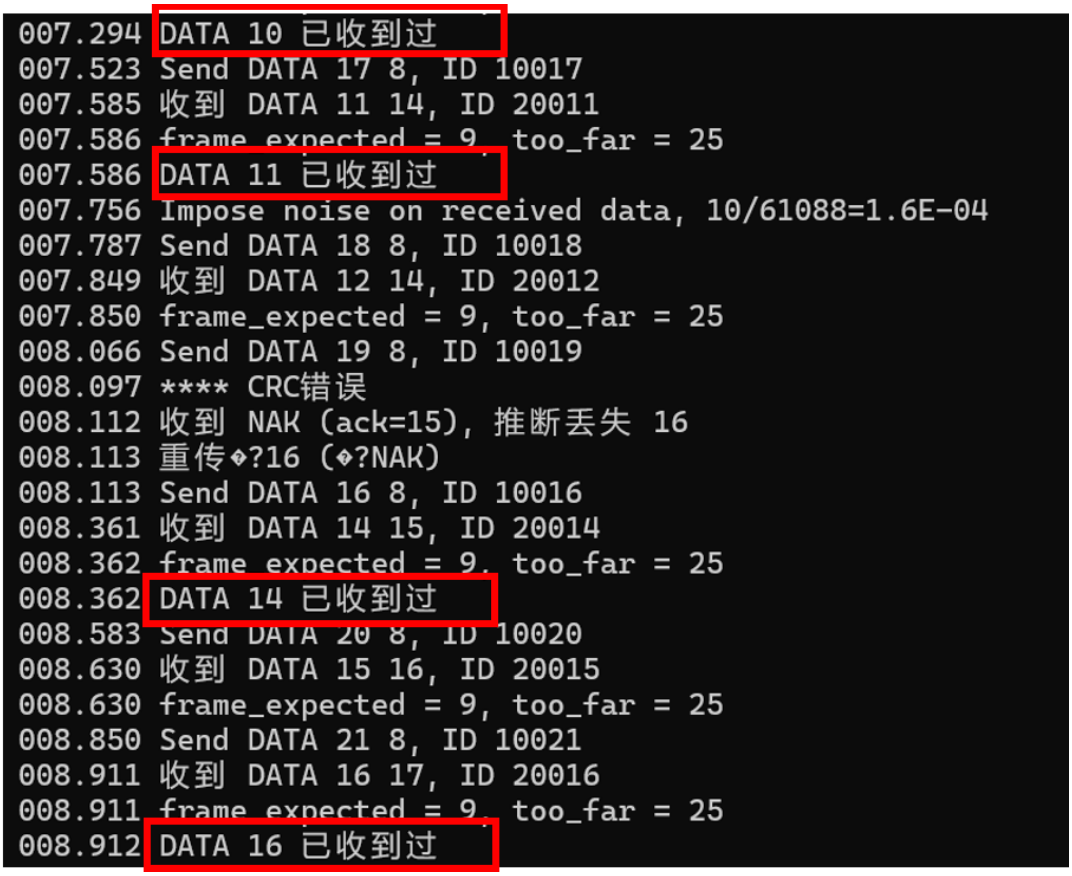
实验结果如下可见高误码率条件下与理论值相差很大。

分析原因如下：

本组利用自行设计的日志文件分析器，分析其中的带宽占用和信息有效比例。得到如下结果：



上述两个圈出的数据分别代表带宽占用和有效信息比（发送了但是收到过的或在窗口外的被认为是无效信息）。可见带宽占用率已经很高，而有效信息率很低。通过观察调试结果，可以看到出现下图所示的现象：



分析原因，是因为 NAK 一旦发送，此后所有的 ACK 和捎带应答传输的 ack 数据段都只指示 nak 前一个数据段，当前方出错帧较多，后续许多已经到达的数据段无法及时发送确认，使得发送方长时间收不到确认，误以为数据出错或丢失，超时后重新发送，然而这些数据已经被发送方接受，就会出现“已收到过”或者是“在窗口外”的情况。

对此，我们采用的解决方法是超时帧数量限定和 Half ACK，具体原理见版本二。

4.2 带有超时帧数量限定和 Half ACK 的选择重传

4.2.1 原理分析

具体的原理如下：

超时帧数量限定：允许有限数量的排序靠前的帧超时重传，剩余的帧超时后重新添加一个 ACK 超时的时间继续计数，这样就不会发送过多的已收到数据占用带宽。

Half ACK：灵感来源于我们观察到老师提供的样例程序中的输出信息存在“Send half ACK 11 12 15 19”，于是认为可以在每一次发送 NAK 时，都进行检测是否有多于 1 个的帧已经到达，如果是，则发送 HALF ACK 帧，该帧可以携带多个已经到达的帧的序号，接收方收到后则停止这些这些帧的计时器。进一步防止无用的重传。

4.2.2 协议参数的选取分析

窗口大小：

窗口大小仍须满足上文提到的内容。

$$W \geq 4.05$$

$$W \geq 2 + 4\alpha = 6.11$$

$$W \geq 2\alpha + 1 + \frac{t}{263}$$

除此之外，由于本协议可以节省下数据超时多发无用数据帧的带宽，所以窗口越大，就越能够利用这段时间发送出去有用的帧，故 W 调节为 64（最多 128 个定时器）。

数据超时时间 t：

数据的超时仍应满足下述关系来占满带宽：

$$t > 1875ms$$

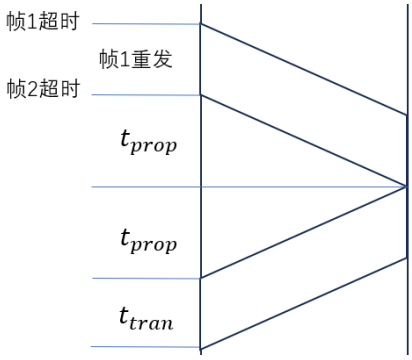
$$W \geq 2\alpha + 1 + \frac{t}{263}$$

由于存在超时重发帧数量的限制，数据应当在允许的范围内越早重传越好。

若超时时间较大，很容易把数据都堆到后面再发送，误码率较低时理论上没有什么影响，但是误码率较高时，发送窗口内的数据发完了之后就容易出现很大的空缺，导致带宽占不满。

所以我们选择参数为 1950ms

Ack 超时时间：



与之前的分析相同，满足如下条件，设置为 270ms

$$t_1 > \frac{(256 + 7) * 8}{8000} = 263ms$$

数据超时后再次延时时间：

一般一次环回即可传送回一帧，此时可以再次继续发送，如右图所示，故

$$t_2 > 270 * 2 + 263 = 803ms$$

然而考虑既然存在数量限制，实际上只要该时间设置的不过大，较小的数据超时时间即使时间到也会再次计时，但是较小的时间块使得协议十分灵活，到时间了就会发送，只是需要计算机做更多的处理。故设置为 100ms。

4.2.3 实验结果分析

实验结果如下：

序号	命令选项	说明	运行时间(秒）	Selective 算法线路利用率(%)		存在的问题
				A	B	
1	--u	无误码信道数据传输	1200	53.54	96.96	
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	51.54	95.05	
3	–f -u	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	96.94	96.94	
4	--f	站点 A/B 的分组层都洪水式产生分组	1200	94.35	94.80	
5	-f –b 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	75.59	76.05	与理论值仍有较小差别

4.2.4 存在的问题

基本已经满足需求，但是由于这种 Half ACK 是只能在发送 NAK 之后发送，故在第一个 NAK 的时间里总是无法响应，这样导致了最后必定存在一些多余帧“在窗口外”或者是“已收到过”，使用这种方法的高误码率下的效率注定要距离理论值差一点。

解决方法有多种，一是改变 Half ACK 的发送逻辑，使其不只是在发送 NAK 之后发送，但是这个条件不易判定，并且如果将这种方法发挥到极致，那么本质就是对每一帧都发送了 ACK，这样 ACK 必然占用较多的带宽，仍旧会拉低一点效率。总之使用 Half ACK 无法完全发挥潜力，达到最大效率。

方法二是启用多 NAK，可以对每一帧单独发 NAK，这样只对错

帧处理，必定占用较少的带宽，并且几乎不会出现多余帧，是理想的最好方法。

4.3 启用多 NAK 的选择重传

4.3.1 原理分析

Half ACK 的发送逻辑，是在发送 NAK 之后发送。本可以修改将这种方法发挥到极致，那么本质就是对每一帧都发送了 ACK。这样 ACK 必然占用较多的带宽，仍旧会拉低一点效率。总之，继续使用 Half ACK 无法完全发挥潜力，达到理论最大效率。

在原来的代码基础上，我们衍生出了启用多 NAK 的方法。可以对每一帧单独发 NAK，这样只对错帧处理，必定占用较少的带宽，参数选取得当，几乎不会出现多余帧

4.3.2 协议参数的选取分析

窗口大小：

窗口大小仍须满足上文提到的内容。

$$W \geq 4.05$$

$$W \geq 2 + 4\alpha = 6.11$$

$$W \geq 2\alpha + 1 + \frac{t}{263}$$

同理本协议可以节省下数据超时多发无用数据帧的带宽，故 W 仍调节为 64（最多 128 个定时器）。

数据超时时间 t：

再一次深入地分析高误码率对时间信道传输的影响，在高误

码率情况下，每 14 秒会出现一次连续错误。如果出现多次 NAK，在一个帧 ACK 超时重传时，上一次发送帧如果在这个时间段被处理，第二次发送的帧就可能直接落在滑动窗口外。在这种情况下，延长数据帧超时时间就尤为重要。但是数据帧的超时时间具体计算较为困难，因此我们采用测试的方式，我们测量结果如下：

超时时间	在窗口外的帧数量	有效帧数	占有效帧的比
1950	173	3595	4.8%
2500	70	2543	2.75%
6000	0	3751	0%

由于$t \leq 263 \times (W - 3\alpha - 1) = 16043$，故可取 6000ms

Ack 超时时间：

与之前的分析相同，设置为 270ms。数据超时后再次延时时间与上述内容相同，也设置为 100ms。

4.3.3 实验结果分析

实验结果如下：

序号	命令选项	说明	运行时间(秒）	Selective 算法线路利用率(%)		存在的问题
				A	B	
1	--u	无误码信道数据传输	1200	53.45	96.96	
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	51.54	95.05	

3	-f -u	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	96.94	96.94	
4	--f	站点 A/B 的分组层都洪水式产生分组	1200	94.35	94.80	
5	-f -b 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	79.96	79.80	

5 总结

在实验探索过程中，我们以基础协议为起点，基于信道传输环境的需求，不断叠加性能优化目标与功能拓展，终于在实验的末期将性能达到了最大理论值。随着调试工作的深入推进，我们才发现，原来基础的重传协议能够有这么多潜在的可能。通过三个协议系统的比对分析，延伸出结果进一步验证了选择重传协议的正确性，并且为滑动窗口协议的优化提供了一定的思路。

四、 研究和探索的问题

1 CRC 校验

1.1 CRC 校验能力

32 位 CRC 校验码可以检测到长度小于等于 32 的所有突发错误，可以检测到所有奇数个位的错误，当一个长度大于 33 位的突发错误发生时，或者几个短突发错误发生时，一个坏帧被当做有效帧通过检测的概率为$\frac{1}{2^{32}}$.

出现大于等于 33 位的坏帧的概率如下：

设随机变量 X 表示一帧中发生错误的位数，X 满足二项分布当$n \geq 20,\ p \leq 0.05$时，用泊松分布逼近二项分布误差很小，此时$\lambda = np = 256 * 0.00001 = 0.00256$。

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

下面使用最大帧长 2104 来计算，因为按最大帧长 256 计算数据过小，精度受限。

$$P(33 \leq k \leq 2104) = 5.163 \times 10^{-93}$$

$$P(\textit{检测不到错误}) = P(33 \leq k \leq 256) \times 2^{-32} = 1.202 \times 10^{-102}$$

每天发送帧的个数为

$$\frac{8000 * 60 * 60 * 24}{8 * 263} * 0.5 = 164258$$

理论上发生一次误码需要

$$\frac{1}{365 * 164258 * P} = 1.38763316 \times 10^{94}\ 年$$

除此之外，数据链路层的更高层也会存在其他检错纠错措施。

1.2 CRC 校验和的计算方法

两种计算方法是等效的。查表法根据异或操作满足交换律和结合律，将手动计算需要的大量异或操作以一个字节为单位提前计算出来，存储进表格，最后根据移位和查表实现快速运算。

构造方法如下：

```
static uint32_t crc32table[256];
```

```
static void hash_makeCRC32table(unsigned int table[]){
    unsigned int value;
    unsigned int i, j;
    for(i=0; i < 256; i++){
        value = i;
        for(j=0; j < 8; j++){
            if(value & 1){
                value = (value >> 1) ^ 0xedb88320;
            }
            else{
                value = (value >> 1);
            }
        }
        table[i] = value;
    }
}
```

计算 CRC16 和 CRC32 所用的时间如下（10 万次）：

crc32 = 0xD6FA738C, avg time = 0.730 us

crc16 = 0xCFC3, avg time = 1.270 us

可见 CRC16 的确更慢，但是所用时间比 CRC32 的 2 倍稍少。RFC1662 中的参数 fcs 是一个“前置”的校验值，PPP 协议中的 FCS 是按帧逐步更新的。每一帧的校验和基于前一帧的 FCS 和当前帧的数据计算。FCS 用于串联多个数据块的 CRC 校验，从而实现连续的帧校验

2 程序设计方面的问题

跟踪功能便于调试，在我们的程序中多次使用了给出的跟踪功能进行调试，比如输出窗口状态，输出帧落在窗口外的信息等。

我们实现的 get_ms()函数如下，利用了 windows 提供的获取时间的 API：

```
#include <stdio.h>
#include <windows.h>
```

```
unsigned long start=GetTickCount64();

unsigned long get_ms() {
    return (unsigned long)GetTickCount64()-start;
}
```

lprintf 的实现接受 const char *format,...的参数，利用 va_list 读取参数，传入__v_lprintf 进行格式化输出。

ACK 计时器关注是第一个被成功接收但是没有发出 ACK 的帧，且不带序号，多个帧共用一个 ACK 计时器，时间到了就发送 ACK，因此调用 start_ack_timer 后不会被清零。而 DATA 计时器与数据本就是一一对应的关系，带有序号，所以重置某一个序号对应的计时器时应当清零重新开始计数，到时间则重发。

3 软件测试方面的问题

设计这么多测试方案是为了保证协议在各种物理环境下都能够正常工作，提高协议的适应性和可靠性

命令选项	说明	测试系统何种环节以及什么情况下会失败
--u	无误码信道数据传输	初步考察协议的正确性，考察数据包的设计是否冗余，是否能够达到较高的利用率
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”	主要测试 ACK 定时器和捎带应答的效果，对错误的处理是否正确
-f -u	无误码信道，站点 A 和站点 B 的分组层都洪水式产生分组	主要测试捎带应答能力和设计是否冗余
--f	站点 A/B 的分组层都洪水式产生分组	测试流量控制能力和错误处理的能力，流量控制较弱则容易产生死锁
-f -b 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}	对效率和设计有高要求，容易效率低下或者出现各种错误。

4 对等协议实体之间的流量控制

实现了基本的流量控制，物理层和网络层都可以根据流量适时关闭和打开，通过 ack，nak，超时重传等交互防止一方不断发送数据淹没另一方。

5 与标准协议的对比

没有考虑包大小不固定的情况，而且在实际中延迟，抖动，误码率都收到周围环境的影响，较为随机。协议需要能够保持稳定，并且能够从错误中恢复。与标准 LAPB 协议和 CCITT 协议的对比，我的协议还缺少数据链路的建立、维持和拆除功能以及更高的可靠性，无法在较恶劣的网络条件下维持稳定的通信。

五、 实验总结

本次实验围绕数据链路层原理展开，核心任务是自行编程实现滑动窗口协议。我们选用选择重传（SR）协议，在仿真环境中分别构建了无噪音与有噪音的信道场景，力求达成两站点间在各种情况下的双工通信。设定信道模型为 8000bps 全双工卫星信道，传播时延 270 毫秒，误码率 10^{-5} ，且网络层分组长度固定为 256 字节。

在实验进程中，我们以卫星信道的传输条件为导向，首先依据已知停等协议模型（stop-wait）搭建协议基础架构。在尝试与确认中我们最终敲定了需要添加的各种变量，并能够在各种测试环境下稳定运行。紧接着，我们小组将目光聚焦在效率提升上。通过计算滑动窗口大小与超时时间，我们成功优化了带宽效率。之后

我们调整其比例，最大程度上优化了有效信息比例。这些改动的确有一定效果，在高误码率的通信环境下我们的信道利用率有一些显著的提升。但这同样是一个限制，因为距离理论值还相差了整整 20 个百分比。

当代码实现进入性能瓶颈阶段，我们只能从不同于协议框架中寻求新办法。一次偶然的比对，我们发现了样例调试过程中的输出信息与自己的差异：每一次发送 NAK 后有时会发送 HALF ACK 帧，该帧可以携带多个已经到达的帧的序号。在此发现上，我们总结了 Half ACK 帧的原理，并在代码上尝试运用了这样功能。结果也令人出乎意料：信道利用率提升了 15 余个百分点，与理论值的差距已经所剩无几。

在接下来的具体分析中，我们又发现了这一种方法的弊端。虽然 Half_ACK 基本已经满足需求，但是由于这种 Half ACK 是只能在发送 NAK 之后发送，故在 NAK 的间隔时间里总是无法响应。这种情况将会导致在 NAK 响应的时间中出现多余帧，以至于在高误码情况下注定要比理论值低。在这样的条件下又衍生出了另外的解决方案：尽可能增加 Half ACK 对 NAK 的响应，或者启用多 NAK。第一种方法的极致是对每一帧都发送 ACK，这显然是不可能的。因此我们再次调整 NAK 发送策略，通过测试、分析、计算，并且优化多个参数、调整多种方法，终于在最后成功达到我们计算的理论值。

在完成实验的基础上，对性能效率的不断优化的过程中，我们进

一步加深了对实验的理解，也是加深对所学知识的理解。

从实际意义来讲，可靠的双工通信是网络通信的关键根基。借由本次实验对滑动窗口协议的优化，能显著提升数据在复杂信道环境中的传输质量，为卫星通信、无线通信等领域的数据传输提供极具价值的技术参照与保障，有力推动通信技术的发展。

通过此次实验，我们收获颇丰。不仅加深了对数据链路层协议的理解，还提升了理论联系实践的能力。在亲身参与协议设计与调试的过程中，对实际分层协议结构有了更为深刻的认知。