**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** | |
|  |  |  | |  |  | |
|  |  |  | |  |  | |
|  |  |  | |  |  | |
| **实**  **验**  **内**  **容** | 本次实验选用的滑动窗口协议为选择重传/回退N步协议，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/回退N步协议，并且未使用/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 | | | | | |

目录

1. 实验内容和实验环境描述
2. 实验任务和内容
3. 实验环境
4. 软件设计
5. 不带ACK计时器的GBN协议
6. 带ACK计时器的GBN协议
7. 带ACK计时器和NAK帧的GBN协议
8. SR协议
9. 性能测试及实验结果分析

1. 不带ACK计时器的GBN协议

2. 带ACK计时器的GBN协议

3. 带ACK计时器和NAK帧的GBN协议

4. SR协议

1. 研究和探索的问题
2. 实验总结和心得体会
3. 源代码清单
4. 组员个人任务描述和心得体会

一、实验内容和实验环境描述

1.实验任务和内容

（1）本次实验通过对数据链路层的学习，实现数据链路层的滑动窗口协议。

（2）滑动窗口协议有两个主要目标，分别是在有噪声的信道下无差错传输和充分利用信道带宽。

（3）主要实现两种滑动窗口协议，分别是GBN协议和SR协议。

（4）本次实验网络层和物理层的协议已经设计好，不需要再改变，只需要对数据链路层的协议（即实验资源文件中的datalink.c文件）进行编写来实现两个虚拟站点之间的通信。

（5）设计协议后，需要进行多种环境的模拟来测试协议性能。

（6）通过协议来理解CRC循环校验的实现。

2.实验环境

（1）本次实验可以在Windows或者是Linux下进行，因为实验内容主要涉及C语言的编写，所以两者实验过程相差并不算大。

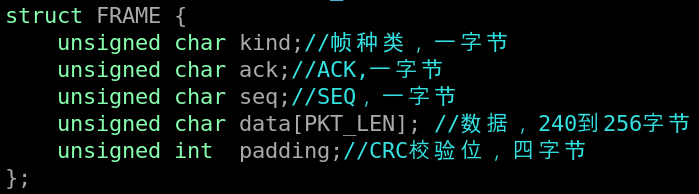
（2）选择Linux作为实验系统，再编写完datalink.c源程序后，使用Make工具进行编译链接生成可执行文件datalink，用该可执行文件测试滑动窗口协议的性能。

二、软件设计

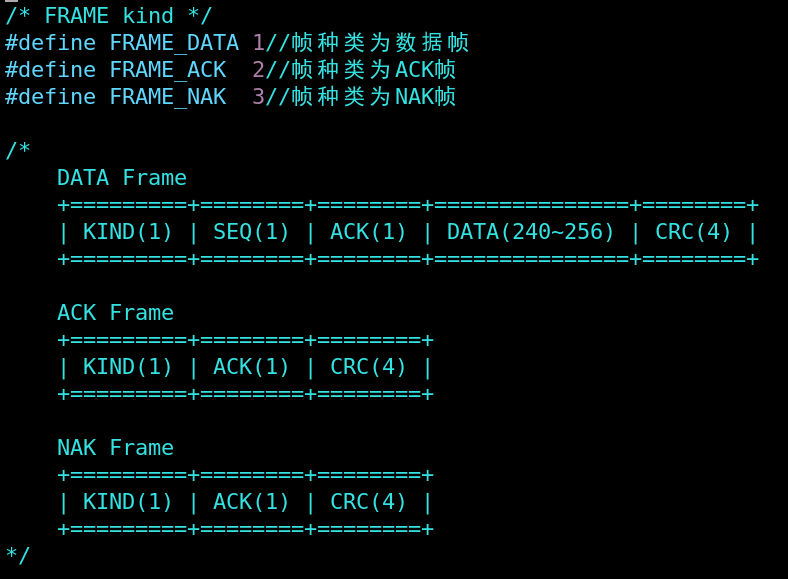
1.不带ACK计时器的GBN协议

（1）数据结构

帧结构体：



其中，帧分为数据帧，ACK帧，NAK帧三种，相应的帧组成如下：



其中，kind一字节，seq一字节，ack一字节，data240-256字节，crc四字节。

一个数据帧的长度并不是可变的。通过查阅指导书，发现网络层中的包长为固定的256字节，因此一个数据帧的长度为固定的256+1+1+1+4=263字节。

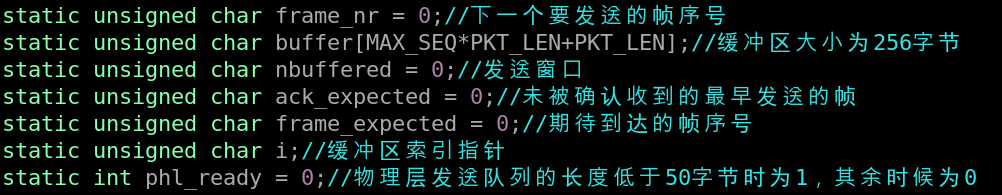
Kind表示帧种类。

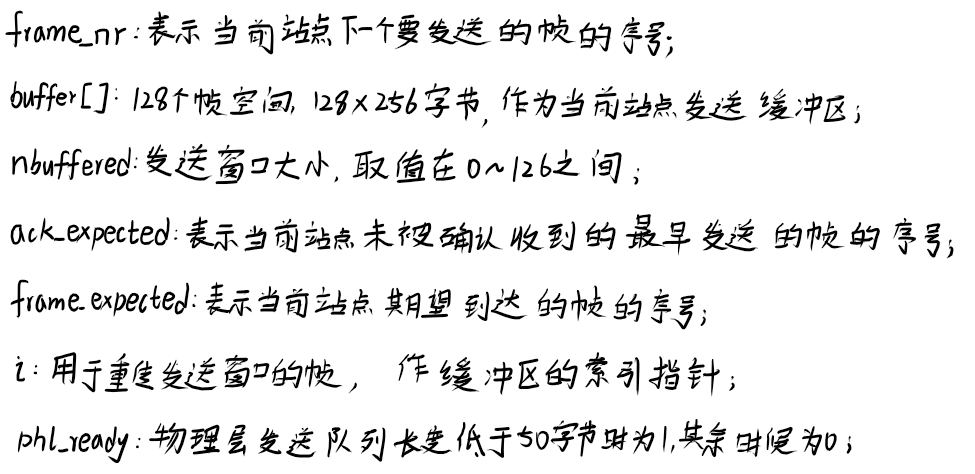
Seq表示帧序列号。

Ack表示发送该帧的站点的上一接收的帧序列号。

Crc用于差错检验。

定义的全局变量：

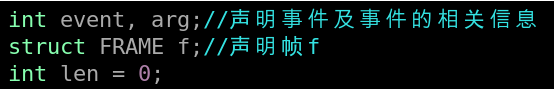




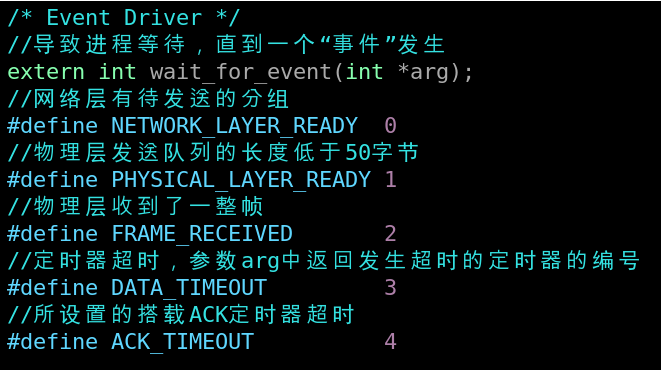
发送窗口和接收窗口大小的确认：

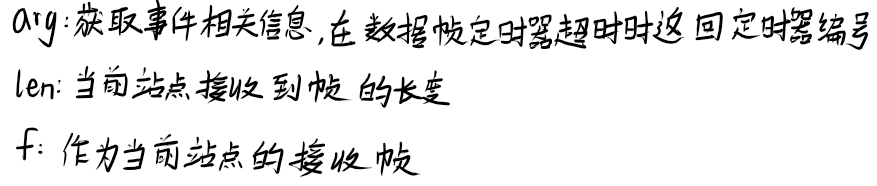
在GBN协议中，小于等于2^n-1，这个实验中n是8位，则发送窗口最大是255。要使理想信道利用率达到100%，使a等于传播时延比发送时延，传播时延为270ms，发送时延为帧长度比上发送比特率，也就是263\*8/8000=263ms，则a=270/263=1.03，在不是捎带应答的GBN协议中，理想信道利用率U=W/(2a+1)，则发送窗口大小W大于3，选择接收窗口大小上限为127，接收窗口大小为1；若是捎带应答的GBN协议，U=W/(2a+2)，则发送窗口大小大于4，选择发送窗口大小上限为127帧，接收窗口大小为1；若是捎带应答的SR协议，则U=W/(2a+2)，发送窗口大小大于4，由于发送缓冲区大小要与接收缓冲区大小相等，且小于等于2^(n-1)，即最大为128，设置发送窗口和接收窗口大小上限为64。

主函数中的变量







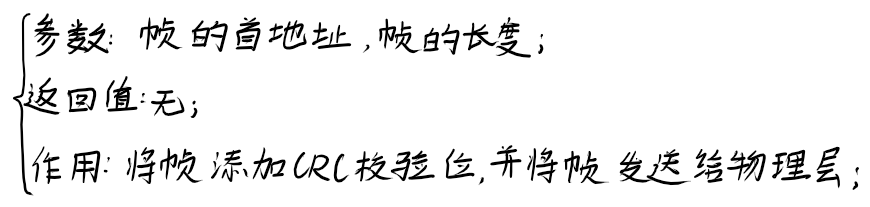


（2）模块结构

子程序及其功能

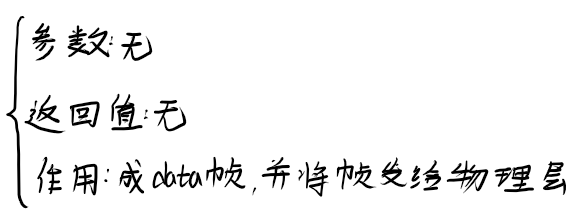
put\_frame函数





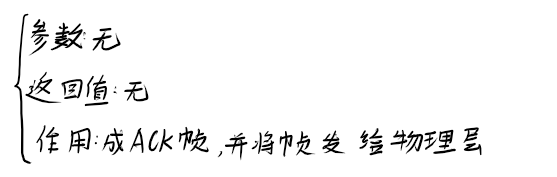
send\_data\_frame函数





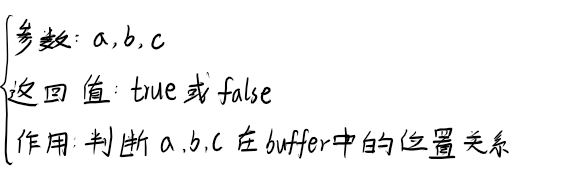
send\_ack\_frame函数





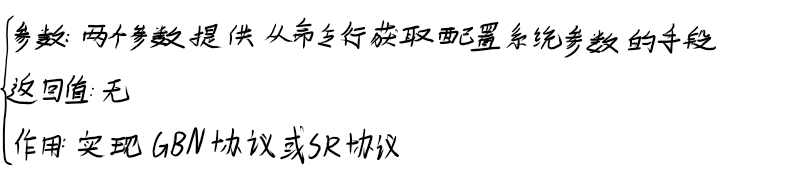
between函数



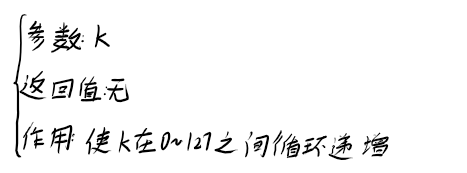


main函数



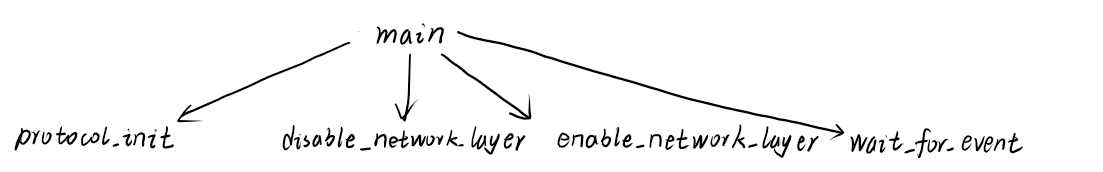


inc函数

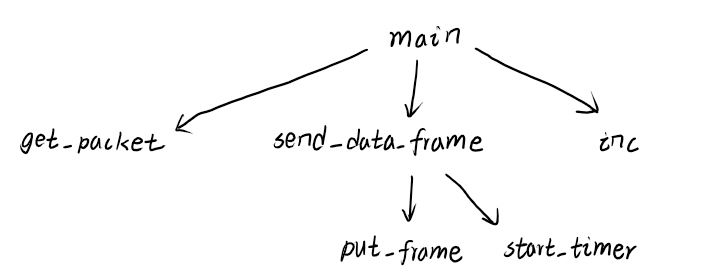


函数调用关系

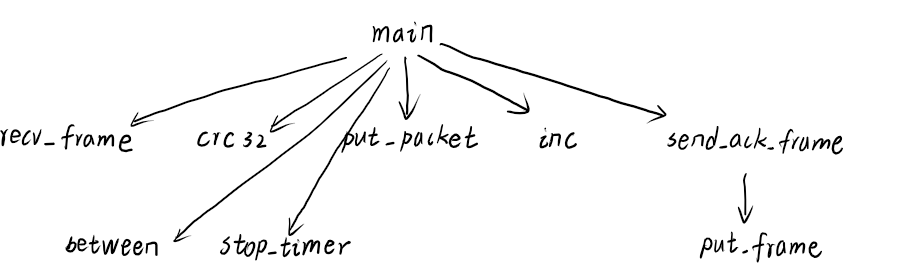
控制及初始化



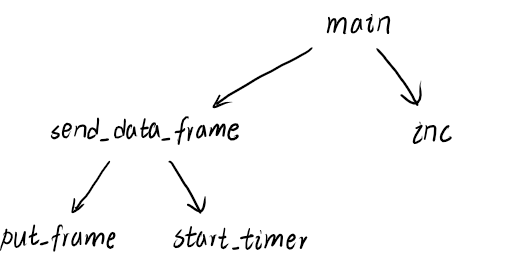
发送帧



接收帧

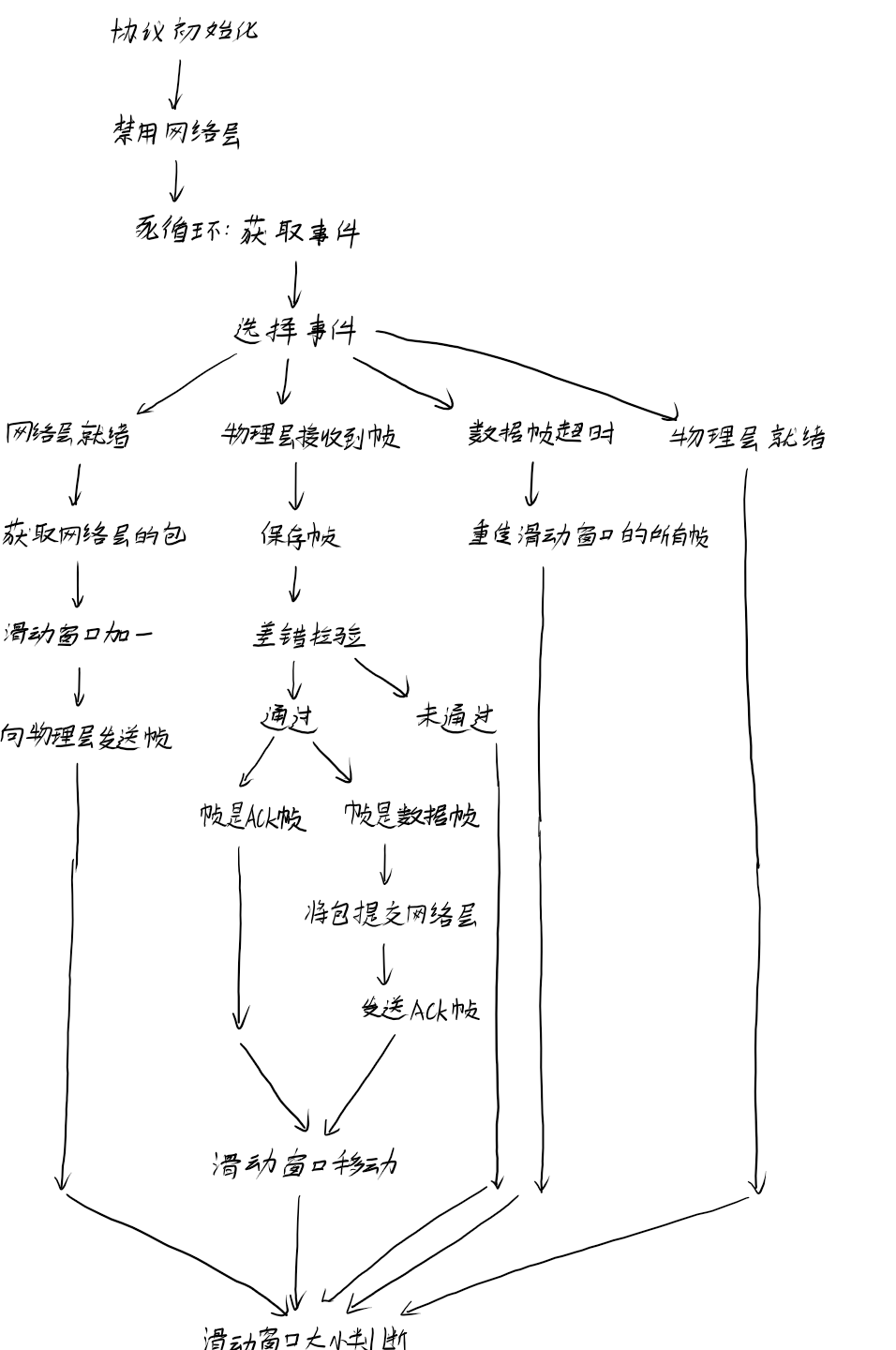
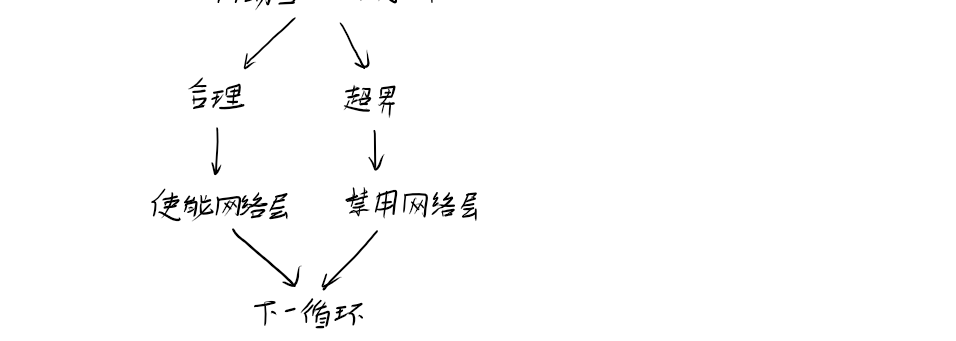


重传帧



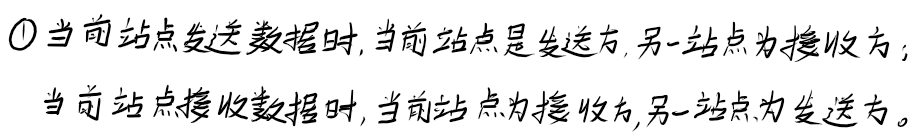
（3）算法流程

流程图

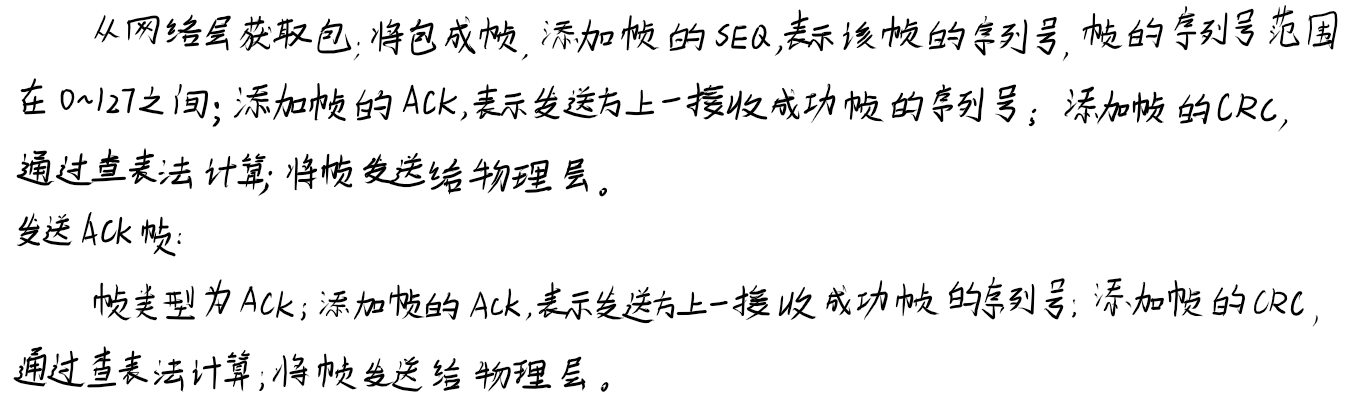
  


算法的主要流程

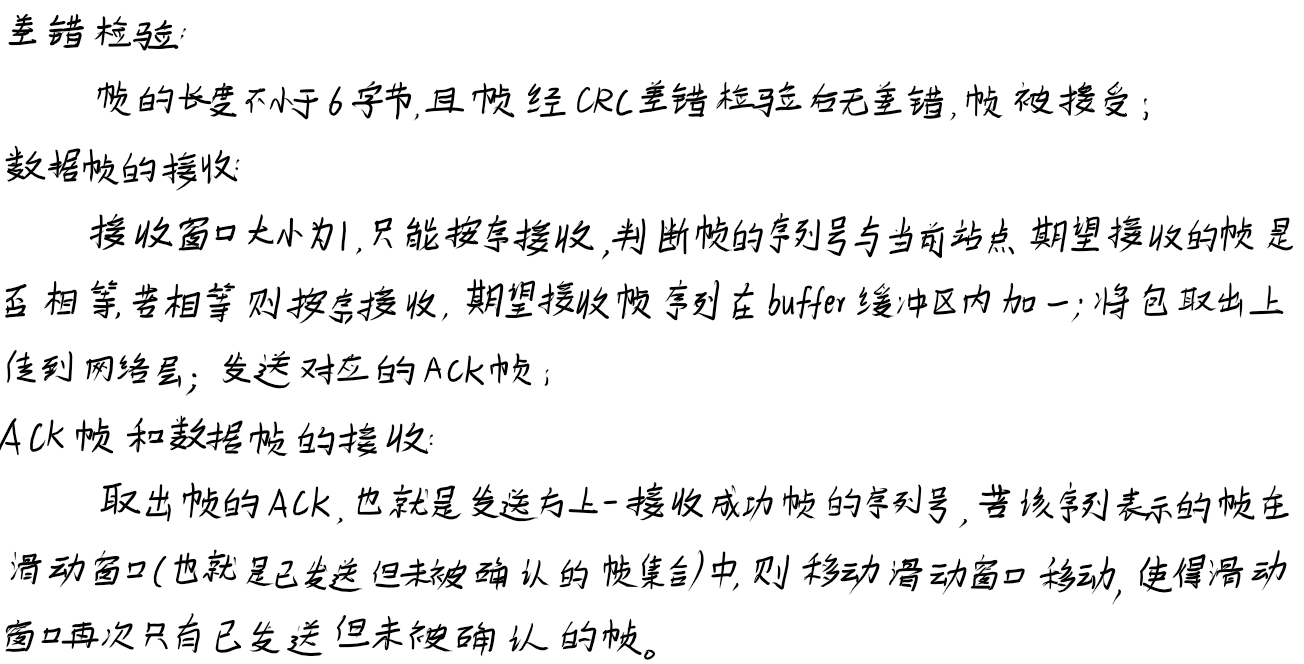
接收方和发送方



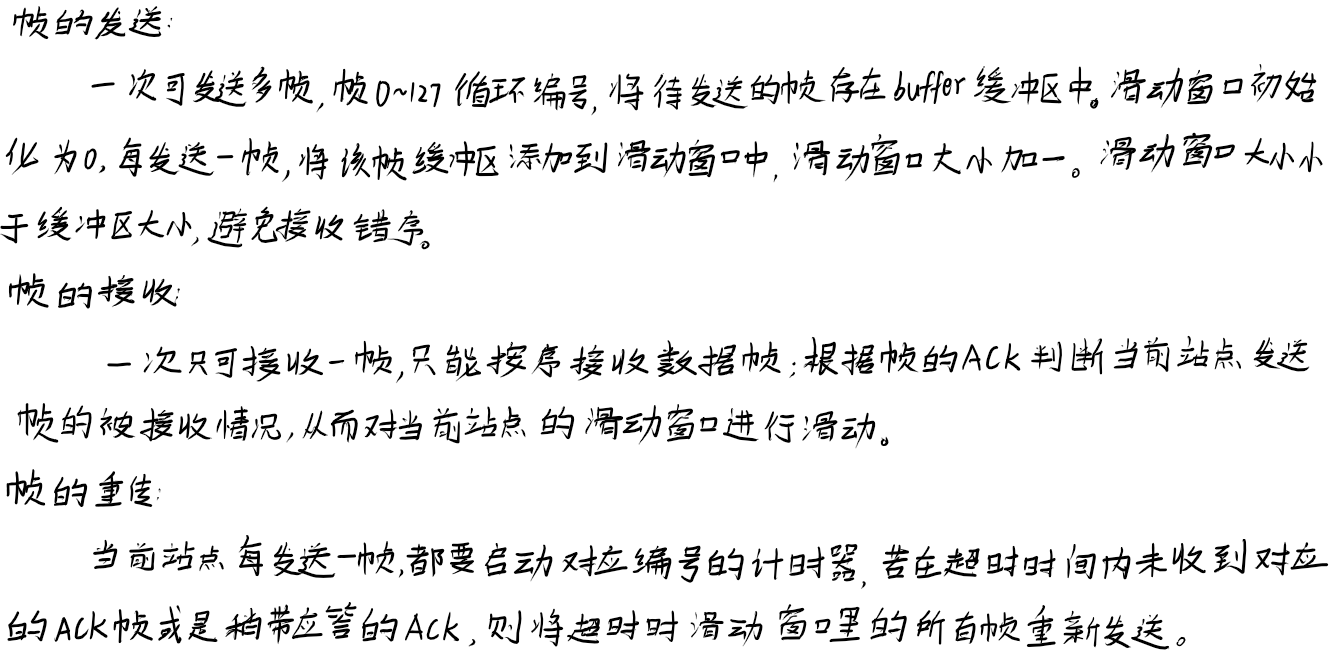
帧的发送



帧的接收

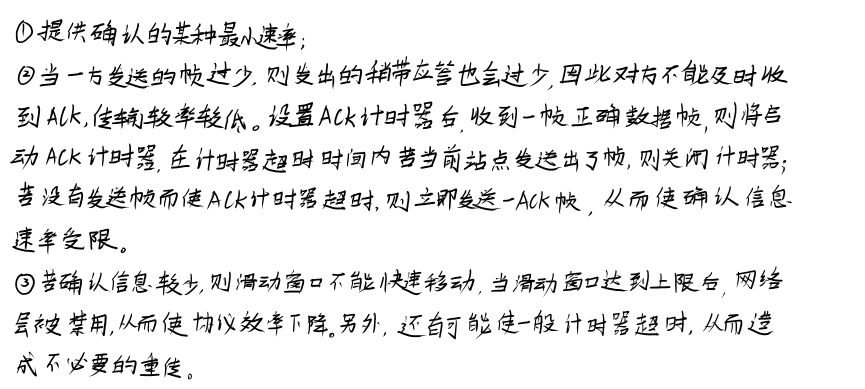


GBN协议

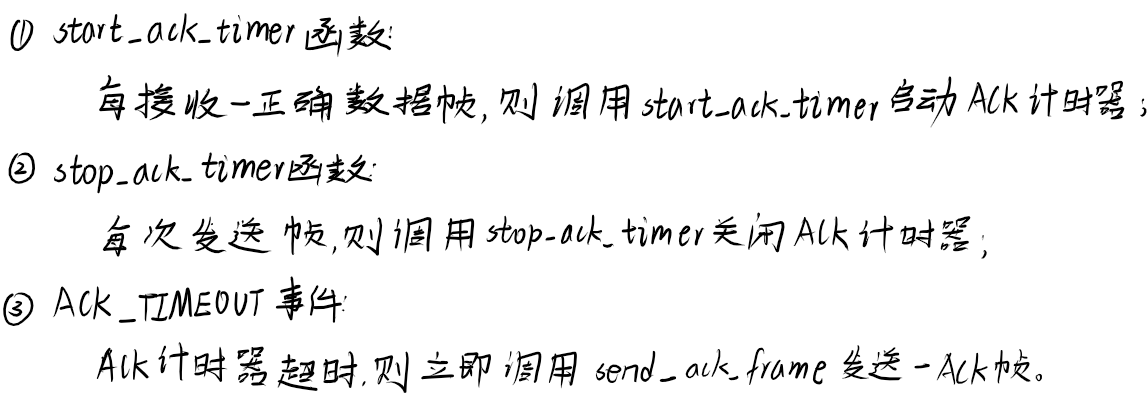


2.带ACK计时器的GBN协议

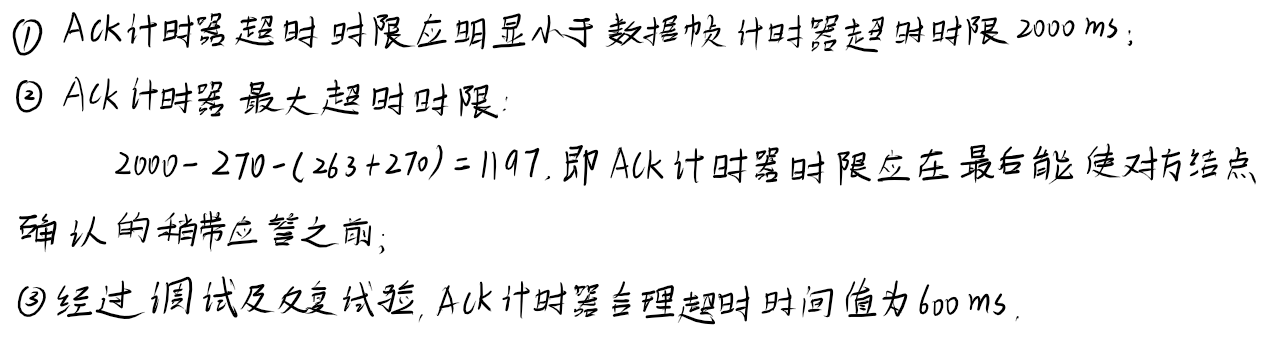
（1）ACK计时器的作用



（2）程序中实现

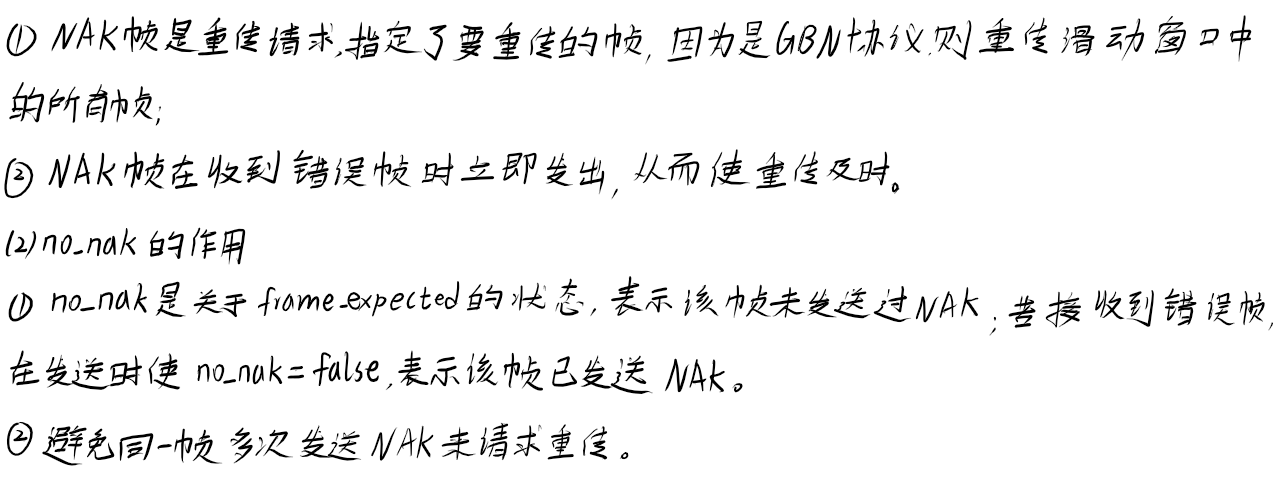


（3）ACK计时器超时时限的确定

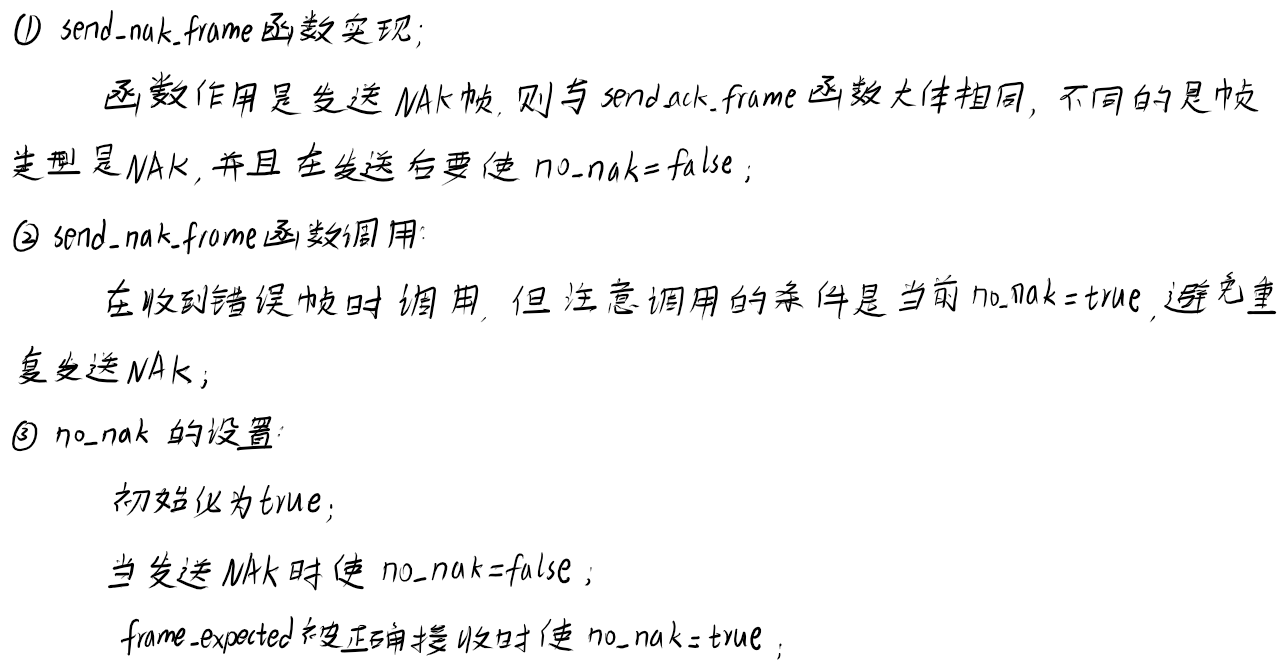


3.带ACK计时器和NAK帧的GBN协议

（1）NAK帧的作用



（2）程序中实现



4.SR协议

（1）接收窗口原理

接收窗口可以一次接收多帧，还可以错序接收，正序上传网络层，从而大大增加协议效率。

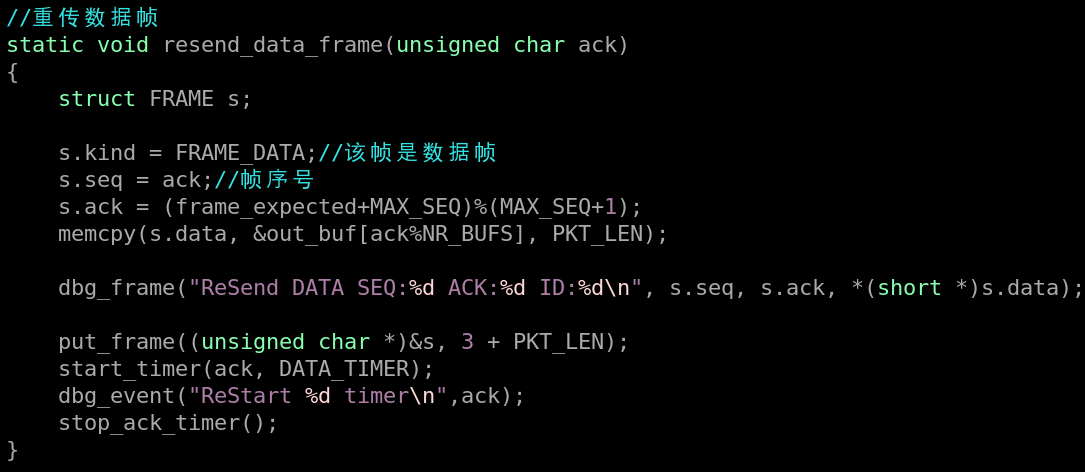
因此，接收窗口大于一。每成功向网络层正确上传一包，接收窗口向前移动一帧。而从物理层接收到的帧应该通过帧的序列落在接收缓冲区的正确位置，在缓冲区最后一个位置不为空，我们将该帧解包上传，这样我们就能正确上传包。

发送窗口或接收窗口大小应该小于等于帧序列号总数的一半。这样是为了避免帧到达时落在缓冲区的错误位置而被错误上传。

接收方缓冲区被协议设置为固定的，即等于接收滑动窗口大小。

（2）接收窗口的实现

接收方期待错误帧只重传一次，因此我们设计一个重传帧函数，当发送方收到NAK时，重传对应的错误帧。函数源码：



发送窗口和接收窗口大小之和不大于帧序列号最大值加一。我们需要将原来的buffer数组改进为in\_buf和out\_buf数组，分别表示接收缓冲区和发送缓冲区，且缓冲区大小为原数组的一半。发送窗口下界为ack\_expected，上界为frame\_nr-1，接收窗口下界为frame\_expected，上界为too\_far-1，接收窗口大小等于接收方缓冲区大小。

arrived数组：该数组是布尔类型数组，我们用这个数组来表示接收缓冲区状态，数组大小与接收缓冲区大小相等，当对应数组元素为真，则表示对应接收缓冲区已满，若为假则表示缓冲区为空。设置这个数组的含义是为了在接收数据时已知缓冲区状态，这样可以将帧放入缓冲区正确位置；另外在上传包时，缓冲区不为空才可以上传对应的包。

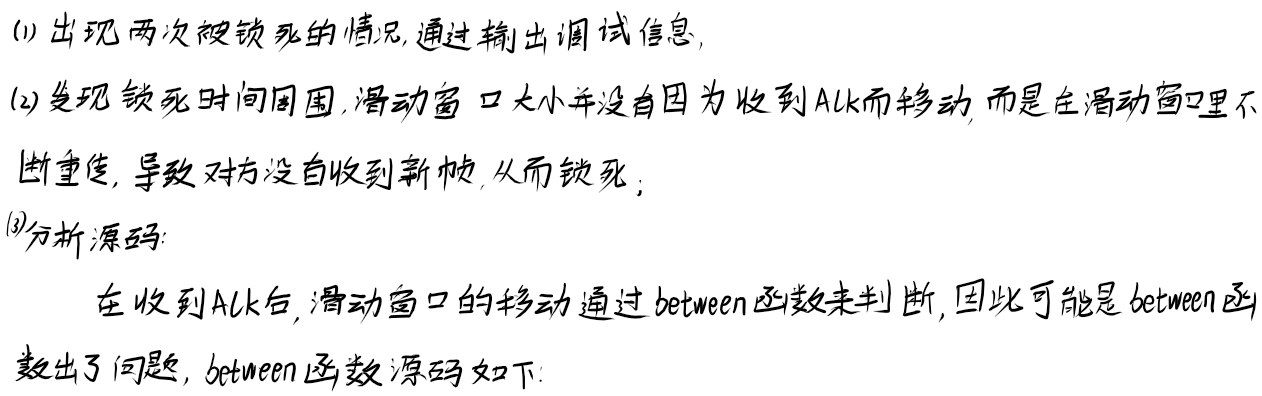
接收窗口若正确收到数据帧，则会直接将帧存入in\_buf数组，并且将对应的arrived数组元素标记为真，通过判断接收窗口的最后一个位置是否为空，将该位置的帧解包上传，并移动接收方滑动窗口。则主要改动发生在收到数据帧时：

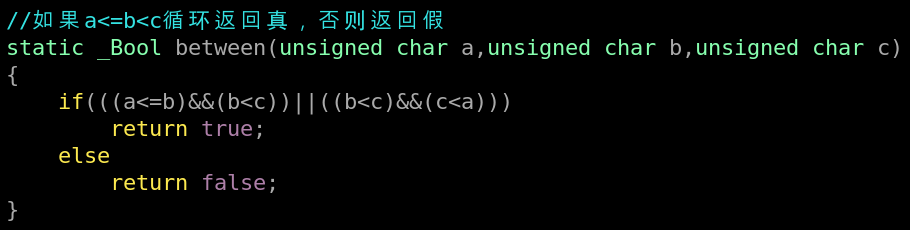


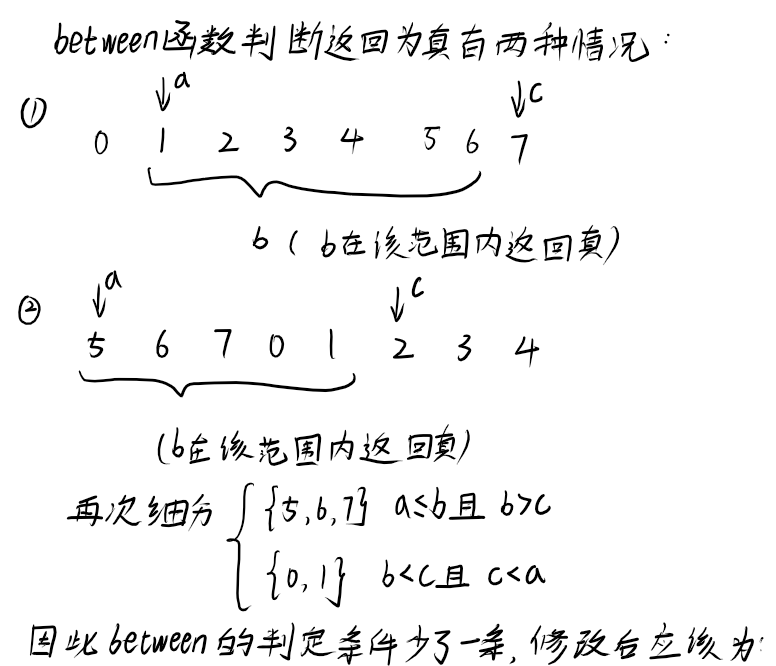
三、实验结果分析

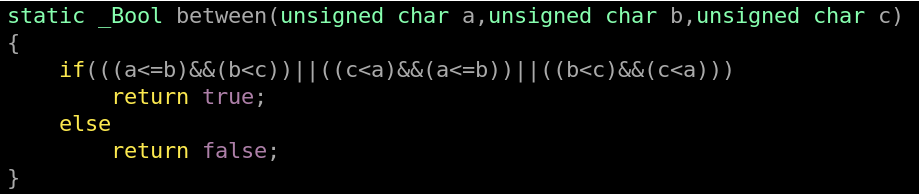
1.不带ACK计时器的GBN协议

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 序号 | 命令 | 说明 | 运行时间（秒） | 接收方线路利用率（%） | | 存在的问题 |
| A | B |
| 1 | datalink au  datalink bu | 无误码信道数据传输 | 1200 | 53.33 | 95.54 | 无 |
| 2 | datalink a  datalink b | 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，慢发 100 秒” | 1200 | 34.27 | 64.57 | 无 |
| 3 | datalink afu  datalink bfu | 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 | 94.46 | 94.46 | 无 |
| 4 | datalink af  datalink bf | 站点 A/B 的分组层都洪水式产  生分组 | 1200 | 69.66 | 70.03 | 无 |
| 5 | datalink af –ber 1e-4  datalink bf –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 | 20.78 | 18.66 | 无 |



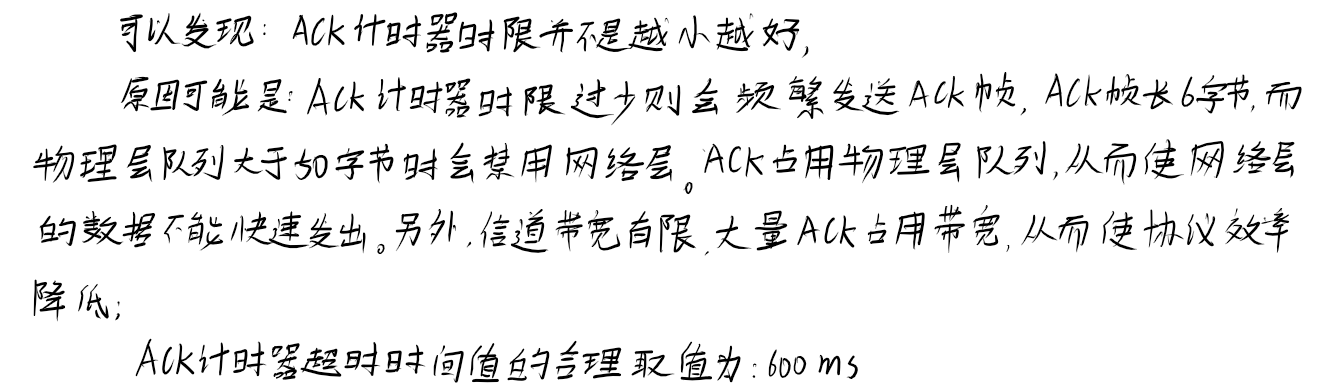




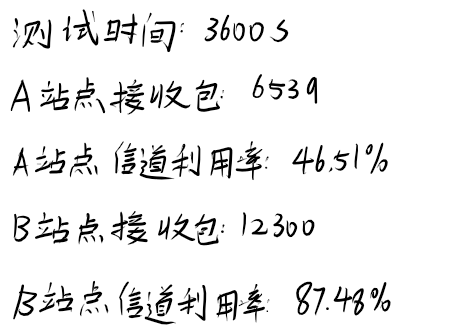


2.带ACK计时器的GBN协议

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ACK\_TIMER(ms) | 运行时间(s) | A利用率(%) | A收包数 | B利用率(%) | B收包数 |
| 0 | 1200 | 39.20 | 1837 | 76.64 | 3588 |
| 200 | 1200 | 39.95 | 1871 | 76.82 | 3599 |
| 400 | 1200 | 38.10 | 1785 | 75.58 | 3538 |
| 600 | 1200 | 40.87 | 1912 | 78.43 | 3672 |
| 800 | 1200 | 37.80 | 1771 | 74.85 | 3506 |
| 1000 | 1200 | 38.91 | 1823 | 74.85 | 3490 |
| 1200 | 1200 | 40.30 | 1888 | 74.96 | 3514 |



3.带ACK计时器和NAK帧的GBN协议



4.SR协议

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 序号 | 命令 | 说明 | 运行时间（秒） | 接收方线路利用率（%） | | 存在的问题 |
| A | B |
| 1 | datalink au  datalink bu | 无误码信道数据传输 | 1200 | 51.57 | 96.97 | 无 |
| 2 | datalink a  datalink b | 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，慢发 100 秒” | 1200 | 50.60 | 89.11 | 无 |
| 3 | datalink afu  datalink bfu | 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 | 无 |
| 4 | datalink af  datalink bf | 站点 A/B 的分组层都洪水式产  生分组 | 1200 | 93.55 | 94.11 | 无 |
| 5 | datalink af –ber 1e-4  datalink bf –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 | 50.18 | 60.85 | 无 |

在测试过程中，有出现因为包顺序上传错误而导致协议失败的情况。通过输出调试信息，发现是因为重传的帧落在了错误的接收缓冲区内，因此被错序上传。而重传的帧之所以落在错误信息，发现TIME\_OUT发生后重传的帧序号oldest\_frame在协议中并没有改变，通过查阅教材发现oldest\_frame所指的是超时计时器的编号，而event返回的arg参数正是超时计时器编号，因此只需要超时重传时重传该计时器对应的数据帧，就可以将该帧落在接收缓冲区正确位置。

同时，计时器编号应该为帧的seq，而不是seq%NR\_BUFS，若采用后一种方式，则会有两个不同序列的帧对应同一个计时器，容易造成错误。采用第一种方式后，这样计数器数量虽然翻倍，但是逻辑更加清晰。我们通过发送数据帧时启动对应序列的计时器，通过接收ACK停止发送缓冲区的对应帧的计时器。若ACK和SEQ序列不同却对NR\_BUFS取余相等，则会导致启动seq%NR\_BUFS相应的计时器，被ack%NR\_BUFS停止相同的计时器，而这时发送和接收的不是同一帧，操作的却是同一计时器，出现逻辑错误。

四、研究和探索的问题

1. CRC校验能力

该客户使用本次实验描述的信道，客户的通信系统每天的使用率 50%，即：每天只有一半的时间在传输数据，那么，根据你对 CRC32 检错能力的理解，发生一次分组层误码事件，平均需要多少年？

CRC校验和出错的概率是非常低的。可能数十年会出现一次，这个概率不会影响用户的实际体验，是可以忽略。

2.CRC 校验和的计算方法

校验和通过查表法直接计算，这相对于按位模2除法更加高效。

3.程序设计方面的问题

（1）8.10 节提出的协议软件的跟踪功能有什么意义？你的程序实现这样的功能了吗？

可以调试输出来实时观测软件的正常运行，我的程序多次调用dbg输出来观察缓冲区变化或者是显示帧的发送与接收信息，这些在调试过程中非常有用。

（2）程序库中获取时间坐标的函数 get\_ms()不是 C 语言标准库中的函数，你能自己实现一个这样的函数吗？可以实现类似的函数。

（3）8.9 节中给出了两个函数 start\_timer()和 start\_ack\_timer()，它们都是定时器函数，两个函数启动定时器的时机不同，而且在定时器到时之前重新调用函数对原残留时间的处理方式也不同，为什么要这样设计？

两个计时器的作用不同，数据帧计时器是为了确认帧的正确到达而设置的，而ACK计时器则是为了确定发送ACK的最小速率。

4.软件测试方面的问题

（1）表 3 中列出了七种测试方案，设计这么多种测试方案的目的是什么？

有效测试协议的成功与健壮性。

（2）分析每种测试方案，每种方案主要是为了瞄准你的协议软件中可能出问题的哪些环节？或者说，你的协议软件存在什么问题时，测试会失败？

无误码信道数据传输：

测试帧的正确发送与接收，正确成帧和解包；

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，慢发 100 秒”：

模拟服务器-用户之间的点对点通信，测试在有误码情况下数据的正确传输及信道利用率；

无误码信道，站点 A 和站点 B 的分组层都洪水式产生分组：

测试缓冲区的正常性，并且得到理想情况下信道利用率；

站点 A/B 的分组层都洪水式产生分组：

在有误码情况下测试信道可达利用率；

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：

测试高误码情况下协议的健壮性及可达利用率。

（3）你觉得还存在有哪些问题是这些测试尚未覆盖的？这些测试方案和验证协议正确性的手段由指导教师给出，如果是由你自己独立完成整个协议的设计和测试，你会采用哪些手段来验证你的程序能正确工作？

最真实的情况下可能复杂的多，数据的发送频率可能是非常不稳定的。

如果我独立完成协议的设计测试，我会将模拟站点之间的通信换为真实两个站点之间的通信，设计有线信道和模拟信道两种模式，当然，这并不是简单的工程，但这是更接近真实情况的方案。

（4）针对本次实验的具体问题，你能不能提出一种更高效的软件测试方案？

采用图形化界面，图形化接收和发送缓冲区的实时状态，实时显示信道利用率和数据的传输情况。

（5）本次实验所提供的程序库还有哪些不足， 怎样才能对协议开发提供更方便的支撑，这关系到在整体软件开发过程中的不同模块间的功能划分问题， 给出你的建议。

模块划分不够明显，不能有效区分物理层、数据链路层、网络层的协议，建议清晰将这三个模块划分开来，以达到网络开发的高效。

5.对等协议实体之间的流量控制

你认为你所设计的滑动窗口协议软件有没有解决两个站点的数据链路层对等实体之间的流量控制问题？如果是已经解决了，那么是怎样解决的？

通过使能或者禁用网络层，使能或者禁用物理层实现流量控制。实际网络需要动态控制流量。

6.与标准协议的对比

信道的稳定性，信道的误码波动，或者是时延的波动都需要考虑。

五、实验总结和心得体会

本次代码编写调试大约8小时，由于使用的是Linux系统来开发，Make工具十分简洁高效，因此只要书写合格的C程序，就能快速测试协议。协议方面遇到数个问题，包括包与帧的组成问题，缓冲区判定问题，ACK计时器和NAK帧的探究，接收窗口的工作原理探究。通过不断复习及不断调试，终于对问题有了解决。通过这次实验，我对数据链路层的滑动窗口协议有了较为底层的认识；对C语言有了更多的理解，C语言是适合底层开发的语言。

六、源代码清单

**本实验报告中没有截图，但附件包含多个测试结果文件（\*.log）以提供更加清晰的结果展示。**

1. 不带ACK计时器的GBN协议文件夹

文件夹名称为noackgobackn。

文件夹含有数据链路层文件datalink.c。

以及多个测试结果文件：

有无误码信道数据传输的测试结果au.log和bu,log，

有无参数测试结果a.log和a.log，

有无误码信道和洪水方式测试结果afu.log和bfu.log，

有洪水方式测试结果af.log和bf.log，

有洪水方式线路误码率设为 10-4测试结果afber.log和bfber.log。

1. 带ACK计时器的GBN协议文件夹

文件夹名称为ackgobackn。

文件夹含有数据链路层文件datalink.c。

以及多个测试结果文件：

有ACK计时器超时时限为0ms的测试结果a0.log和b0.log，

有ACK计时器超时时限为200ms的测试结果a200.log和b200.log，

有ACK计时器超时时限为400ms的测试结果a400.log和b400.log，

有ACK计时器超时时限为600ms的测试结果a600.log和b600.log，

有ACK计时器超时时限为800ms的测试结果a800.log和b800.log，

有ACK计时器超时时限为1000ms的测试结果a1000.log和b1000.log，

有ACK计时器超时时限为1200ms的测试结果a1200.log和b1200.log。

1. 带ACK计时器和NAK帧的GBN协议文件夹

文件夹名称为acknakgobackn。

文件夹含有数据链路层文件datalink.c。

以及多个测试结果文件：

有无参数的测试结果datalink-A.log和datalink-B.log。

4. SR协议文件夹

文件夹名称为selectiverepeat。

文件夹含有数据链路层文件datalink.c。

以及多个测试结果文件：

有无误码信道数据传输的测试结果au.log和bu,log，

有无参数测试结果a.log和a.log，

有无误码信道和洪水方式测试结果afu.log和bfu.log，

有洪水方式测试结果af.log和bf.log，

有洪水方式线路误码率设为 10-4测试结果afber.log和bfber.log。

七、组员个人任务描述和心得体会

1. 成员一：邹晟芊

我主要完成SR协议部分的实现（第三部分）。

选择重传协议在前面两位小组成员的努力下已经不算困难了，感谢团队的力量。我们距离选择重传协议就仅差一个接收窗口了。怎么实现呢？

接收窗口可不仅仅是变多了那么简单，SR协议更重要的在于其能错序接收，按序上传，而且重传数据帧时再也不需要重传发送窗口里的所有帧了，只需要将NAK对应的帧发送一次就足够了。这些功能是非常有用且强大的。

我重点确定了发送窗口以及接收窗口的大小，以及各个窗口的上下界。问题的关键在于两个窗口的改变时机。我发现调试输出函数非常有用。因此，每次窗口一改变，我便对相应的窗口上下界进行输出，从而对窗口进行实时的监控。

既然上手做实现，我对实现细节十分好奇，为什么接收窗口可以错序接收并且按序上传呢？原来，接收窗口每次只是将窗口最后一个非空区域的帧上传给网络层，而帧到达时，只只要让其落在接收窗口的正确位置就可以了，从而能够接收多帧。同时，增加一个辅助数组来实时显示接收缓冲区的空或非空的状态。

我最担心的问题还是发生了，在调试过程中，协议出现了上传包顺序错误的问题。还是以发送窗口和接收窗口为出发点，发现是因为超时重传的帧落在了对方错误的接收缓冲区内，因此被错序上传。而超时重传的帧要落在正确位置，就要有正确的序列号，我们就要知道具体是哪个计时器超了时，原来计时器超时会返回对应的编号，将重传的帧序号与计时器编号统一起来，从而对该问题有了解决。

通过这次实验，可谓是收获颇多。不仅仅对数据链路层的相关知识有了更多了解，而且对于协议的开发有了更多体会，对C语言有了更多理解，这样的动手实践充满了价值。

1. 成员二：李若彬

我主要完成GBN协议部分（第一部分）。

刚开始做实验时，在选择实验环境时，我们小组在Windows环境和Linux环境之间选择了Linux环境，原因是Linux环境的编写过程足够清爽，Make工具非常的简明且清晰，只需要一行命令，就可以生成可执行文件来进行测试。

在学习数据链路层的知识时，我们学习了大量的概念，但是对其真正的实现并不算理解，这次试验是十分宝贵的一次与协议实现交流的机会。我先对数据链路层的基础知识进行了复习，明白了数据链路层在物理层和网络层之间的作用。其次，通过对老师提供的停等协议文件进行学习，阅读源码来了解其中的模块结构和实现细节，但是直接看代码学习效率并不高，我将大量注释添加到了老师的源码里，终于对协议有了较为清晰的认知。终于，到了实打实上手的部分。GBN协议与停等协议的主要区别在于发送窗口的改进，通过数据结构和算法的实现，终于得到了一个能跑的程序。

接下来就是测试调试程序啦，果然出现了问题。那就慢慢调吧，一个程序员就是付出百分之一来写代码，剩下的百分之九十九就要调试代码了。出现了两次协议锁死的问题，屏幕上不再输出任何东西，这是一个程序员最不想看到的。怎么办，原来，有好多调试输出信息函数供我选择，通过将发送窗口的信息输出，原来协议的锁死是因为发送窗口的锁死，而发送窗口的滑动的判断函数是可以知道的，很快就对这个问题有了解决。

终于，通过输出多组调试信息，我的代码在多重测试的检验下存活了下来，可喜可贺。我连忙将工程上传到gitee上供我的小组成员来做进一步的实验。

通过这次实验，我终于对数据链路层的知识有了更深的理解，同时，这种编程对于我的编程技术提高来说可谓是作用重大。同时，期待后续同学能实现更进一步的协议。

1. 成员三：胡家田

我主要在GBN协议的基础上实现ACK计时器和NAK帧（第二部分）。

实验指导书中指出可选搭载ACK技术的GBN协议作为实现协议。通过阅读前面战友的代码（诚挚感谢他能写注释），我发现GBN协议本身已经实现了部分ACK技术，在发送数据帧时，已经将ACK捎带应答发送了出去，而在收到帧时，则会立即发送ACK帧。我于是有了疑问，ACK技术到底还有什么呢？

通过学习数据链路层相关知识，原来ACK技术不仅有捎带应答和ACK帧，重要的还要有ACK计时器。我心中有了很大的疑问，实验指导书中指出数据帧的计时器和ACK帧计时器有非常大的不同，这简直出了一个难题，两个函数启动定时器的时机不同，而且在定时器到时之前重新调用函数对原残留时间的处理方式也不同。为什么会这样呢？原来，数据帧计时器是为了确认帧的正确到达而设置的，而ACK计时器则是为了确定发送ACK的最小速率。因此数据帧计时器在启动时要将帧编号传入，而在收到相应ACK时再关闭相应编号的计时器。而ACK计时器是没有编号的，也就是只有一个，在接收到一数据帧时启动ACK计时器，而在计时器时限内若有发送帧，则关闭计时器，若计时器超限，则立即发出ACK帧。因此，有了ACK计时器的存在，确认应答信息就能以一个相对稳定的速度来发送，从而提高了协议的效率。

接下来要确认ACK计时器的时限设置。通过调试多组数据，发现ACK计时器时限并不是越小越好，也不是越大越好。就像《红楼梦》林妹妹说的：“今儿他来，明儿我再来，这样间错开来，不至于太冷落，也不至于太热闹。”

最后是搭载NAK帧，再问问为什么，NAK帧有什么用呢？原来，NAK帧是立即重传请求，以达到最快的将错误的数据帧重新发送从而提高协议效率。

通过这次实验，我对滑动窗口协议有了更为深刻的理解，期待有更多这样能使我有提升动手机会。