

**计算机网络**

**课程实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | 可靠数据传输协议-停等协议-GBN协议的设计与实现 | | | | | |
| 姓名 | 冯开来 | | 院系 | 计算学部 | | |
| 班级 | 1903602 | | 学号 | 119020125 | | |
| 任课教师 | 李全龙 | | 指导教师 | 李全龙 | | |
| 实验地点 | 格物207 | | 实验时间 | 11.6 | | |
| 实验课表现 | 出勤、表现得分(10) |  | 实验报告  得分(40) |  | 实验总分 |  |
| 操作结果得分(50) |  |
| 教师评语 | | | | | | |
|  | | | | | | |

****

|  |
| --- |
| 实验目的： |
| 1. 理解可靠数据传输的基本原理 2. 理解滑动窗口协议的基本原理 3. 掌握停等协议的工作原理 4. 掌握GBN 的工作原理 5. 掌握基于UDP 设计并实现一个停等协议的过程与技术 6. 掌握基于UDP 设计并实现一个GBN 协议的过程与技术 |
| 实验内容： |
| 1. 基于UDP 设计一个简单的停等协议，实现单向可靠数据传输（服务器到客户的数据传输）。 2. 基于UDP 设计一个简单的GBN 协议，实现单向可靠数据传输（服务器到客户的数据传输）。 3. 模拟引入数据包的丢失，验证所设计协议的有效性。 4. 改进所设计的停等和GBN协议，支持双向数据传输； 5. 基于所设计的停等协议，实现一个C/S 结构的文件传输应用. 6. 将所设计的GBN 协议改进为SR 协议 |
| 实验过程： |
| 1. **GBN客户端（停等协议只需将GBN协议窗口中大小改为1）**   进入客户端功能之后，当匹配到输入的是“-time”或者“-quit”则作为数据包发送给服务端。如果匹配到输入是“-testgbn [X][Y]”，则进入GBN传输阶段。    而如果匹配到的是“testgbn [X][Y]”的话，则开始初始化数据包。将“testgbn [X][Y]”这段报文发到服务端（服务端也会有相应函数操作）。在等待服务端回复设置为UDP为阻塞模式的时候，客户端发起握手连接给服务端，两者进入三次握手阶段（我也不知道为什么UDP传输要三次握手，只是参考代码这么写我就写了）。分别记为case 0和case 1。  第一个case 0是等待握手阶段。如果服务端发回来的报文的seq匹配的话，就说明准备文件传输，并初始化rcvSeq（上一个收到的序列号）和waitSeq（我希望收到的序列号），和。并转到阶段1。    第二个case1是等待接受数据阶段。这里面我们引入随机变量b，模拟数据包丢失的情况。如果没有丢失，那么判断是不是期望的数据包（即waitSeq – Seq是否为0），如果是期望收到的序列号，我们将缓存内容copy下来，并让制作ack（包括我确认收到的序列号、我一共收到了多少packet。    如果不是我期望收到的数据包，需要返回ack。不过这回返回的ack的数据包里面除了一共收到的packet数量不一样，其他和上面的ack一样。  最后发送ack给服务端，这一小段就完成了。     1. **GBN服务端（停等协议只需将GBN协议窗口中大小改为1）**   在服务端运行之后，首先会初始化套接字并绑定端口地址监听，监听到客户端发送的命令执行函数，如果是“-time”和“-quit”则执行返回时间并打包发给客户端和退出程序。    如果收到的是“-testgbn [X][Y]”的话，就会进入三次握手阶段，阶段过后就是数据发送阶段。    在case1也就是握手阶段，服务端首先调用recvfrom函数，接收到来自客户端的握手报文，如果recvSize小于0则说明该数据包丢失或者有错误，此时waitCount++（即等待时间加一）当waitCount超过3次（或者10次，这个可以改），则会提示超时。如果收到的是我们想要的握手的数据包，那就会初始化curSeq（现在的序列号），curAck（现在确认的ack），waitCount（重置计时器），并转到阶段2。    在case2中也就是数据传输阶段。服务端首先调用函数seqIsAvailable()函数查看是否有空的序列号，如果有空的序列号说明窗口还有剩余，然后将当前序列号+1并封装到buffer[0]，将相应长度的分组放到buffer[1]，开始将数据包发送给客户端。发完之后curSeq++并对SEQ\_SIZE取模得到新的curSeq。（因为取模会得到0，所以会在一开始会对curSeq+1）。    接下来是等待ack，如果没有收到ack，则会返回-1，并且计时器+1，直到waitCount超过10（即10次没有收到），则会重传刚刚的数据包。如果等到了想要的ack，则会重置计时器，并且如果buffer[1]（刚刚在客户端提到，会放入总共收到的packet组数）和packet总组数相等的话，就会打印数据传输完成。     1. **模拟引入丢包**   在客户端等待数据接受阶段（case1），上文中提到了引入随机变量b，如果b的返回值大于0的话，则说明丢包。我们具体来看看lossInLossRatio()函数，根据丢失率随机生成一个数字，判断是否丢失，丢失在返回TRUE，否则返回FALSE。    在返回后，会重新执行recvfrom()函数，而在服务端，因为收不到ack则会waitCount++（到一定程度会重发数据包）。     1. **支持双向数据传输**   本次实验实现双向数据传输的方法是，在运行程序后，由用户选择，该服务端（客户端）执行服务端功能还是客户端功能，所以整体来看，其实就是合并了客户端和服务端的代码。这样客户端能执行服务端代码，也能传输数据。           1. **实现C/S结构的文件传输应用**   实现文件传输，服务端首先要将文件内容拷贝到缓存。并计算出总的packet数，并且将每一个组的ack设为TURE（表明该分组还没发送成功即没有收到相应ack）    客户端在收到服务端发送的数据包之后，如果是期望收到的，则将收到的数据缓存到recvPaper中。    在服务端如果发送了所有的分组，那么将会在最后的分组的状态码设置为255，表明已经发送所有的分组。    然后在客户端，如果收到了状态码为255的分组，则表明数据传输已经完成，客户端print所有的数据，即缓存在recvPaper的数据。     1. **SR客户端**   SR客户端和GBN客户端的区别在于GBN只能接受我期望收到的seq，否则就丢弃，而SR客户端可以接受任意的seq（只要在窗口范围内）    当然如果收到的序列号已经达到我的recv\_base了那么就让窗口基数++。在SR客户端我还模拟了ack丢失的情况，同样是调用lossInLossRatio函数，如果b返回TRUE则说明丢包并重新进行接受数据。     1. **SR服务端**   SR服务端和GBN服务端的区别在于，GBN期望收到的ack只有一个，而SR可以期望收到多个ack，这个ack范围在send\_base到send\_base + SEQ\_SIZE（当然要小于总的packet数）。在接受ack阶段，如果发生了丢包或者其他错误，程序会遍历期望收到的ack，查看哪些ack是还没有收到（即没有被确认），然后对应分组i的计时器++，超过三次则重新传送分组。    当然如果收到了ack，则会对第i个分组的ack进行处理，将他设置为已经接受。 |
| 实验结果： |
| 1. 基本功能 2. 获取时间      1. 退出      1. 单向传递      1. 附加功能 2. 模拟数据包丢失   M0M4U9N}C}T`MH6{{$F0RM4   1. 双向数据传递   服务端选择客户端功能，客户端选择服务端功能    客户端给服务端传输数据     1. 文件传输   这里选择的是打开“test.txt”文件，在服务端传输成功后，客户端打印出来。  文件内容：    传输成功，客户端print结果：     1. SR协议   I$[~K%`0K2{PZY@L[DOWL`Y |
| 问题讨论： |
| 本次实验停等协议是GBN协议的特殊情况（窗口大小为1），所以两者的数据分组格式、程序流程图等几乎一样。   1. GBN（停等）协议数据分组格式、确认分组格式、各个域的作用   在以太网中，数据帧的MTU为1500字节，所以UDP数据报的数据部分应小于1472字节（除去IP头部20字节与UDP头的8字节），为此，定义UDP数据报的数据部分格式为：    Seq为1个字节，取值为0~255，（故序列号最多为256个）；  Data≤1024个字节，为传输的数据；  最后一个字节放入EOF0，表示结尾。  ACK数据帧定义：    由于是从服务器端到客户端的单向数据传输，因此ACK数据帧不包含任何数据，只需要将ACK发送给服务器端即可。ACK字段为一个字节，表示序列号数值。末尾放入0，表示数据结束。   1. GBN（停等）协议两端程序流程图     左图为服务端流程，右图为客户端流程   1. GBN（停等）协议典型交互过程   **服务端：**  1.开启端口访问，等待客户端发起请求。  2.收到客户请求消息后，读取请求数据的内容，构造数据包并准备发送，开启新的发送数据报端口，在新端口上向客户端进行发送数据报同时开启计时器。  3.接受客户端返回的ACK确认数据报，根据序列号判断选择滑动窗口发送帧还是重传操作。  **客户端：**  1.向服务端发送请求，然后等待服务端反馈；  2.接收到服务端数据报后进行判断是否为当前欲接收的数据报，是则接收，不是则丢弃，然后返回ACK确认。    典型交互中，出现的情况有发送的数据报丢失、超时或收到重复ACK导致重传；   1. 数据分组丢失验证模拟方法   实验中采用对某数取余，当余数为0时则视为该数据分组丢失，不返回确认ACK；  客户端对收到的数据帧个数计数，当对某数取余后余数为0时则不返回确认ACK从而模拟丢包现象；     1. 程序实现的主要类（或函数）及其主要作用   void timeoutHandler();//超时重传处理函数，滑动窗口内的数据帧都要重传  void ackHandler(char c);//收到 ack，累积确认，取数据帧的第一个字节  bool seqIsAvailable();//当前序列号curSeq是否可用  void printTips();//打印提示信息  BOOL lossInLossRatio(float lossRatio);//根据丢失率随机生成一个数字，判断是否丢失,丢失则返回 TRUE，否则返回 FALSE  void getCurTime(char\* ptime);//获取当前系统时间，结果存入 ptime 中   1. UDP编程主要特点    1. udp发送和接收没有缓冲区，发送和接收都是整包，自动保持包的边界    2. udp包的发送和接收不保证一定成功，不保证按正确顺序抵达    3. 在接收udp包时，如果接收包时给定的buffer太小的话，会发生异常，要捕获异常，相应调整buffer的大小，和给出反馈信息。    4. 如果不允许丢包的情况出现的话，要有重发机制来保证，如：每发一条信息，只有收到正确的ack的时候，才证明成功 2. 实验验证结果   见上文   1. 详细注释的源程序：   见压缩包附件 |
| 心得体会： |
| 通过本次实验，更加理解了GBN 协议和SR协议的区别，以及通信过程中，从一端传送数据报到另一端接受，返回ACK，再到一端接受ACK，必要时重传的逻辑过程，也更加了解到窗口设计的巧妙之处。 |