|  |
| --- |
| 实验目的： |
| 理解滑动窗口协议的基本原理；掌握 GBN 的工作原理；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GBN 协议，实现单向可靠数据传输（服务器到客户的数据传输）。  2) 模拟引入数据包的丢失，验证所设计协议的有效性。  3) 改进所设计的GBN 协议，支持双向数据传输；  4）将所设计的GBN 协议改进为SR 协议。 |
| 实验过程： |
| GBN协议数据分组格式：   |  |  | | --- | --- | | Seq | Data |   Seq是序列号（1-20），Data是传输的数据。  确认分组格式：   |  | | --- | | ACK |   直接返回ACK编号。  GBN协议的设计与实现  与简单停等协议相比，GBN协议事实上就是把发送窗口的长度设置为n，而接收窗口的长度仍保持为1，也就是GBN协议不接受乱序到达的数据，如果数据乱序到达，那么在接受处将会决定放弃这一部分数据；而GBN协议采用的是累积确认的方式，也就是说如果当前等待接收的数据编号为1,2,3，而收到了编号为3的ack，那么就表示这三个数据已经都正确接收。  GBN协议流程图如下所示：    图 1 传输端流程图  接收端流程图：    图 2 接收端流程图  SR协议的改进思路：  SR协议与GBN协议不同，它不仅拥有发送窗口，同时也拥有接收窗口（缓存）；在超时之后不像GBN一样从最后一个成功接收（成功收到的ACK）的下一个位置进行传输，而是将发送窗口内把所有并未接收成功的数据进行传输。  对于接收端则是成功来什么数据（且未接受过）就接收，然后返回对应的ACK，而不是单纯表示<=ACK的数据全部成功接收，以此提高效率。  而在SR协议中，最重要的就是窗口内成功接收数据的记录（缓存），我在代码中使用ack[]数组来实现，ack[i]为FALSE表示i号数据成功接收并收到了ACK，否则说明还需要继续重传/接收。  接收端收完这一数据后，将窗口内全部的缓存数据按序写入文本中即可。  主要函数的功能及实现：      GBNack确认    SRack确认    GBN重传    SR重传 |
| 实验结果： |
| 时间获取      GBN传输    SR传输 |
| 问题讨论： |
| 由于UDP协议是无连接的传输协议，无法保证可靠的数据传输，因此如果在上层使用的时候需要上层来实现可靠的数据传输的部分。而且和TCP协议不同，UDP是一种无连接的协议，因此在传输数据之前并不需要三次挥手建立连接，尽管在本次实验中一部分代码确实使用了类似于连接的方式实现，但是需要注意的是那只是逻辑上的连接，并不存在真实的连接过程。  SR协议的出现主要就是为了解决在使用GBN的过程中的一定量的资源浪费的情况，事实上SR协议、GBN协议、停等协议这三者在本质上的区别就是接收窗口和发送窗口大小的区别，至于其他的一些区别都是一些小的区别。需要注意的是因为SR协议为每一个发送窗口的数据都设置了一个计时器，每次都重传超时的部分。在实际实现的过程中每次只需要比对发送窗口最低位是否超时，如果超时则重传，如果没有超时则表示发送窗口中没有超时的数据。  SR协议中窗口大小的设置可能会出现一些错误，原因在于它在Ns+NR > 2k时会导致无法区分S的某些数据是上一轮的数据还是这一轮的数据（seq号循环重复造成的错误），因此要确保Ns+NR ≤2k。  要注意等待超时的时间：过长会使文件传输过慢；过短会导致文件明明并未丢包但是由于超时等待时间过短而导致其被判定为了丢包。还有一种解决方法：将超时的判定改为接受相同ACK的次数（仅GBN协议）。 |
| 心得体会： |
| 在本次实验中，我更深刻地体会了GBN协议和SR协议的内涵，对每一步骤以及它的核心都有了更好的理解（如GBN中的累计确认、SR中的发送和接收窗口、窗口长度等）；  同时我也进一步熟悉了socket编程中的更多细节，也很高兴能够成功实现基于UDP的可靠数据传输。 |