**《计算机网络实践》课程报告**





**TCP 的设计与实现**

**姓 名 原磊鑫 张恩赐**

**学 院 智能与计算**

**专 业 计算机科学**

**年 级 大三**

**2021 年 8 月 20 日**

1. **报告摘要**

简要介绍需要解决的具体问题、协议设计和实现，以及主要实验结果。

1. **任务分析**

分析实践任务需要解决的具体问题。

1. **协议设计**

根据任务要求设计协议。包括总体设计、数据结构设计和协议规则设计。

总体设计：根据任务分析划分功能模块、理清模块组织结构和关系等。

**数据结构设计：**详细描述所设计协议的数据结构。包括协议头部结构、主要的数据结构等。

1. **首先是TCP的报文头部结构**

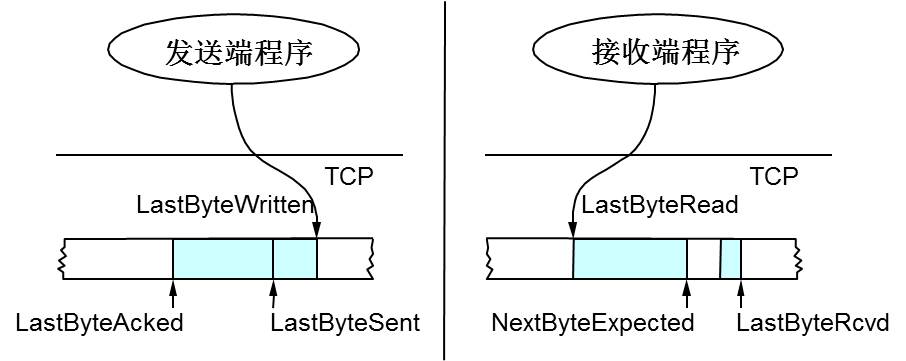
|  |  |  |
| --- | --- | --- |
| source\_port | | destination\_port |
| seq\_num | | |
| ack\_num | | |
| hlen | | plen |
| advertised\_window | | checksum |
| flags | ext | padding |

1. Source\_port：源端口
2. Destination\_port：目标端口
3. Seq\_num： 表示本报文段所发送数据的第一个字节的编号。在TCP连接中所传送的字节流的每一个字节都会按顺序编号，若出现序列号回绕，再次从0开始
4. Ack\_num：表示接收方期望收到发送方下一个报文段的第一个字节数据的编号。
5. Hlen： 表示首部长度
6. Plen： 协议地址长度
7. Advertised\_window： 定义窗口大小，可作为滑动窗口解决流量控制
8. Checksum： 检验和，用于差错控制
9. Flags： TCP的编码位，包括URG,ACK,PSH,RST,SYN,FIN.
10. Ext： 外部
11. Padding ： 该字段长度不固定，这是个用于补充头部字段使得它的长度为32位字的整数倍的一个伪字段

**2. TCP包的设计：**

TCP包是没有IP地址的，但是有源端口和目标端口，一般来说，一个TCP连接需要四个元组（实际上需要五个，但是此次实验为TCP协议，所以为四元组）（src\_ip,src\_port,dst\_ip,dst\_port）

3. TCP缓冲区设计：



接收端LastByteRead指向了TCP缓冲区中读到的位置，NextByteExpected指向的地方是收到的连续包的最后一个位置，LastByteRcved指向的是收到的包的最后一个位置。

发送端的LastByteAcked指向了被接收端Ack过的位置（表示成功发送确认），LastByteSent表示发出去了，但还没有收到成功确认的Ack，LastByteWritten指向的是上层应用正在写的地方。

4.TCP监听队列

在调用listen函数后，会为此socket创建一个用于保存半连接和已经建立连接的socket队列

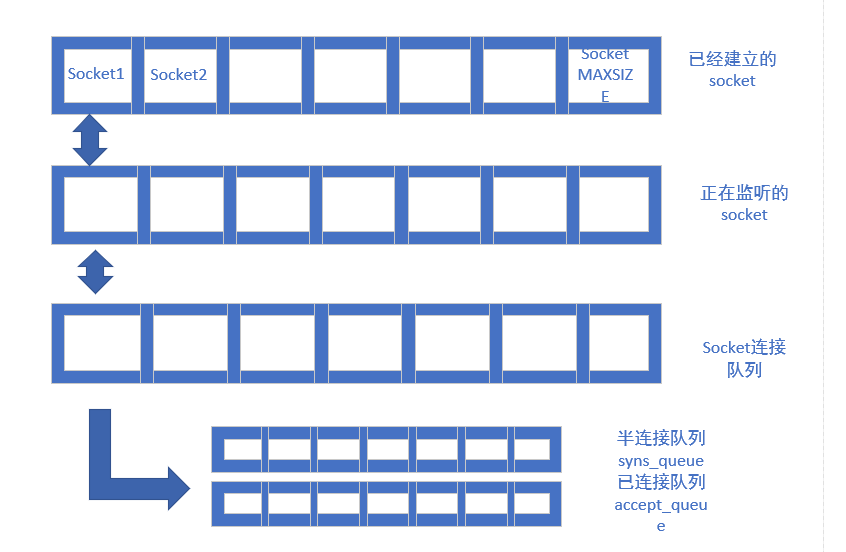
因此，需要定义一个TCP连接队列tcp\_sock\_queue,此结构体包含了

一个半连接队列

tju\_tcp\_t\* syns\_queue[SYNS\_Queue\_LEN];

一个已经连接队列

tju\_tcp\_t\* accept\_queue[ACCEPT\_Queue\_LEN];



如图所示，当一个新的数据包传入时，先计算其hash值，然后根据hash值进行一下操作

* + - 1. 如果以此hash为索引可以找到一个已经建立的socket

即：established\_socks[hashval]!=NULL

就将数据传给此socket

* + - 1. 如果1找不到，那么就在listen中找

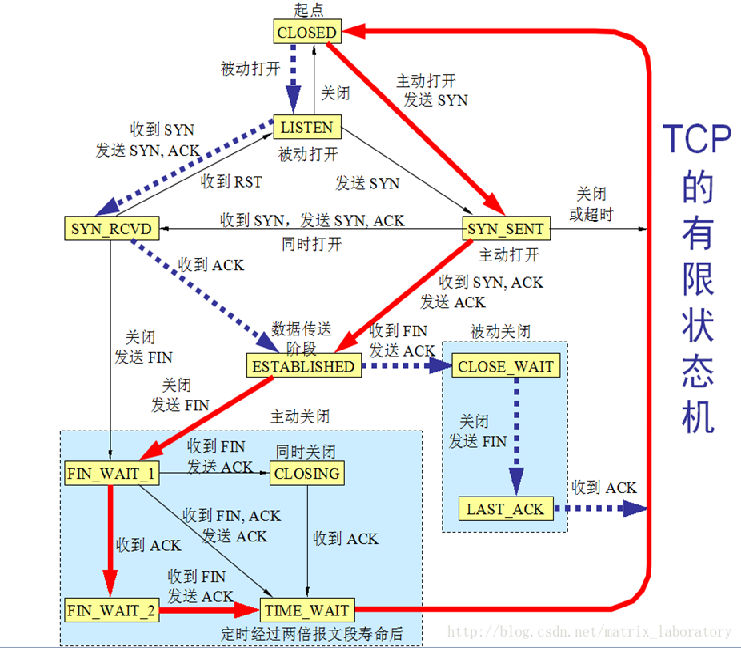
即：listen\_socks[hashval]!=NULL

就将此数据传给监听socket

* + - 1. 如果是一个合法的socket连接过程，那么1，2中就必定符合一个（而且只会符合一个，因为一个socket不会又建立又监听）
      2. 对于一个listen的socket来说，如果收到一个SYN第一次握手报文，那么就在自己对应的socket的半连接队列里创建一个新的socket（这三个大的数组都是以socket的hash为索引的）
      3. 服务器端完成连接后，会把新建立的socket从半连接队列中转到已连接队列，并将这个新socket注册到已连接socket数组中
      4. 使用accept函数时，从监听socket的已连接队列中返回一个队头socket。

**协议规则设计：**详细描述协议完成各个功能的协议规则。使用FSM图来表示每个功能模块的规则。

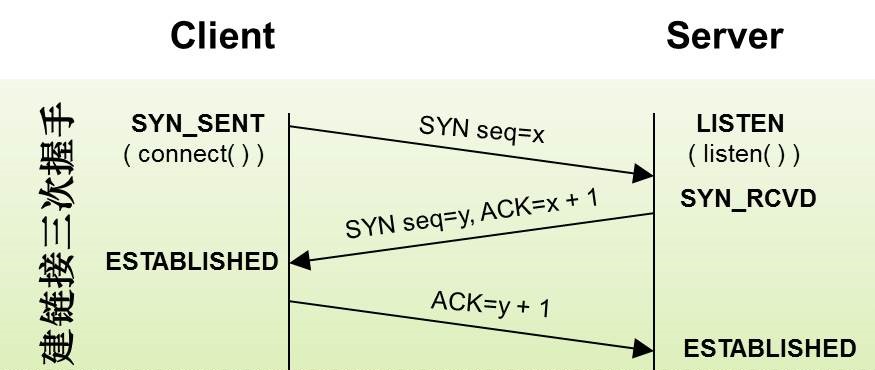
* + - 1. **实际TCP协议的状态机**

****

本次实验是在Linux系统上用UDP来实现模拟TCP协议，故相应流程应仿照TCP设计  
附TCP状态转化：



* + - 1. **TCP建立连接三次握手**



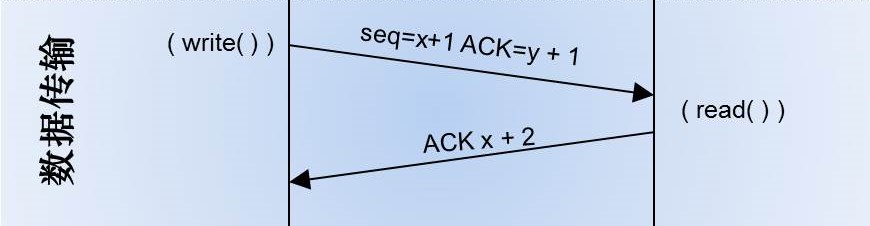
TCP使用三次握手的方式进行连接：

**第一次握手：**客户端主动发送一个TCP报文段，其中SYN字段设为1（表示建立连接请求），且包含了该报文段的初始序列号；

**第二次握手：**服务端在收到客户端的报文段后，发送自己的TCP报文段回应，其中SYN字段和ACK字段均设为1,该报文段的序列号为它自己的初始序列号，确认号为第一次握手中获得的序列号+1；

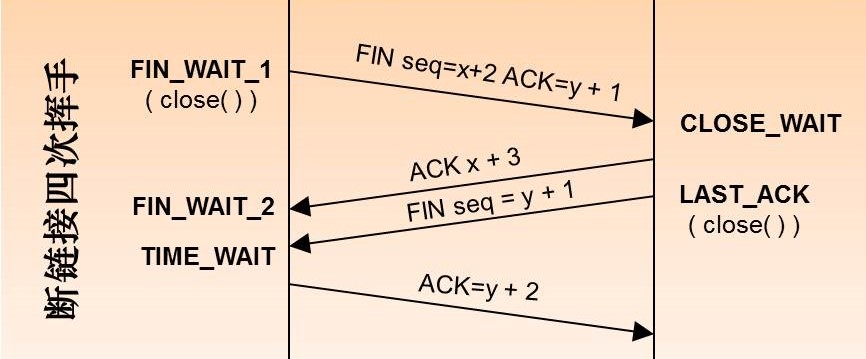
**第三次握手：**客户端在接收到服务端的回应报文段后，响应服务端，其中的ACK字段设为1。序列号为y+1。

* + - 1. **TCP连接协议**



数据传输如图所示即可

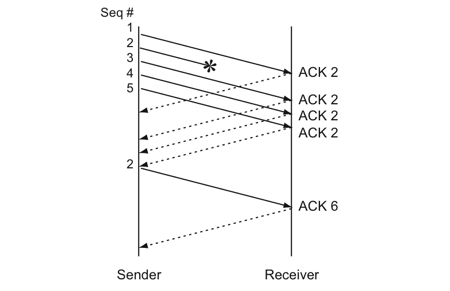
* + - 1. **TCP关闭连接**



1. 关闭的发起端发送的报文段中中FIN和ACK字段为1，要求“单方面”关闭连接
2. 接收端回应接收到报文，但是可能被动关闭方可能数据并未完全传输完成，所以继续传输数据给发起段，发起段确认接收数据
3. 然后被动关闭方在发送“关闭报文”（FIN字段为1），之后客户端发送响应，TCP连接关闭
4. 在此过程之中，TCP连接处于半关闭状态，即连接的一方已经关闭，另一方任然传输数据直到它关闭为止。
   * + 1. **TCP重传机制**

TCP保证所有的数据包都可以到达；

这里使用快速重传机制和Fast Retranmit的算法

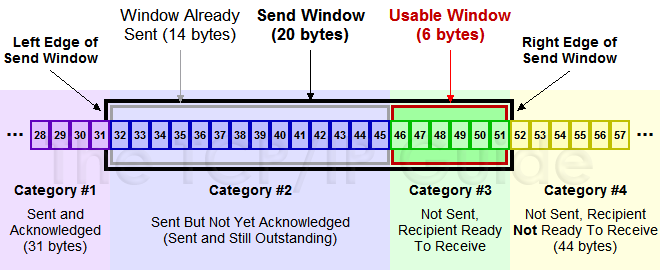


在出现乱序数据包的情况下，如上图所示：如果发送方发出了1，2，3，4，5份数据，第一份先到送了，于是就ack回2，结果2因为某些原因没收到，3到达了，于是还是ack回2， 后面的4和5都到了，但是还是ack回2，因为2还是没有收到，于是发送端收到了三个ack=2的确认，知道了2还没有到，于是就马上重转2。然后，接收 端收到了2，此时因为3，4，5都收到了，于是ack回6。

* + - 1. **滑动窗口设计**

依靠TCP报头中的advertised\_window字段来使接收端向发送端传输自己还有多少缓冲区可以接收数据。从而发送端就可以根据这个接收端的处理能力来发送数据，实现可靠传输。

发送方的滑动窗口示意图：



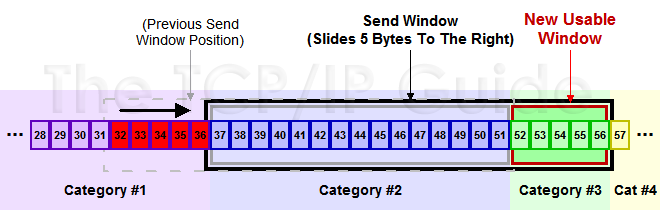
#1已收到ack确认的数据。

#2发还没收到ack的。

#3在窗口中还没有发出的（接收方还有空间）。

#4窗口以外的数据（接收方没空间）

滑动后的示意图（接受了36的ACK，并且发出了46-51的字节）

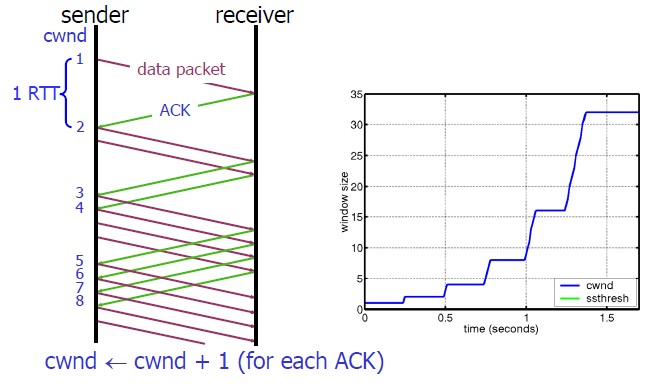


* + - 1. **拥塞避免算法—congestion avoidance**

TCP在传输数据时首先采用慢启动：

1. 连接建好的开始先初始化cwnd = 1;表明可以传一个MSS大小的数据
2. 每当收到一个ACK，cwnd++,呈线性上升；
3. 每当过了一个RTT，cwnd = cwnd \* 2 ，呈指数上升
4. 当ssthresh（窗口上限），当cwnd >= ssthresh时，就会进入拥塞避免算法

示意图如下：



拥塞避免算法：

（1）sshthresh = cwnd /2

（2）cwnd 重置为 1

（3）进入慢启动过程

1. **协议实现**

详细描述功能实现的细节。主要功能模块使用流程图或者伪代码来辅助说明。禁止贴源码。

1. **实验结果及分析**

测试所实现协议的功能和性能，并对性能结果进行分析。需要针对考察点逐一展开。

1. **个人总结**

总结自己在实践过程中遇到的各类问题、困难以及解决过程中的收获，对实践内容等方面的体会与建议。