### 实验报告

1. JAVA Applet演示
2. GBN协议：

演示中的发送窗口大小为5，所以发送端同时最多有5个未确认分组。GBN的接受窗口大小为1，即只有期望的序号的分组会被接受。所以一个分组丢失，后面发送的分组都会被丢弃，发送端窗口不能移动；当定时器超时时，窗口内的所有分组重发。如果一个确认丢失则并不会造成影响，因为后面的累计确认包含了前面的。

1. SR协议：

演示中的发送窗口和接受窗口大小都为5，所以发送端同时最多有5个未确认分组。接受窗口内收到的分组都会被接收，并发送相应的ACK（而不是累计）。如果一个分组丢失，发送窗口接收不到响应的ACK，发送窗口会被卡住，超时之后只有这个分组重发。在收到这个ACK后，发送窗口会一直移动到最后一个ACK以后。如果一个ACK丢失，则对发送方对影响和分组在发送时丢失是一样的，而接收方窗口已经移动，不会重复接收。

1. TCP流量控制

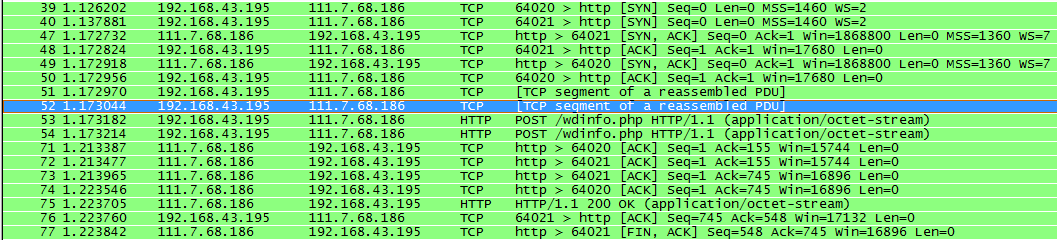
演示中发送方想向接收方发送4K大小对文件，但是两者发送接收缓存都只有2K，所以先发送2K，到达接收方缓存，接收方发ACK2048和WIN=0，说明缓存已满。发送方接收到后，会发送1B的报文，接收方接收后仍回ACK2048和WIN=0，这个过程循环以保持联络，直到接收进程取走缓存，发送WIN=2048，发送方的缓存中剩余的2K被发送，收到相应ACK后不再通信，因为传输已经完成。如果发送文件小于双方缓存，则一次就可以发送完。

1. 利用Ethereal研究TCP传输过程

报告要求：①利用所捕获的分组，分析本次上传的整个流程，在分析中应包括：客户端主机的IP地址、服务器的IP地址、TCP端口号、各SYN报文段的序号、SYNACK报文段、含POST命令的TCP报文段序号等关键数据（分析过程中请适当抓图并划线标注之）；②评估主机与服务器之间TCP连接的性能。

注：为了避免多次操作，本次试验可使用“SAVE”保存完整捕获分组。

实验中是修改了github头像。



分析：

从这一系列分组中可以知道，本机的ip为192.168.43.195，服务器ip为111.7.68.195，端口为80（http）；不知道是不是因为我在上传时多点了一次，所以有两个并行的，历程和传输的数据完全相同的连接，一个端口号为64020，一个为64021，我选择忽略64020的。可以看到，我的主机与服务器进行了三次握手（40，47，48），第一次是本机发送SYN请求连接，第二次是服务器发送ACK和SYN表示同意建立连接，第三次是本机发送ACK表示连接建立成功。之后传输开始，使用的是POST，头和数据体被分开传送（51是头，大小为154 bytes，53是体，大小为590 bytes）；之后服务器收到两个报文，发送ACK=155和ACK=745（72，73），之后又发送状态码为200的http报文（75），表示成功处理了请求，本机发送ACK表示它知道了（76），服务器收到ACK后发送FIN关闭连接（77），本机发送FIN和ACK（81）同意关闭，服务器发送ACK（85），连接关闭。

51的数据：

0000 3c fa 43 ad db 84 60 36 dd 7e 54 73 08 00 45 00 <.C...`6 .~Ts..E.

0010 00 c2 30 dd 40 00 40 06 69 2c c0 a8 2b c3 6f 07 ..0.@.@. i,..+.o.

0020 44 ba fa 15 00 50 7c 5c 1c cb 0b 46 51 43 50 18 D....P|\ ...FQCP.

0030 11 44 89 9f 00 00 50 4f 53 54 20 2f 77 64 69 6e .D....PO ST /wdin

0040 66 6f 2e 70 68 70 20 48 54 54 50 2f 31 2e 31 0d fo.php H TTP/1.1.

0050 0a 43 6f 6e 74 65 6e 74 2d 54 79 70 65 3a 20 61 .Content -Type: a

0060 70 70 6c 69 63 61 74 69 6f 6e 2f 6f 63 74 65 74 pplicati on/octet

0070 2d 73 74 72 65 61 6d 0d 0a 55 73 65 72 2d 41 67 -stream. .User-Ag

0080 65 6e 74 3a 20 50 6f 73 74 0d 0a 48 6f 73 74 3a ent: Pos t..Host:

0090 20 71 75 72 6c 2e 66 2e 33 36 30 2e 63 6e 0d 0a qurl.f. 360.cn..

00a0 43 6f 6e 74 65 6e 74 2d 4c 65 6e 67 74 68 3a 20 Content- Length:

00b0 35 39 30 0d 0a 43 61 63 68 65 2d 43 6f 6e 74 72 590..Cac he-Contr

00c0 6f 6c 3a 20 6e 6f 2d 63 61 63 68 65 0d 0a 0d 0a ol: no-c ache....

标明了HTTP版本和内容类型等信息

POST中的数据：是看不懂的乱码，应该是图片的格式。

POST中的分段信息（截错图了，Frame应该是51和53）：

捕获(2)

评估TCP连接的性能：

TCP连接需要进行三次握手，能够保证连接的一定的正确性（虽然不能完全保证）。在建立连接之后用POST传递数据，分成头和体分别传输，服务器收到后发两个ACK以及请求处理成功的通知，等待本机知道后，发送FIN请求关闭连接，与连接建立的过程相似，关闭也需要三次。总体来说连接的建立和关闭都比较可靠，而数据传输依赖ACK的对应关系和检验和而比较可靠。虽然互相发送报文的次数较多，但是这对可靠TCP传输是必要的。

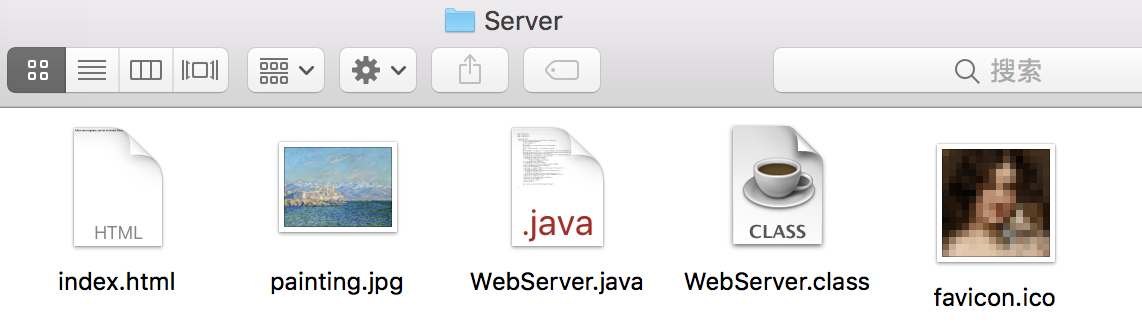
1. Socket编程

未改动的程序运行之后

第一次

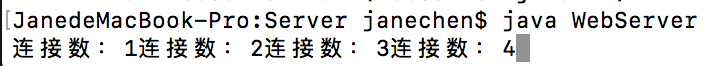
第二次： ：

同时可以看到终端上的进程已经退出。



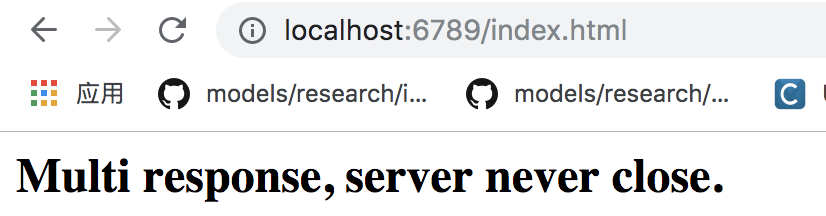
**改动后：**当连接了四个页面，java程序所在文件夹内容如上图时：

服务端显示：

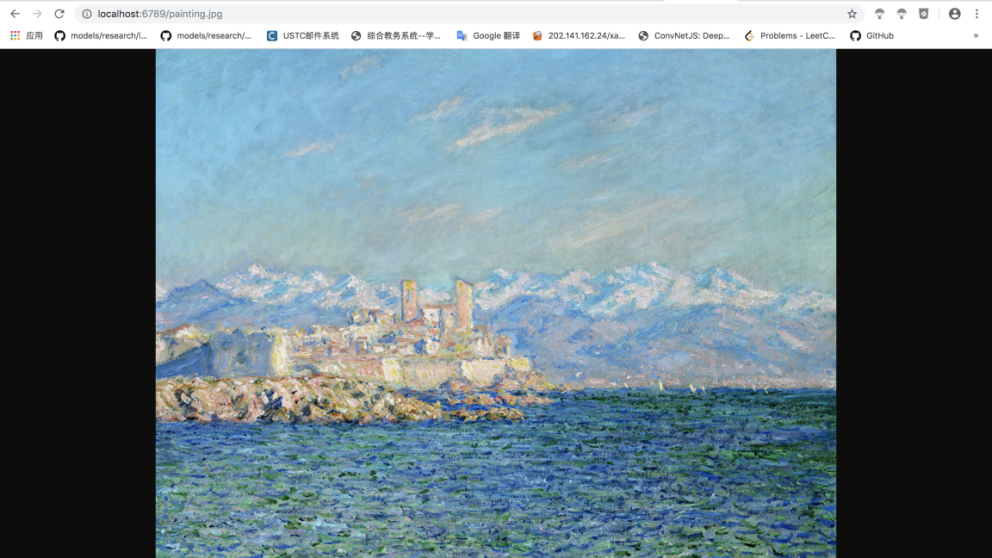


浏览器显示：

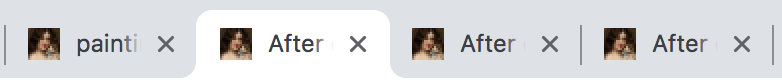
请求html



请求图片



由于crome浏览器会请求图标，所以我只好做了一个图标。这个图标只需要连接一次，只后可能是被浏览器保存了，不需要再请求。



**源代码：**

**import** java.io.\*;

**import** java.net.\*;

**import** java.util.\*;

**class** WebServer{

**public** **static** **void** main(String args[])**throws** Exception{

String requestMessageLine;

String fileName="";

**int** i = 0;

ServerSocket listenSocket=**new** ServerSocket(6789);

//listenSocket永远守候在6789端口

**while**(**true**) { //只要主机的服务程序不结束，就一直可以建立新的连接

Socket connectionSocket = listenSocket.accept();

i++; //连接计数

BufferedReader inFromClient = **new** BufferedReader(**new** InputStreamReader(connectionSocket.getInputStream()));

DataOutputStream outToClient = **new** DataOutputStream(connectionSocket.getOutputStream());

requestMessageLine=inFromClient.readLine();

StringTokenizer tokenizedLine = **new** StringTokenizer(requestMessageLine);

**if**(tokenizedLine.nextToken().equals("GET")){

fileName=tokenizedLine.nextToken();

**if**(fileName.startsWith("/")==**true**)

fileName=fileName.substring(1);

File file=**new** File(fileName);

**int** numOfBytes=(**int**)file.length();

FileInputStream inFile=**new** FileInputStream(fileName);

**byte**[] fileInBytes=**new** **byte**[numOfBytes];

inFile.read(fileInBytes);

outToClient.writeBytes("HTTP/1.0 200 Document Follows\r\n");

**if**(fileName.endsWith(".jpg"))

  outToClient.writeBytes("Content-Type:image/jpeg\r\n");

**if**(fileName.endsWith(".gif"))

  outToClient.writeBytes("Content-Type:image/gif\r\n");

outToClient.writeBytes("Content-Length:"+numOfBytes+"\r\n");

outToClient.writeBytes("\r\n");

outToClient.write(fileInBytes,0,numOfBytes);

System.***out***.print("连接数："+i); //服务端显示连接数

connectionSocket.close();

inFile.close(); //为了下一次还能用这个指针打开，要把现在打开的关掉

}

**else** System.***out***.println("Bad Request Message");

}

  }

}