计算机网络课程实验三

1.1 实验目的
1.2 实验内容
1.2.1 JAVA Applet 演示
1.2.1.1 Go-back-N 回退N步协议
1.2.1.2 Selective Repeat 选择重传协议
1.2.1.3 Flow Control 流量控制协议
1.2.2 利用 Ethereal 研究 TCP 传输过程
1.2.3 Socket 编程
1.2.3.1 构造一个简单的Web服务器
1.2.3.2 可以响应多个请求的 WebServer.java 程序(串行依次响应实现)
1.2.3.3 可以响应多个请求的 WebServer.java 程序(多线程实现)

1.1 实验目的

本次实验的主要目的是让我们对网络原理有更深刻和更直观的认识；掌握网络基本技术如组网、截获/分析数据包、网络编程等，为以后更深入的学习和利用计算机网络打下一个好的基础。

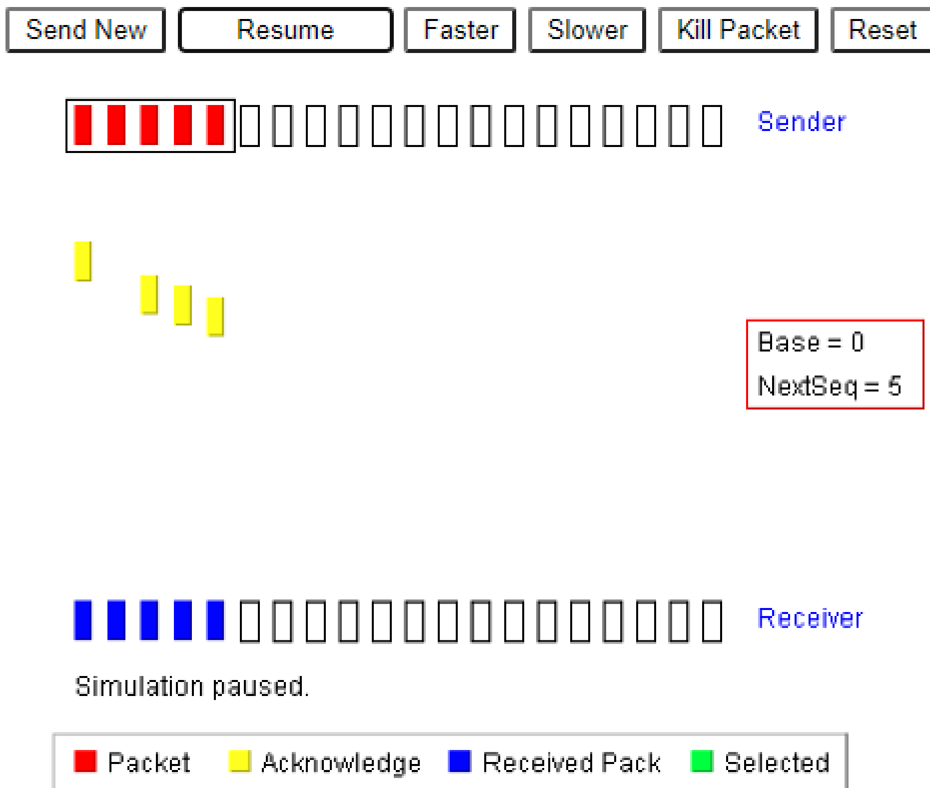
1.2 实验内容

1.2.1 JAVA Applet 演示

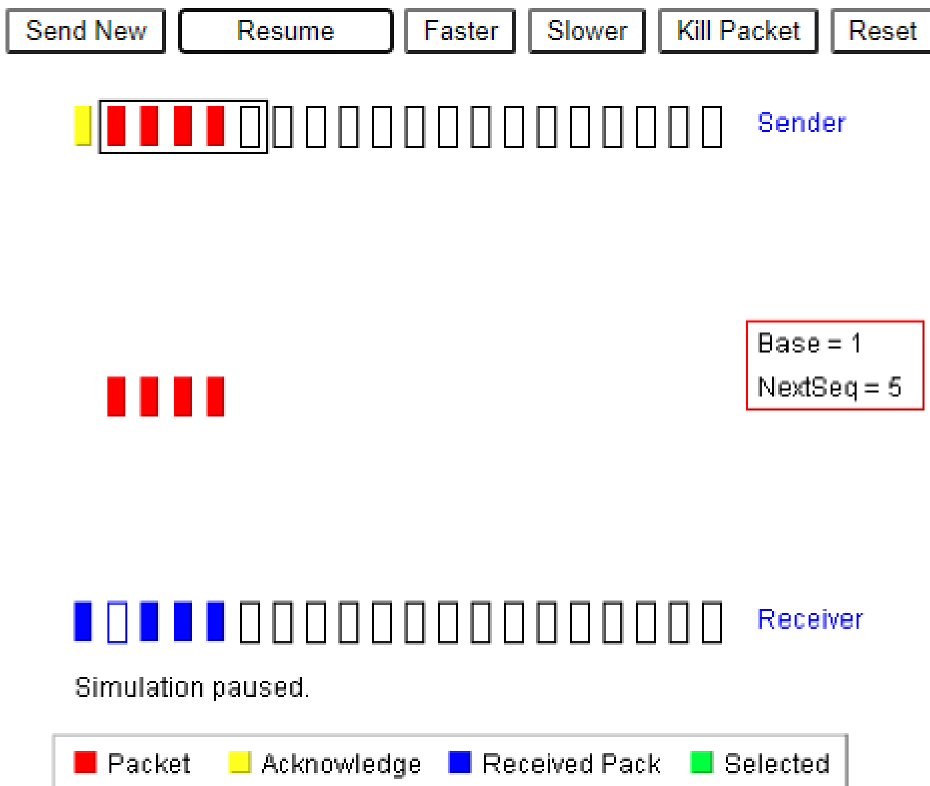
1.2.1.1 Go-back-N 回退N步协议

回退N步协议允许发送方同时发送N个未经确认的分组，在这个演示实验中发送窗口大小为5，红色代表发送但未经确认的分组，黄色代表ACK和收到确认的分组，蓝色代表接收方接收到的分组，发送窗口内的白色方块代表可以发送的分组，发送窗口外的白色方块代表还不能使用的分组，另外，定义base为最早的未经确认的分组序号，nextseq为发送方下一个准备发送的分组序号。

根据GBN协议，对序号为n的分组采取累积确认的方式，表明接收方已经正确收到序号为n的以前且包括n在内的所有分组。在下面这种情况中，发送方连续发送5个分组，接收方按序收到并返回5个ACK，但是途中第二个分组的ACK丢失，接收方收到了剩下的分组确认，可以据此判断分组已经收到了所有5个分组，发送窗口向前移动，base将变为5，此时可以继续连续发送5个未经确认的分组。



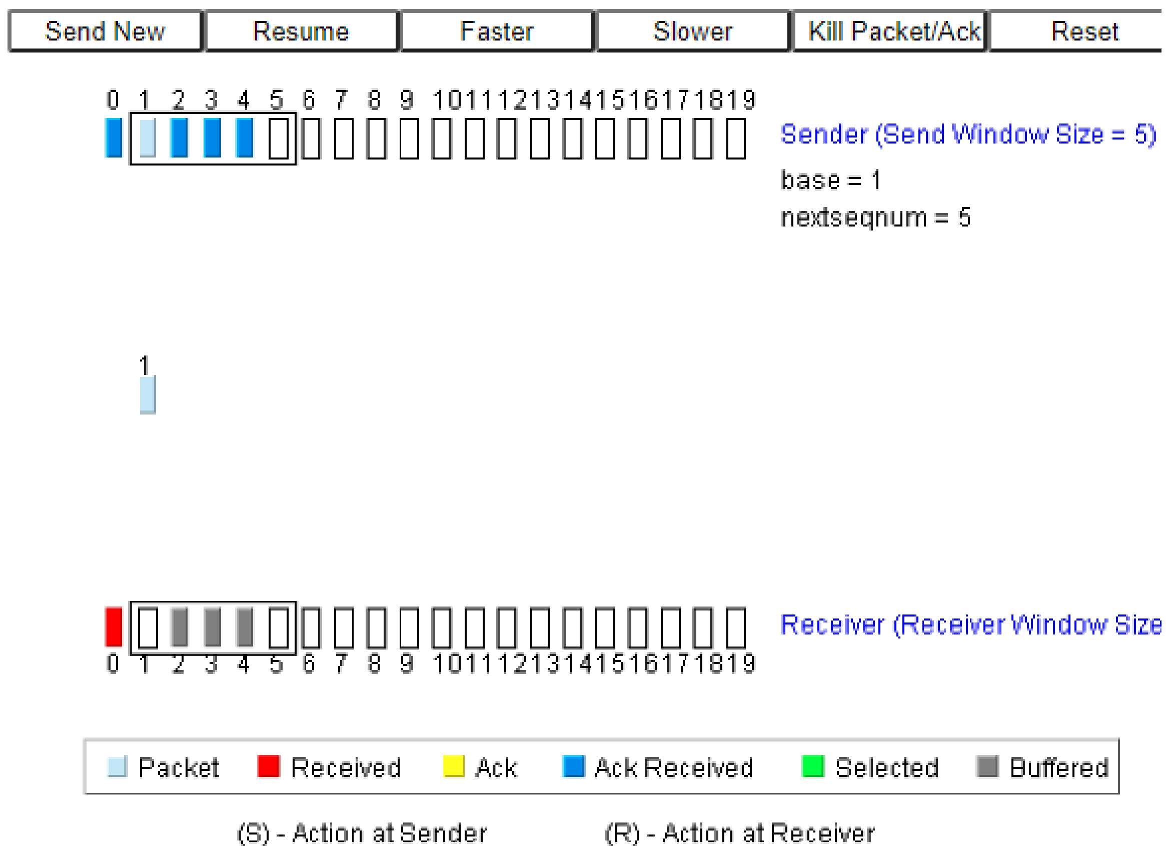
下面这种情况里，发送方连续发送5个分组，但是途中第二个分组丢失，接收方只返回第一个分组的ACK，并丢弃剩下失序到达的分组，发送方为最早发送但未经确认的分组即第二个分组设置了一个定时器，当出现超时时，发送方将重传所有已发送但还未确认的分组，如下图所示，当收到这些分组的ACK后，发送窗口向前移动，从而可以继续传送新的分组。



1.2.1.2 Selective Repeat 选择重传协议

选择重传协议允许发送方同时发送N个未经确认的分组，在这个演示实验中发送窗口大小为5，浅蓝色代表发送但未经确认的分组，黄色代表ACK，蓝色代表收到确认的分组，红色代表接收方确认收到并交付给上层的分组，灰色代表失序到达并被接收方缓存的分组，发送窗口内的白色方块代表可以发送的分组，发送窗口外的白色方块代表还不能使用的分组，接收窗口内的白色方块代表期待收到的分组，另外，定义base为最早的未经确认的分组序号，nextseq为发送方下一个准备发送的分组序号。

根据SR协议，失序的分组将会被缓存直到所有序号更小的分组皆被收到为止，然后将它们一起按序交付给上层。在下面这种情况里，序号为1的分组在发送时丢失，接收方收到了序号为2、3、4的分组并将它们缓存，发送方为未经确认的分组设置了一个定时器，当判断超时后，发送方单独重传这个分组，如下图所示，当接收方收到这个分组后，这个分组连同分组2、3、4将一起交付给上层，然后发送分组1的ACK，随后接收窗口向前移动。



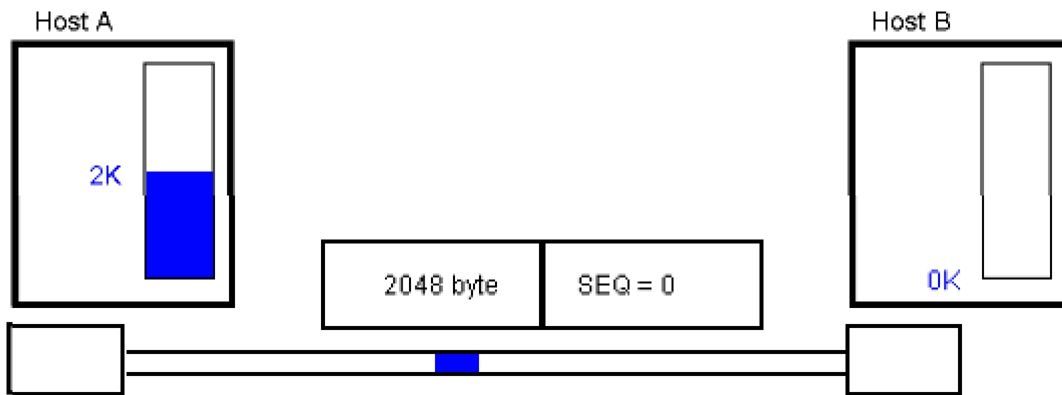
1.2.1.3 Flow Control 流量控制协议

本次小程序形象地展示了流量控制协议。一条TCP连接为每一侧主机都设置了一个接收缓存，TCP连接收到按序到达的字节后将其放入接收缓存，但应用程序可能不能及时地读取数据，为了消除接收方缓存溢出的可能性，TCP 为它的应用程序提供了流量控制协议，在发送方有一个接收窗口的变量来提供流量控制服务，表示接收方还有多少可用的缓存空间，接收方通过将当前接收窗口的值放入返回发送方的报文从而告知发送方接收窗口的大小，当接收窗口为0时，发送方会发生只有一个字节的报文段，这些报文段会被接收方确认，缓存清空后继续发送新的数据。下面给出了一个流量控制协议的实例，文件大小为4K字节，发送接收缓存区大小2K字节，接收方随机从缓存区读取一块大小为2K字节的数据，图中还给出了发送报文的大小和字节序号、响应报文中ACK值和窗口大小这些信息。

Start | Stop | Resume | Reset

File size: 4 Kbytes

Buffer Size: 2 Kbytes



Send Buffer : 2K

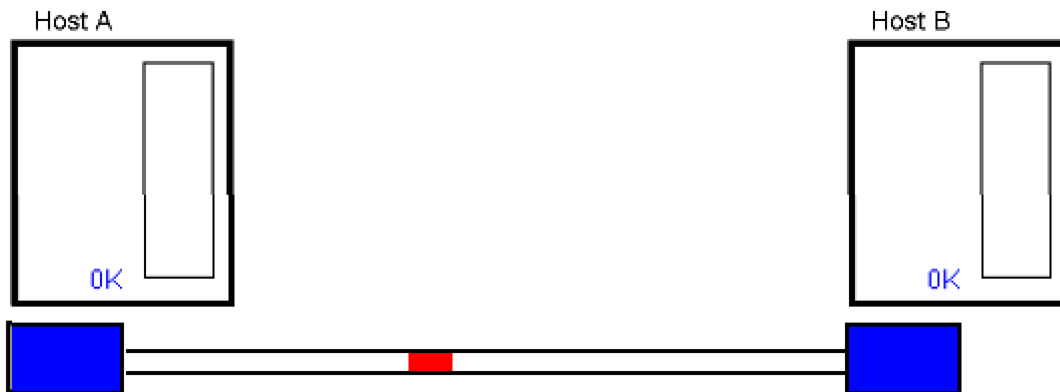
Receive Buffer : 2K

Rcv Buffer Size : 2K

Start | Stop | Resume | Reset

File size: 4 Kbytes

Buffer Size: 2 Kbytes



Send Buffer : 2K

ACK = 2048 | WIN = 0

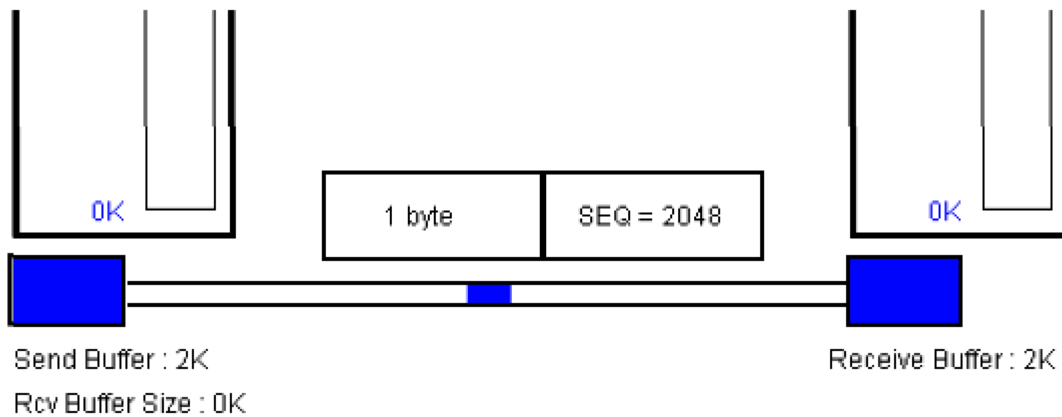
Receive Buffer : 2K

Rcv Buffer Size : 2K

Start | Stop | Resume | Reset

File size: 4 Kbytes

Buffer Size: 2 Kbytes



1.2.2 利用 Ethereal 研究 TCP 传输过程

这次实验我利用教材的配套实验来完成用 Ethereal 研究 TCP 传输过程，操作过程如下：

1. 访问https://gaia.cs.umass.edu/ethereal-labs/alice.txt，将alice.txt文件保存至本地。
2. 访问https://gaia.cs.umass.edu/ethereal-labs/TCP-ethereal-file1.html，界面如下

Upload page for TCP Ethereal Lab
Computer Networking: A Top Down Approach Featuring the Internet, 3rd edition
Copyright 2004 J.F. Kurose and K.W. Ross, All Rights Reserved

If you have followed the instructions for the TCP Ethereal Lab, you have *already* downloaded an ASCII copy of Alice and Wonderland from http://gaia.cs.umass.edu/ethereal-labs/alice.txt and you also *already* have the Ethereal packet sniffer running and capturing packets on your computer.

Click on the Browse button below to select the directory/file name for the copy of alice.txt that is stored on your computer.

选择文件 alice.txt

Once you have selected the file, click on the "Upload alice.txt file" button below. This will cause your browser to send a copy of alice.txt over an HTTP connection (using TCP) to the web server at gaia.cs.umass.edu. After clicking on the button, wait until a short message is displayed indicating the the upload is complete. Then stop your Ethereal packet sniffer - you're ready to begin analyzing the TCP transfer of alice.txt from your computer to gaia.cs.umass.edu!!

Upload alice.txt file

3. 开始Ethereal分组俘获。
4. 找到alice.txt文件，点击Upload alice.txt file
5. 停止Ethereal分组俘获。
6. 在筛选框中输入“tcp”，可以看到主机和服务器之间传输的一系列的tcp和http报文。

下图为抓到的包，可以看到TCP连接的三次握手过程，主机的IP为192.168.43.244，服务器的IP地址为128.119.245.12，从报文可以看到主机的端口号为51982，服务器的端口号为http80号端口，主机首先向服务器发送一个SYN报文段，其SYN比特置为1，初始序号为0，该报文段不包含应用层数据所以长度为0，服务器收到这个SYN报文段后，返回一个SYNACK报文段表示同意建立连接，该报文段的初始序号为0，ACK=1，SYN比特置为1,还可以看到最大报文段长度为1360字节，在收到SYNACK报文段后，客户还要向服务器发送另一个报文段对服务器允许连接的报文段进行确认，该报文段序号等于1，ACK等于1，SYN比特置为0。至此TCP的三次握手完成。当然，也可以看到主机与服务器建立了不止一个TCP连接，如下面主机渡口51983也与服务器建立了一个TCP连接。之后主机便向服务器开始传递报文，下面截取了一段报文，可见主机不断向服务器发送分组，报文序号不断增加，服务器也相应地返回ACK。

```
(Untitled) - Ethereal
File  Edit  View  Go  Capture  Analyze  Statistics  Help
Filter:                                  Expression...  Clear  Apply

No.  Time      Source           Destination      Protocol Info
 12 0.436474  192.168.43.244   52.139.153.205   TCP    51944 > https [ACK] Seq=1 Ack=1 Win=1022 Len=0
 13 1.361662  192.168.43.244   128.119.245.12   TCP    51982 > http [SYN] Seq=0 Len=0 MSS=1460 WS=8
 14 1.611125  192.168.43.244   128.119.245.12   TCP    51983 > http [SYN] Seq=0 Len=0 MSS=1460 WS=8
 15 1.683418  128.119.245.12   192.168.43.244   TCP    http > 51982 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1360 WS=7
 16 1.683509  192.168.43.244   128.119.245.12   TCP    51982 > http [ACK] Seq=1 Ack=1 Win=66560 Len=0
 17 1.684077  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
 18 1.684317  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
 19 1.684326  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
 20 1.684337  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
 21 1.684390  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
 22 1.684393  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
 23 1.684396  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
 24 1.684400  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
 25 1.684404  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
 26 1.684407  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
 27 1.905063  128.119.245.12   192.168.43.244   TCP    http > 51983 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1360 WS=7
 28 1.905341  192.168.43.244   128.119.245.12   TCP    51983 > http [ACK] Seq=1 Ack=1 Win=66560 Len=0
 29 1.959424  128.119.245.12   192.168.43.244   TCP    http > 51982 [ACK] Seq=1 Ack=653 Win=30592 Len=0
 30 1.959590  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
 31 1.973609  128.119.245.12   192.168.43.244   TCP    http > 51982 [ACK] Seq=1 Ack=2013 Win=33536 Len=0
 32 1.973746  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
 33 1.973769  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
 34 1.973807  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
 35 1.974152  128.119.245.12   192.168.43.244   TCP    http > 51982 [ACK] Seq=1 Ack=4733 Win=39296 Len=0
 36 1.974279  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
 37 1.974313  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
 38 1.974333  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
 39 1.974377  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]

⊞ Frame 152 (1414 bytes on wire, 1414 bytes captured)
⊞ Ethernet II, Src: 3c:6a:a7:1a:95:4b (3c:6a:a7:1a:95:4b), Dst: e8:3f:67:50:44:90 (e8:3f:67:50:44:90)
⊞ Internet Protocol, Src: 192.168.43.244 (192.168.43.244), Dst: 128.119.245.12 (128.119.245.12)
⊟ Transmission Control Protocol, Src Port: 51982 (51982), Dst Port: http (80), Seq: 120985, Ack: 1, Len: 1360
    Source port: 51982 (51982)
    Destination port: http (80)
    Sequence number: 120985    (relative sequence number)
    [Next sequence number: 122345    (relative sequence number)]
    Acknowledgement number: 1    (relative ack number)
    Header length: 20 bytes
  ⊞ Flags: 0x0010 (ACK)
    Window size: 66560 (scaled)
    Checksum: 0x1b5e [correct]
    [Reassembled PDU in frame: 196]
    TCP segment data (1360 bytes)
```

下图为包含POST命令的HTTP报文，主机向服务器请求了一个http://gaia.cs.umass.edu/ethereal-labs/lab3-1-reply.htm的web页面，版本为HTTP/1.1，报文段序号152265。

```
(Untitled) - Ethereal
File  Edit  View  Go  Capture  Analyze  Statistics  Help
Filter:                                  Expression...  Clear  Apply

No.  Time      Source           Destination      Protocol Info
170 2.562575  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
171 2.562581  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
172 2.564279  128.119.245.12   192.168.43.244   TCP    http > 51982 [ACK] Seq=1 Ack=80185 Win=183296 Len=0
173 2.564288  128.119.245.12   192.168.43.244   TCP    http > 51982 [ACK] Seq=1 Ack=81545 Win=182528 Len=0
174 2.564292  128.119.245.12   192.168.43.244   TCP    http > 51982 [ACK] Seq=1 Ack=82905 Win=183296 Len=0
175 2.564294  128.119.245.12   192.168.43.244   TCP    http > 51982 [ACK] Seq=1 Ack=84265 Win=183296 Len=0
176 2.564296  128.119.245.12   192.168.43.244   TCP    http > 51982 [ACK] Seq=1 Ack=85625 Win=183296 Len=0
177 2.564299  128.119.245.12   192.168.43.244   TCP    http > 51982 [ACK] Seq=1 Ack=86985 Win=182528 Len=0
178 2.564396  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
179 2.564409  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
180 2.564422  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
181 2.564429  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
182 2.564434  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
183 2.564441  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
184 2.564448  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
185 2.580335  52.114.128.75    192.168.43.244   TCP    https > 51935 [FIN, ACK] Seq=0 Ack=1 Win=2050 Len=0
186 2.580471  192.168.43.244   52.114.128.75    TCP    51935 > https [ACK] Seq=1 Ack=1 Win=1022 Len=0
187 2.586227  52.114.128.75    192.168.43.244   TCP    https > 51984 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1360 WS=8
188 2.586452  192.168.43.244   52.114.128.75    TCP    51984 > https [ACK] Seq=1 Ack=1 Win=262144 Len=0
189 2.587070  192.168.43.244   52.114.128.75    SSL    Continuation Data
190 2.801907  128.119.245.12   192.168.43.244   TCP    http > 51982 [ACK] Seq=1 Ack=91065 Win=182528 Len=0
191 2.802046  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
192 2.802065  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
193 2.802074  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
194 2.802081  192.168.43.244   128.119.245.12   TCP    [TCP segment of a reassembled PDU]
195 2.802872  128.119.245.12   192.168.43.244   TCP    http > 51982 [ACK] Seq=1 Ack=92425 Win=186112 Len=0
196 2.802995  192.168.43.244   128.119.245.12   HTTP   POST /ethereal-labs/lab3-1-reply.htm HTTP/1.1 (text/plain)
197 2.806869  128.119.245.12   192.168.43.244   TCP    http > 51982 [ACK] Seq=1 Ack=93785 Win=189056 Len=0
198 2.806880  128.119.245.12   192.168.43.244   TCP    http > 51982 [ACK] Seq=1 Ack=95145 Win=192000 Len=0
199 2.806887  128.119.245.12   192.168.43.244   TCP    http > 51982 [ACK] Seq=1 Ack=96505 Win=194944 Len=0
200 2.828455  128.119.245.12   192.168.43.244   TCP    http > 51982 [ACK] Seq=1 Ack=97865 Win=197888 Len=0
201 2.828817  128.119.245.12   192.168.43.244   TCP    http > 51982 [ACK] Seq=1 Ack=99225 Win=200704 Len=0
```

```
Transmission Control Protocol, Src Port: 51982 (51982), Dst Port: http (80), Seq: 152265, Ack: 1, Len: 709
  Source port: 51982 (51982)
  Destination port: http (80)
  Sequence number: 152265    (relative sequence number)
  [Next sequence number: 152974    (relative sequence number)]
  Acknowledgement number: 1    (relative ack number)
  Header length: 20 bytes
  Flags: 0x0018 (PSH, ACK)
  Window size: 66560 (scaled)
  Checksum: 0x7937 [correct]
  TCP segment data (709 bytes)
[Reassembled TCP Segments (152973 bytes): #17(652), #18(1360), #19(1360), #20(1360), #21(1360), #22(1360), #23(1360), #24(1360), #2
Hypertext Transfer Protocol
  POST /ethereal-labs/lab3-1-reply.htm HTTP/1.1\r\n
  Host: gaia.cs.umass.edu\r\n
  Connection: keep-alive\r\n
  Content-Length: 152321\r\n
  Cache-Control: max-age=0\r\n
  Upgrade-Insecure-Requests: 1\r\n
  Origin: null\r\n
  Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryyDS3naNQ71gzZ1BJ\r\n
  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36 Ec
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=
  Accept-Encoding: gzip, deflate\r\n
  Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6\r\n
  \r\n
```

在浏览器可以看到如下页面，表示文件上传成功。观察主机与服务器之间的TCP报文段，可以看到传输采取了滑动窗口协议，在一个RTT内，主机连续发送了多个分组，提高了TCP连接的性能和传输效率。

Congratulations!

You've now transferred a copy of alice.txt ffrom your computer to gaia.cs.umass.edu. You should now stop Ethereal packet capture. It's time to start analyzing the captured Ethereal packets!

1.2.3 Socket 编程

1.2.3.1 构造一个简单的Web服务器

在同一个文件夹下新建名为 WebServer.java 和 index.html 的文件，将附录中的内容写入文件当中，编译WebServer.java文件，然后运行编译成功的WebServer.class文件，在浏览器的地址栏中输入http://localhost:6789/index.html，即可成功显示出index.html文件，但如果重新打开一个窗口，在地址栏中输入相同的地址，会显示无法找到该网页，说明这段代码只能响应单个请求。

1.2.3.2 可以响应多个请求的 WebServer.java 程序(串行依次响应实现)

我编写的可以响应多个请求的 WebServer.java 程序源代码如下：

```
import java.io.*;
import java.net.*;
import java.util.*;

class WebServer{
    public static int connection(Socket connectionSocket) throws Exception{
        String requestMessageLine;
        String fileName="";

        BufferedReader inFromClient = new BufferedReader(new
InputStreamReader(connectionSocket.getInputStream())); //定义一个类型为
BufferedReader的inFromUser流对象
        DataOutputStream outToClient = new
DataOutputStream(connectionSocket.getOutputStream()); //创建连接到套接字的流对象，提供
到套接字的输出
        requestMessageLine=inFromClient.readLine(); //将用户的输入读入字符串
        StringTokenizer tokenizedLine = new StringTokenizer(requestMessageLine);
//将字符串标记化

        if(tokenizedLine.nextToken().equals("GET")){ //判断是否为一个请求
            fileName=tokenizedLine.nextToken();
        if(fileName.startsWith("/")==true)
            fileName=fileName.substring(1); //找到请求文件的路径
            String a = new String("favicon.ico"); //请求的如果是图标，忽略这个请求
            if(fileName.equals(a))
```

```
                connectionSocket.close();
            else{
                File file = new File(fileName); //打开文件
                int numOfBytes = (int)file.length(); //统计长度
                FileInputStream inFile = new FileInputStream(fileName);
                byte[] fileInBytes = new byte[numOfBytes];
                inFile.read(fileInBytes); //读入文件
                outToClient.writeBytes("HTTP/1.0 200 Document Follows\r\n"); //返
回报文
                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); //发送数据到
outToClient流
                connectionSocket.close(); //关闭连接套接字
                inFile.close();

                return 1; //如果是一次成功连接，函数返回值为1
            }
        }
        else System.out.println("Bad Request Message");
        return 0; //如果没有成功连接，函数返回值为0
    }

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

        ServerSocket listenSocket=new ServerSocket(6789); //创建欢迎套接字，聆听TCP
连接请求
        int num=0; //记录成功连接的次数

        while(num < 5){ //限制最大连接次数
            Socket connectionSocket = listenSocket.accept(); //创建连接套接字
            if (connection(connectionSocket)==1){ //调用connection方法，并通过返回值
判断是否连接成功
                num++;
                System.out.println(String.valueOf(num) + " times of
connection"); //打印出当前为止的连接数
            }
        }

    }
}
```

这个程序利用串行依次响应的方法实现了响应多个请求的服务器程序，在`WebServer`类中，主函数创建了类型为`ServerSocket`的`listenSocket`变量，这个`listenSocket`在6789号端口上监听，在一个`while`循环内，当有客户连接时，用`accept`方法创建一个新的连接套接字，接着调用`Connection()`方法对这个连接进行处理。

这个`Connection()`方法的主要内容与示例相似，先是定义数据的输入输出流，接着对从客户来的字节流进行处理，找到请求的文件名，读入文件，接着根据文件类型写返回报文，并将字节类型的文件一起返回给用户，关闭连接套接字，返回 1 结束。

有些浏览器在请求网页时，其实发出了两个请求报文，`GET /index.html HTTP/1.1`和`GET /favicon.ico HTTP/1.1`，前者为请求网页，后者是请求图标，但我们的网页没有制作这个图标，所以favicon.ico是不希望去访问的，否则可能发生路径错误，为此我采用了如下程序忽略了这个请求：

```
String str = new String("favicon.ico");
if(fileName.equals(str))
onnectionSocket.close();
```

这是一个字符串比较程序，`fileName`获得文件名的方式与示例程序相同，当发现请求的是favicon.ico时，关闭连接套接字，返回 0 结束。编译运行这个程序，在浏览器连续输入http://localhost:6789/index.html，即可在终端看到连接数，可见这个程序可以响应多个请求。

```
1 times of connection
2 times of connection
3 times of connection
4 times of connection
5 times of connection
```

1.2.3.3 可以响应多个请求的 WebServer.java 程序(多线程实现)

我编写的可以响应多个请求的 WebServer.java 程序源代码如下：

```
import java.io.*;
import java.net.*;
import java.util.*;

public class WebServer {

    public WebServer() throws IOException {
        ServerSocket listenSocket = new ServerSocket(6789); //创建欢迎套接字，聆听TCP连接请求
        while(true){
            Socket connectionSocket=listenSocket.accept(); //创建连接套接字
            new Connection(connectionSocket).start(); //创建一个新的线程对象并启动线程
        }
    }

    public static void main(String[] args) {
        try {
            new WebServer(); //多线程Web服务器程序开始运行
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}

class Connection extends Thread { //线程子类
    public static int num = 0;
    private Socket connectionSocket;

    public Connection(Socket s) throws IOException {
        this.connectionSocket = s;
    }

    public void run() { //线程启动后运行此方法
```

```
        try {
            String requestMessageLine;
            String fileName = "";
            BufferedReader inFromClient = new BufferedReader(new
InputStreamReader(connectionSocket.getInputStream())); //定义一个类型为
BufferedReader的inFromUser流对象
            DataOutputStream outToClient = new
DataOutputStream(connectionSocket.getOutputStream()); //创建连接到套接字的流对象，提供
到套接字的输出
            requestMessageLine = inFromClient.readLine(); //将用户的输入读入字符串
            StringTokenizer tokenizedLine = new
StringTokenizer(requestMessageLine); //将字符串标记化

            if(tokenizedLine.nextToken().equals("GET")){ //判断是否为一个请求
                fileName = tokenizedLine.nextToken();
                if(fileName.startsWith("/")==true)
                    fileName = fileName.substring(1); //找到请求文件的路径
                String str = new String("favicon.ico"); //请求的如果是图标，忽略这个
请求
                if(fileName.equals(str))
                    connectionSocket.close();
                else{
                    File file = new File(fileName); //打开文件
                    int numOfBytes = (int)file.length(); //统计长度
                    FileInputStream inFile = new FileInputStream(fileName);
                    byte[] fileInBytes = new byte[numOfBytes];
                    inFile.read(fileInBytes); //读入文件
                    outToClient.writeBytes("HTTP/1.0 200 Document Follows\r\n");
//返回报文
                    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); //发送数据到
outToClient流
                    connectionSocket.close(); //关闭连接套接字
                    inFile.close();

                    num++; //记录连接的次数
                    System.out.println(String.valueOf(num) + " times of
connection"); //打印出当前为止的连接数
                }

            }
        }
         catch (IOException e) {
            e.printStackTrace();
        }
    }

}
```

这个程序利用了多线程的方式实现了响应多个请求的服务器程序，在`WebServer`类中，主函数调用`WebServer`方法，在这个方法里面，创建了类型为`ServerSocket`的`listenSocket`变量，这个`listenSocket`在6789号端口上监听，在一个`while`循环内，当有客户连接时，用`accept`方法创建一个新的连接套接字，接着使用`new Connection(connectionSocket).start()`启动一个新的线程。

`Connection`类继承了`Thread`类，因此可以方便地用`start`方法为其分配内存，开启一个新的线程。在这个类里有两个属性，一个是`num`用来记录连接的次数，还有一个是`connectionSocket`连接套接字，在`Connection`构造方法里对其进行了初始化，接着是`run`方法，主要内容与示例相似，先是定义数据的输入输出流，接着对从客户来的字节流进行处理，找到请求的文件名，接着读入文件，接着根据文件类型写返回报文，并将字节类型的文件一起返回给用户，关闭连接套接字，在终端打印出当前为止的连接数，结束。

有些浏览器在请求网页时，其实发出了两个请求报文，`GET /index.html HTTP/1.1`和`GET /favicon.ico HTTP/1.1`，前者为请求网页，后者是请求图标，但我们的网页没有制作这个图标，所以favicon.ico是不希望去访问的，否则可能发生路径错误，为此我采用了如下程序忽略了这个请求：

```
String str = new String("favicon.ico");
if(fileName.equals(str))
onnectionSocket.close();
```

这是一个字符串比较程序，`fileName`获得文件名的方式与示例程序相同，当发现请求的是favicon.ico时，关闭连接套接字，结束。编译运行这个程序，在浏览器连续输入http://localhost:6789/index.html，即可在终端看到连接数，可见这个程序可以响应多个请求。

```
1 times of connection
2 times of connection
3 times of connection
```