Web服务器配置与交互过程分析

姓名：徐文斌
学号：2010234

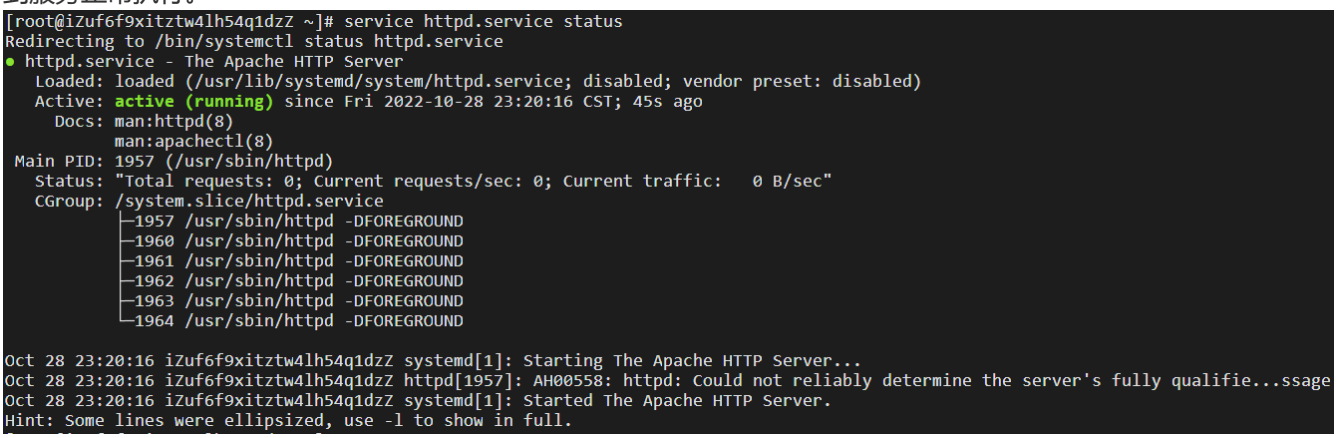
1. Web服务器搭建

为了体验比较真实的web客户端与服务器的交互过程，这里我选择白嫖阿里云的云服务器来搭建Web服务器。服务器为Centos操作系统，我们只需要安装Apache服务，并在Apache的默认根目录/var/www/html下编写自己的html网页即可。

1.1 安装Apache并启动服务

```
// 安装Apache服务及其扩展包
yum -y install httpd mod_ssl mod_perl mod_auth_mysql
// 启动Apache服务
systemctl start httpd.service
// 查看Apache服务状态
service httpd.service status
```

我们只需要将上述命令直接复制到终端中执行即可，最终执行命令查看Apache服务状态，结果如下图所示，可以看到服务正常执行。



1.2 编写html网页

我们首先将自己的logo上传到服务器中，命名为logo.jpg。然后在目录/var/www/html下编写文件index.html，文件内容如下。

```
<!DOCTYPE HTML>
<html>
<head>
    <meta charset="utf-8">
    <title>徐文斌的网页</title>
</head>
<body>
<div>
```

```
专业：计算机科学与技术<br>
姓名：徐文斌<br>
学号：2010234<br>
<img src="logo.jpg", alt="logo">
</div>
</body>
</html>
```

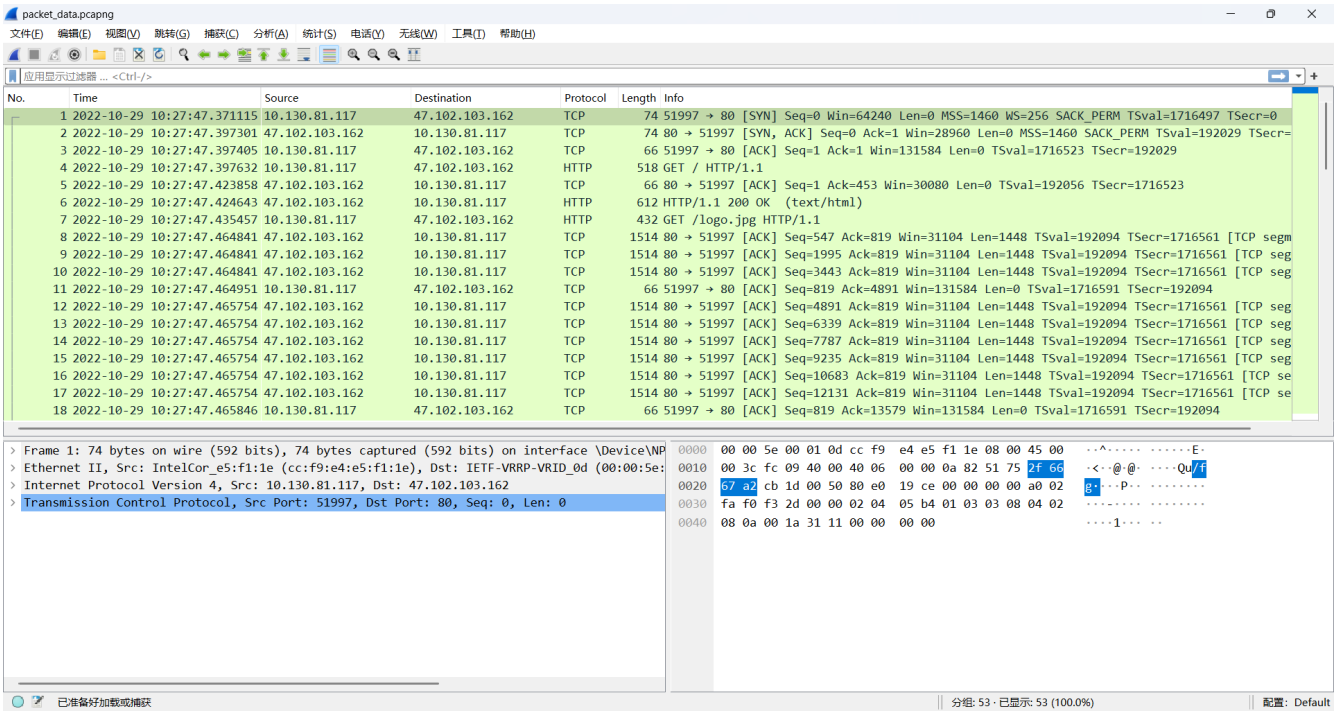
1.3 访问网页进行测试

完成了前两步之后，我们就可以访问Web服务器得到自己编写的页面了，结果如下图所示。

专业：计算机科学与技术
姓名：徐文斌
学号：2010234

2. 客户端与服务器交互过程分析

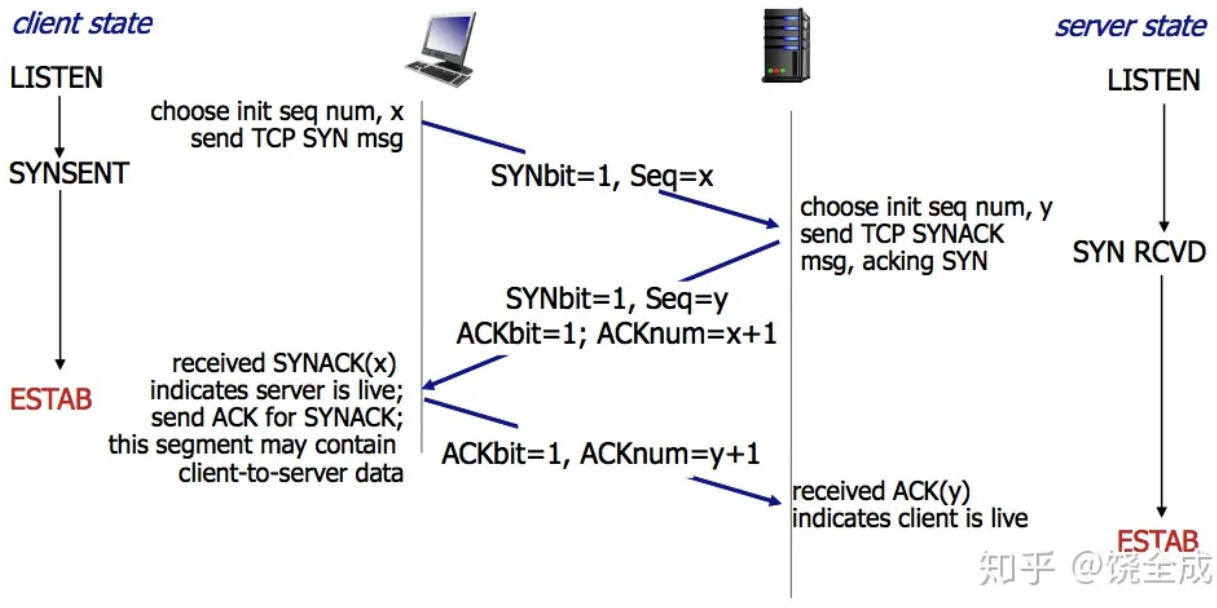
我们打开wireshark选择WLAN进行捕获。使用浏览器访问自己的网页，wireshark抓包结果如下图。下面我们对通信中的三次握手、http请求、四次挥手进行一个简要的分析。



2.1 三次握手

TCP是一个可靠的传输层协议，协议的可靠性主要通过序列号和滑动窗口机制来实现。TCP的三次握手过程作用就是为了确认双方的接收能力和发送能力是否正常、指定自己的初始化序列号为后面的可靠性传送做准备，主要起到同步序列号的作用。下图为TCP三次握手的过程。

- 第一次握手：客户端给服务端发一个SYN报文，并指明客户端的初始化序列号ISN(c)，此时客户端处于SYN_SENT状态。
- 第二次握手：服务器收到客户端的SYN报文之后，会以自己的SYN报文作为应答，为了确认客户端的SYN，将客户端的ISN+1作为ACK的值，此时服务器处于SYN_RCVD的状态。
- 第三次握手：客户端收到SYN报文之后，会发送一个ACK报文，值为服务器的ISN+1。此时客户端处于ESTABLISHED状态。服务器收到ACK报文之后，也处于ESTABLISHED状态，此时双方已建立起了连接。



下面我们对wireshark抓到的包进行分析。本地客户端主动向服务器发起建连请求。

```
10.130.81.117       47.102.103.162      TCP        74 51997 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM TSval=1716497 TSecr=0
47.102.103.162      10.130.81.117       TCP        74 80 → 51997 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM TSval=192029 TSecr=
10.130.81.117       47.102.103.162      TCP        66 51997 → 80 [ACK] Seq=1 Ack=1 Win=131584 Len=0 TSval=1716523 TSecr=192029
```

先看第一个报文段，如下图所示，由客户端向服务器发送。发送该报文后客户端进入SYN_SENT阶段，等待服务器响应。报文中FLAG中的SYN位置为1。客户端发送时会随机初始化一个初始序号放在序号段中。在这里初始化为2162170318。

```
Sequence Number: 0    (relative sequence number)
Sequence Number (raw): 2162170318
[Next Sequence Number: 1    (relative sequence number)]
Acknowledgment Number: 0
Acknowledgment number (raw): 0
1010 .... = Header Length: 40 bytes (10)
Flags: 0x002 (SYN)
   000. .... .... = Reserved: Not set
   ...0 .... .... = Accurate ECN: Not set
   .... 0... .... = Congestion Window Reduced: Not set
   .... .0.. .... = ECN-Echo: Not set
   .... ..0. .... = Urgent: Not set
   .... ...0 .... = Acknowledgment: Not set
   .... .... 0... = Push: Not set
   .... .... .0.. = Reset: Not set
   .... .... ..1. = Syn: Set
   .... .... ...0 = Fin: Not set
   [TCP Flags: ··········S·]
```

再看第二个报文，如下图所示，该报文由服务器发来，表明服务器已经预留了缓存和创建了连接相关的变量。该报文既用于对来自客户端报文中的序列号进行确认，也用于和客户端同步服务器自己的初始序列号。所以它的SYN和ACK都被置位。服务端根据客户端传过来的ACK报文，将首部的确认号置为client_isn+1，所以我们看到确认号部分为上面的数字加1。同时，报文中还含有服务器自己的初始序列号，这里为1630698414。

```
Sequence Number: 0    (relative sequence number)
Sequence Number (raw): 1630698414
[Next Sequence Number: 1    (relative sequence number)]
Acknowledgment Number: 1    (relative ack number)
Acknowledgment number (raw): 2162170319
1010 .... = Header Length: 40 bytes (10)
Flags: 0x012 (SYN, ACK)
   000. .... .... = Reserved: Not set
   ...0 .... .... = Accurate ECN: Not set
   .... 0... .... = Congestion Window Reduced: Not set
   .... .0.. .... = ECN-Echo: Not set
   .... ..0. .... = Urgent: Not set
   .... ...1 .... = Acknowledgment: Set
   .... .... 0... = Push: Not set
   .... .... .0.. = Reset: Not set
   .... .... ..1. = Syn: Set
   .... .... ...0 = Fin: Not set
   [TCP Flags: ·····A··S·]
```

最后一个报文，如下图所示。这个报文由客户端发送。他相当于告诉服务器“我知道你没问题了，咱们可以通话了”。报文中的ACK值为上一个来自服务器的报文中的SEQ值加1，用于对服务器的初始序列号进行确认。到此为止，TCP连接已经成功建立。

```
Sequence Number: 1    (relative sequence number)
Sequence Number (raw): 2162170319
[Next Sequence Number: 1    (relative sequence number)]
Acknowledgment Number: 1    (relative ack number)
Acknowledgment number (raw): 1630698415
1000 .... = Header Length: 32 bytes (8)
Flags: 0x010 (ACK)
   000. .... .... = Reserved: Not set
   ...0 .... .... = Accurate ECN: Not set
   .... 0... .... = Congestion Window Reduced: Not set
   .... .0.. .... = ECN-Echo: Not set
   .... ..0. .... = Urgent: Not set
   .... ...1 .... = Acknowledgment: Set
   .... .... 0... = Push: Not set
   .... .... .0.. = Reset: Not set
   .... .... ..0. = Syn: Not set
   .... .... ...0 = Fin: Not set
   [TCP Flags: ·······A····]
```

2.2 HTTP访问

请求报文

http访问中，浏览器首先向服务器发送http请求报文，以请求某一资源。如下所示为浏览器发送的第一个http请求报文。这里可以看到请求类型是get，http协议的版本为1.1。后面又有一些请求信息，如客户端类型，字体，编码方式，请求的网页资源等一系列信息。

```
10.130.81.117      47.102.103.162      HTTP      518 GET / HTTP/1.1
Hypertext Transfer Protocol
  GET / HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): GET / HTTP/1.1\r\n]
    Request Method: GET
    Request URI: /
    Request Version: HTTP/1.1
  Host: 47.102.103.162\r\n
  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Ge
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,imag
  Accept-Encoding: gzip, deflate\r\n
  Accept-Language: zh-CN,zh;q=0.9,en-US;q=0.8,en;q=0.7\r\n
  Cache-Control: max-age=0\r\n
  Upgrade-Insecure-Requests: 1\r\n
  \r\n
  [Full request URI: http://47.102.103.162/]
  [HTTP request 1/2]
  [Response in frame: 6]
  [Next request in frame: 7]
```

响应报文

服务器接收到来自客户端的http报文后，会将客户端请求的数据发回给客户端。如下所示，可以看到状态码为200，意味着请求成功。报文结构和请求是类似的，这里将上一步请求的数据发送过来。

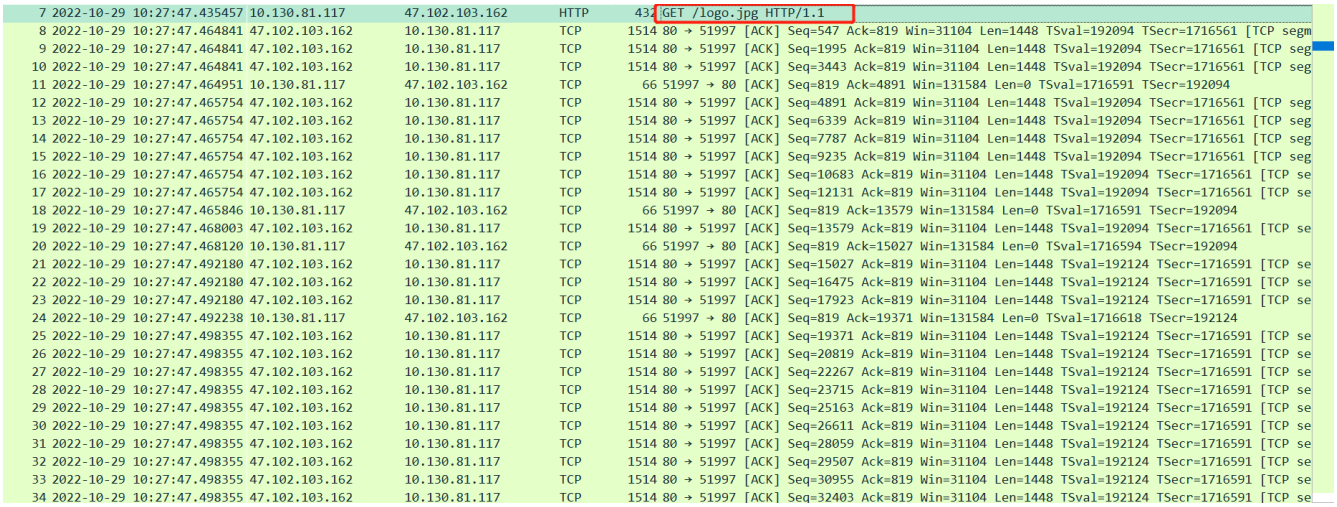
```
47.102.103.162      10.130.81.117       HTTP      612 HTTP/1.1 200 OK  (text/html)
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
    [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
    Response Version: HTTP/1.1
    Status Code: 200
    [Status Code Description: OK]
    Response Phrase: OK
  Date: Sat, 29 Oct 2022 02:27:45 GMT\r\n
  Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips mod_perl/2.0.11 Perl/v5.16.3\r\n
  Last-Modified: Sat, 29 Oct 2022 02:23:27 GMT\r\n
  ETag: "ff-5ec230f48c650"\r\n
  Accept-Ranges: bytes\r\n
  Content-Length: 255\r\n
  Content-Type: text/html; charset=UTF-8\r\n
  \r\n
  [HTTP response 1/2]
  [Time since request: 0.027011000 seconds]
  [Request in frame: 4]
  [Next request in frame: 7]
  [Next response in frame: 48]
  [Request URI: http://47.102.103.162/]
  File Data: 255 bytes
```

查看发送过来的资源，发现直接把index.html发送过来了。

```
Line-based text data: text/html (16 lines)
  <!DOCTYPE HTML>\n
  <html>\n
  <head>\n
      <meta charset="utf-8">\n
      <title>徐文斌的网页</title>\n
  </head>\n
  <body>\n
  <div>\n
  专业：计算机科学与技术<br>\n
  姓名：徐文斌<br>\n
  学号：2010234<br>\n
  <img src="logo.jpg", alt="logo">\n
  </div>\n
  </body>\n
  </html>\n
  \n
```

请求图片资源

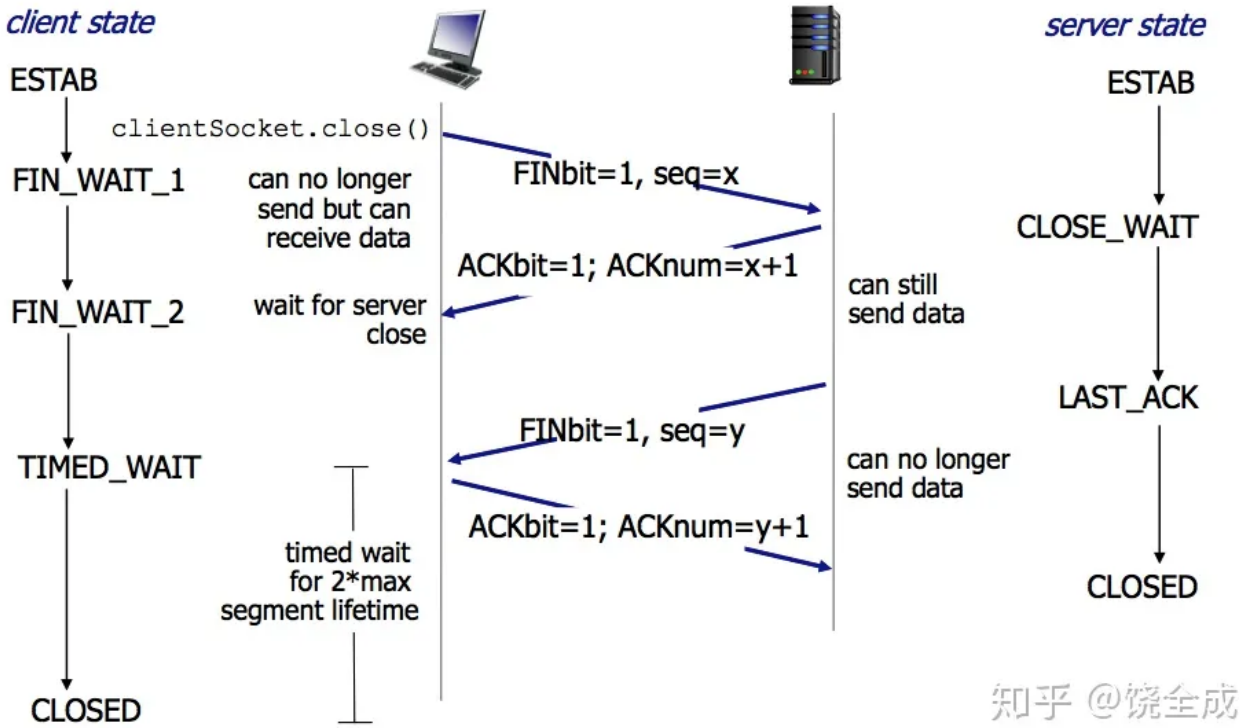
上述来自服务器的http响应报文只向客户端发回了html文件，并没有发送嵌入到html中的图片资源。浏览器对收到的html文件进行解析时，会再次向服务器发送请求，请求获得图片资源。如下图所示，浏览器再次向服务器发送GET报文，请求得到logo.jpg图片，GET报文和上述第一次发送的GET报文是类似的。服务器接收到请求报文后，会将图片资源发回给浏览器。这里由于图片比较大，一个TCP数据段无法装下，服务器将图片拆分成了多个TCP数据段发回给浏览器，可以看到下图服务器向客户端发送了许多TCP数据段。



2.3 四次挥手

TCP连接是双向传输的对等的模式，就是说双方都可以同时向对方发送或接收数据。当有一方要关闭连接时，会发送指令告知对方，我要关闭连接了。这时对方会回一个ACK，此时一个方向的连接关闭。但是另一个方向仍然可以继续传输数据，等到发送完了所有的数据后，会发送一个FIN段来关闭此方向上的连接。接收方发送ACK确认关闭连接。接收到FIN报文的一方只能回复一个ACK，它是无法马上返回对方一个FIN报文段的，需要等待来自应用层的结束数据传输的指令到来，才会向对方发送FIN报文段。下图为TCP四次挥手的过程。

- 第一次挥手：客户端发送一个FIN报文，报文中会指定一个序列号。此时客户端处于FIN_WAIT1状态，停止发送数据，等待服务端的确认。
- 第二次挥手：服务端收到FIN之后，会发送ACK报文，且把客户端的序列号值加1作为ACK报文的ACK值，表明已经收到客户端的报文了，此时服务端处于CLOSE_WAIT状态。
- 第三次挥手：如果服务端也想断开连接了，和客户端的第一次挥手一样，发给FIN报文，且指定一个序列号。此时服务端处于LAST_ACK的状态。
- 第四次挥手：客户端收到FIN之后，一样发送一个ACK报文作为应答，且把服务端的序列号值加1作为自己ACK报文的ACK值，此时客户端处于TIME_WAIT状态。需要过一阵子以确保服务端收到自己的ACK报文之后才会进入CLOSED状态，服务端收到ACK报文之后，就处于关闭连接状态了，处于CLOSED状态。



抓包得到的四次挥手过程如下图所示。和上述过程不同的是，这里是由服务器首先发起的FIN报文段。可以看到在客户端发送对来自服务器的图片资源进行确认的ACK报文段后，过了将近5秒，服务器发现自己无事可做时，进行第一次挥手，向客户端发送FIN报文来请求断开连接。第二次挥手，客户端收到FIN报文后向服务器发送ACK报文。然后是第三次挥手，客户端释放完资源后，向服务器发送FIN报文段，通知服务器客户端也没有剩余的工作了，请求断开连接。第四次挥手，服务器接受到FIN报文后向客户端发送ACK报文。此后两方完全断开连接。

```
2022-10-29 10:27:47.566953 10.130.81.117  47.102.103.162  TCP  66 51997 → 80 [ACK] Seq=819 Ack=44958 Win=130560 Len=0 TSval=1716693 TSecr=192151
2022-10-29 10:27:52.466921 47.102.103.162 10.130.81.117   TCP  66 80 → 51997 [FIN, ACK] Seq=44958 Ack=819 Win=31104 Len=0 TSval=197099 TSecr=1716693
2022-10-29 10:27:52.467046 10.130.81.117  47.102.103.162  TCP  66 51997 → 80 [ACK] Seq=819 Ack=44959 Win=130560 Len=0 TSval=1721593 TSecr=197099
2022-10-29 10:27:52.467126 10.130.81.117  47.102.103.162  TCP  66 51997 → 80 [FIN, ACK] Seq=819 Ack=44959 Win=130560 Len=0 TSval=1721593 TSecr=197099
2022-10-29 10:27:52.493664 47.102.103.162 10.130.81.117   TCP  66 80 → 51997 [ACK] Seq=44959 Ack=820 Win=31104 Len=0 TSval=197125 TSecr=1721593
```

报文具体内容和上述分析的过程一致，这里不做过多的展示。