

# 计算机网络实验报告

Lab2 配置Web服务器，编写简单页面，分析交互过程

网络安全空间学院 信息安全专业

2112492 刘修铭 1063

[https://github.com/lxmliu2002/Computer\\_Networking](https://github.com/lxmliu2002/Computer_Networking)

## 一、Web服务器配置

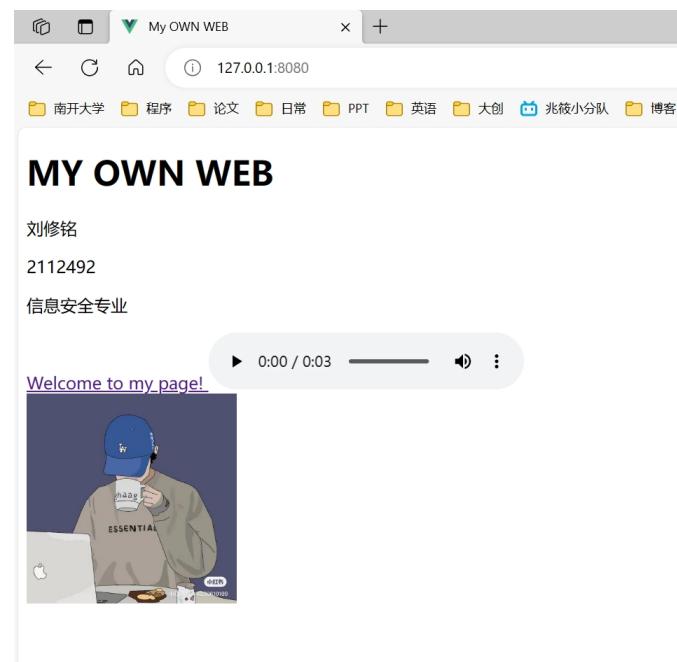
本项目借助Node.js，使用parcel-bundler打包配置Web服务器。

使用命令行，输入 `npm start` 指令，可以看到项目得以成功运行。



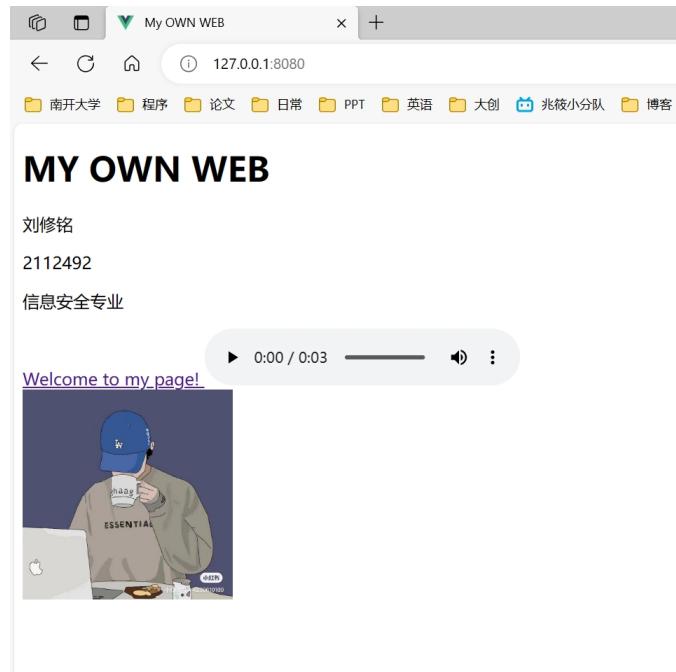
```
npm start
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。
安装最新的 PowerShell，了解新功能和改进！https://aka.ms/PSWindows
PS E:\刘修铭\南开大学\个人材料\课程\2023-2024 第1学期\计算机网络 张建忠, 徐敬东\Computer_Networking\lab2\codes\lxmliu200
2> npm start
> lxmliu2002@1.0.0 start
> node ./node_modules/parcel-bundler/bin/cli serve ./src/index.html --port 8080 --host 0.0.0.0
Server running at http://0.0.0.0:8080
Built in 685ms.
```

打开浏览器，输入 `http://127.0.0.1:8080` 得到如下界面。



## 二、简单页面编写

在给定框架的基础上，进行内容删改，使之成为一个自己的个人主页，包括专业、学号、姓名、LOGO、自我介绍音频信息等内容。为避免报告冗长，已将源代码附后。



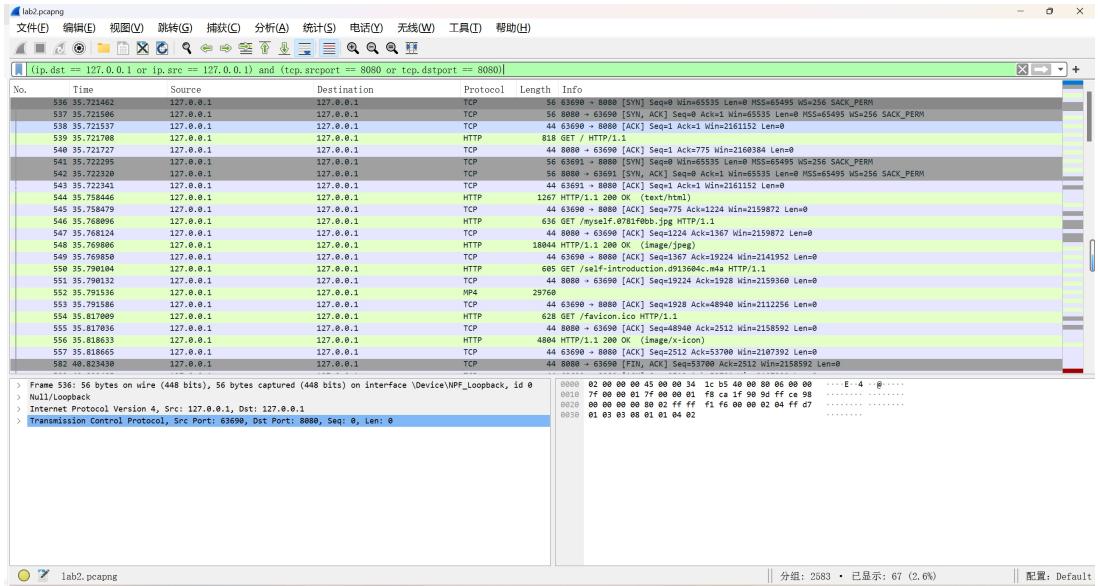
## 三、交互过程分析

### (一) 准备工作

- 下载安装Wireshark抓包软件
- 启动软件，选择 Adapter for loopback traffic capture 接口
- 按照前面创建的设定设置过滤器 (ip.dst == 127.0.0.1 or ip.src == 127.0.0.1) and (tcp.srcport == 8080 or tcp.dstport == 8080)

### (二) 数据分析

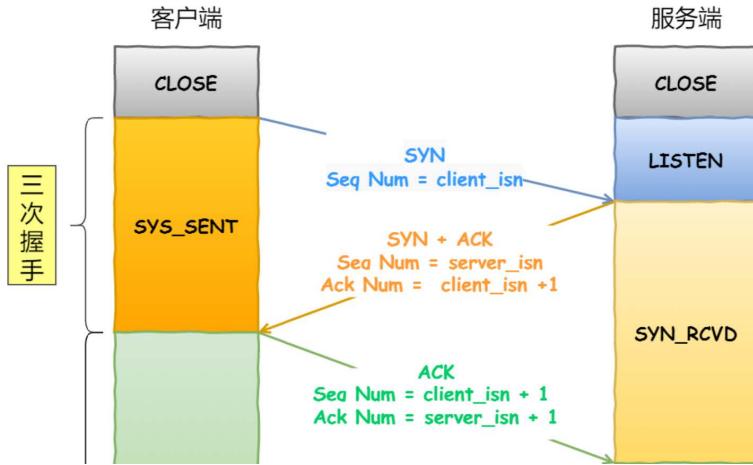
按照上面设定，运行一段时间，得到如下的抓包结果。



接下来将对整个交互过程进行分析。

## 1. TCP三次握手建立连接

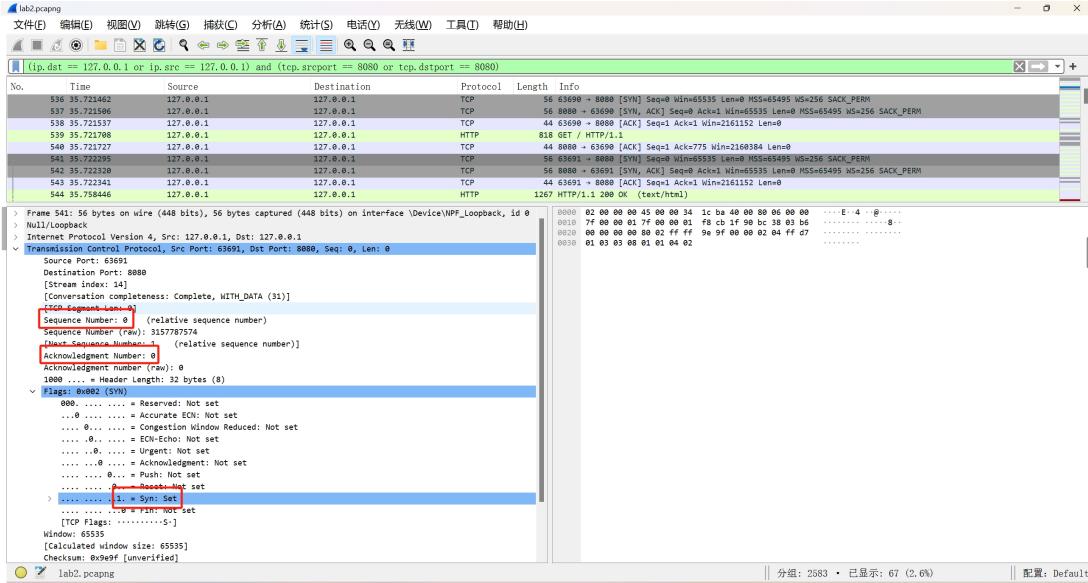
TCP使用三次握手建立连接



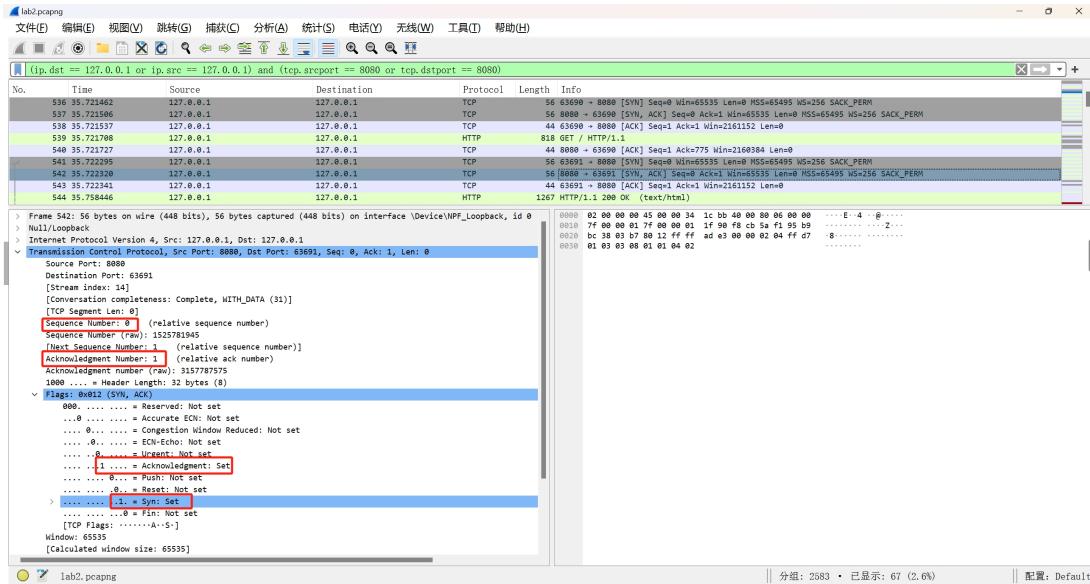
在wireshark中可以看到三次握手的过程。

1120 130.640416	127.0.0.1	127.0.0.1	TCP	56 63772 -> 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
1121 130.640459	127.0.0.1	127.0.0.1	TCP	56 8080 -> 63772 [SYN, ACK] Seq=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
1122 130.640489	127.0.0.1	127.0.0.1	TCP	44 63772 -> 8080 [ACK] Seq=1 Ack=1 Win=2161152 Len=0

- 第一次握手：建立连接时，客户端发送SYN到服务器，并进入SYN\_SENT状态，等待服务器确认。

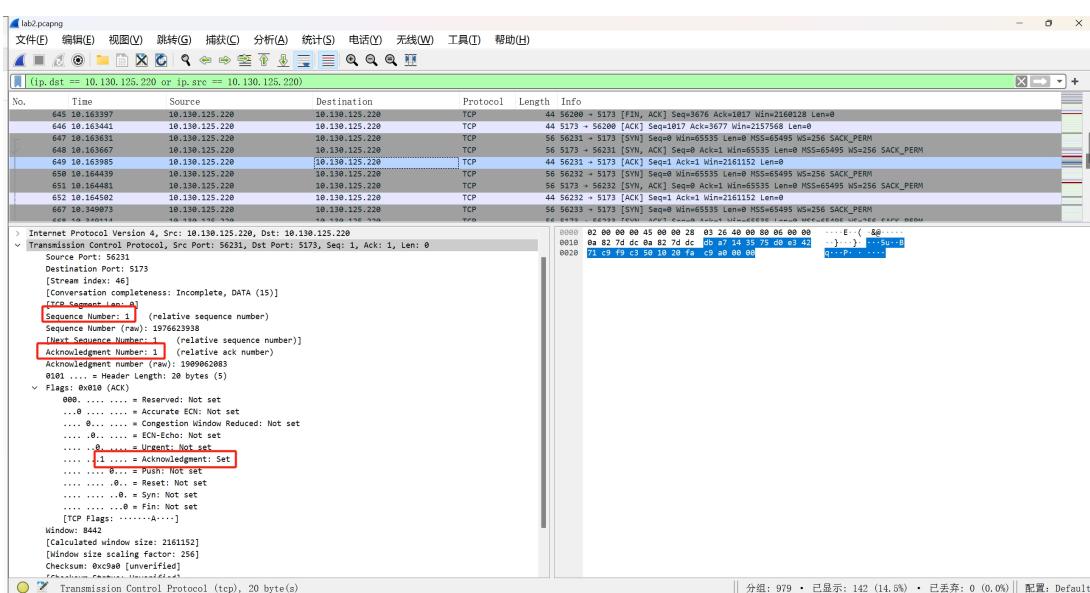


- TCP客户端向服务器端发送连接请求报文段，即客户端发送SYN到服务器，并进入SYN\_SENT状态，等待服务器端确认
- 第一次握手成功说明客户端的数据可以被服务端收到，说明客户端的发功能可用，说明服务端的收功能可用。但客户端自己不知道数据是否被接收
  - Source Port: 63691
  - Destination Port: 8080
  - SYN : 标志位，表示请求建立连接
  - Seq = 0 : 初始建立连接值为0，数据包的相对序列号从0开始，表示当前还没有发送数据
  - Ack =0: 初始建立连接值为0，已经收到包的数量，表示当前没有接收到数据
  - WIN = 65535来自Window size: 65535
  - MSS = 65495来自 Maximum segment size: 65495byte，最长报文段，TCP包所能携带的最大数据量，不包含TCP头和Option。一般为MTU值减去IPv4头部(至少20字节)和TCP头部(至少20字节)得到
  - WS = 256 来自windows scale : 8 (multiply by 256): 窗口扩张，放在TCP头之外的Option，向对方声明一个shift count，作为2的指数，再乘以TCP定义的接收窗口，得到真正的TCP窗口
- 第二次握手：服务器收到请求后，回送SYN+ACK信令到客户端，此时服务器进入SYN\_RECV状态。



- 第二次握手时，服务器端为该TCP连接分配缓存和变量
- 服务器端收到数据包后由标志位 SYN=1 得知客户端请求建立连接，然后便发送确认报文段（SYN+ACK 信令）到客户端，接着服务器进入SYN\_RECV状态
- 第二次握手成功说明服务端的数据可以被客户端收到，说明服务端的发功能可用，说明客户端的收功能可用。同时客户端知道自己的数据已经正确到达服务端，自己的发功能正常。但是服务端自己不知道数据是否被接收
  - Source Port: 8080
  - Destination Port: 63691
  - Seq = 0 : 初始建立值为0，表示当前还没有发送数据
  - Ack = 1 : 表示当前端成功接收的数据位数，虽然客户端没有发送任何有效数据，确认号还是被加1，因为包含SYN或FIN标志位

- 第三次握手：客户端收到SYN+ACK包，向服务器发送确认ACK包，客户端进入ESTABLISHED状态，服务器收到请求后也进入ESTABLISHED状态，完成三次握手，此时TCP连接成功，客户端与服务器开始传送数据。



- 第三次握手时，客户端为该TCP连接分配缓存和变量

- 客户端收到服务器端确认后，检查 ack 是否为  $x+1$ , ACK 是否为 1, 如果正确则再发送一个确认报文段给服务器，同时客户端进入ESTABLISHED状态，服务器收到请求后也进入ESTABLISHED状态，三次握手完成
- 第三次握手成功说明服务端知道自己的数据已经正确到达客户端端，自己的发功能正常。至此服务成功建立
  - Source Port: 63691
  - Destination Port: 8080
  - Seq = 1 : 表示当前已经发送1个数据
  - Ack = 1 : 表示当前端成功接收的数据位数
  - ACK : 标志位，表示已经收到记录

## 2. 发送与收取数据

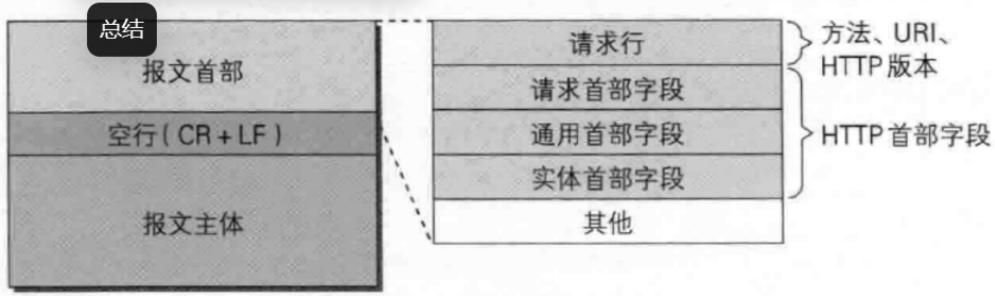
客户端和服务端建立连接后，开始传输数据。

127.0.0.1	TCP	56 63690 → 8080 [SYN] Seq=0 Win=65535 Le
127.0.0.1	TCP	56 8080 → 63690 [SYN, ACK] Seq=0 Ack=1 W
127.0.0.1	TCP	44 63690 → 8080 [ACK] Seq=1 Ack=1 Win=21
127.0.0.1	HTTP TCP	818 GET / HTTP/1.1
127.0.0.1	TCP	44 8080 → 63690 [ACK] Seq=1 Ack=775 Win=
127.0.0.1	TCP	56 63691 → 8080 [SYN] Seq=0 Win=65535 Le
127.0.0.1	TCP	56 8080 → 63691 [SYN, ACK] Seq=0 Ack=1 W
127.0.0.1	TCP	44 63691 → 8080 [ACK] Seq=1 Ack=1 Win=21
127.0.0.1	HTTP TCP	1267 HTTP/1.1 200 OK (text/html)
127.0.0.1	TCP	44 63690 → 8080 [ACK] Seq=775 Ack=1224 W
127.0.0.1	HTTP	636 GET /myself.0781f0bb.jpg HTTP/1.1
127.0.0.1	TCP	44 8080 → 63690 [ACK] Seq=1224 Ack=1367
127.0.0.1	HTTP	18044 HTTP/1.1 200 OK (image/jpeg)
127.0.0.1	TCP	44 63690 → 8080 [ACK] Seq=1367 Ack=19224
127.0.0.1	HTTP	605 GET /self-introduction.d913604c.m4a H
127.0.0.1	TCP	44 8080 → 63690 [ACK] Seq=19224 Ack=1928

- HTTP请求由三部分组成：请求行，消息报文，请求正文；格式为：Method Request-URI HTTP-Version CRLF；Method表示请求方法；Request-URI是一个统一资源标识符；HTTP-Version表示请求的HTTP协议版本；CRLF表示回车和换行。
- HTTP响应由三部分组成：状态行，消息报头，响应正文；格式为：HTTP-Version Status-Code Reason-Phrase CRLF；HTTP-Version表示服务器HTTP协议的版本，Status-Code表示服务器发回的响应状态代码，Reason-Phrase表示状态代码的文本描述。
  - 浏览器向域名发出HTTP请求；
  - 通过TCP- IP (DNS) ->MAC (ARP) ->网关->目的主机；
  - 目的主机收到数据帧，通过IP->TCP->HTTP，HTTP协议单元会回应HTTP协议格式封装好的HTML形式数据（HTTP响应）；
  - 该HTML数据通过TCP->IP (DNS) ->MAC (ARP) ->网关->我的主机；
  - 主机收到数据帧，通过IP->TCP->HTTP->浏览器，浏览器以网页形式显示HTML内容。

### (1) HTTP请求报文

一个HTTP请求报文由请求行 (request line) 、请求头部 (request header) 、空行和请求数据4个部分构成。

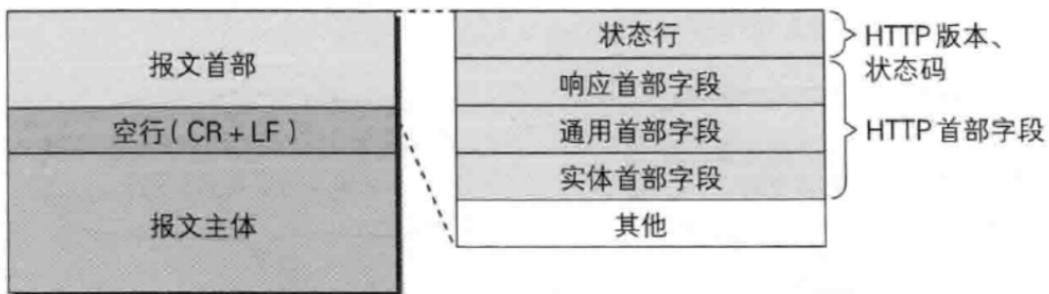


图：请求报文

- **请求行**, 给出了请求类型, URI给出请求的资源位置(/)。HTTP中的请求类型包
  - 请求行数据格式由三个部分组成: 请求方法、URI、HTTP协议版本, 他们之间用空格分隔。
- **请求头**, 主要是用于说明请求源、连接类型、以及一些Cookie信息等
  - 请求头部紧跟着请求行, 该部分主要是用于描述请求正文。
- **请求正文**, 一般用于存放POST请求类型的请求正文

## (2) HTTP响应报文

HTTP响应报文由状态行（HTTP版本、状态码（数字和原因短语））、响应头部、空行和响应体4个部分构成。



图：响应报文

- **状态行**, 主要给出响应HTTP协议的版本号、响应返回状态码、响应描述, 同样是单行显示。
  - 1XX 请求正在处理
  - 2XX 请求成功
    - 302 Found 资源的URI已临时定位到其他位置, 客户端不会更新URI。
    - 303 See Other 资源的URI已更新, 明确表示客户端要使用GET方法获取资源。
    - 304 Not Modified 当客户端附带条件请求访问资源时资源已找到但未符合条件请求。
  - 400 请求报文中存在语法错误
  - 403 对请求资源的访问被服务器拒绝
  - 404 请求资源不存在
  - 405 请求的方式不支持
  - 5XX 服务器错误
    - 503 服务器暂时处于超负载状态或正在进行停机维护
- **响应头部**, 与响应头部类似, 主要是返回一些服务器的基本信息, 以及一些Cookie值等
- **响应体**, 为请求需要得到的具体数据, 可以为任何类型数据, 一般网页浏览返回的为html文件内容

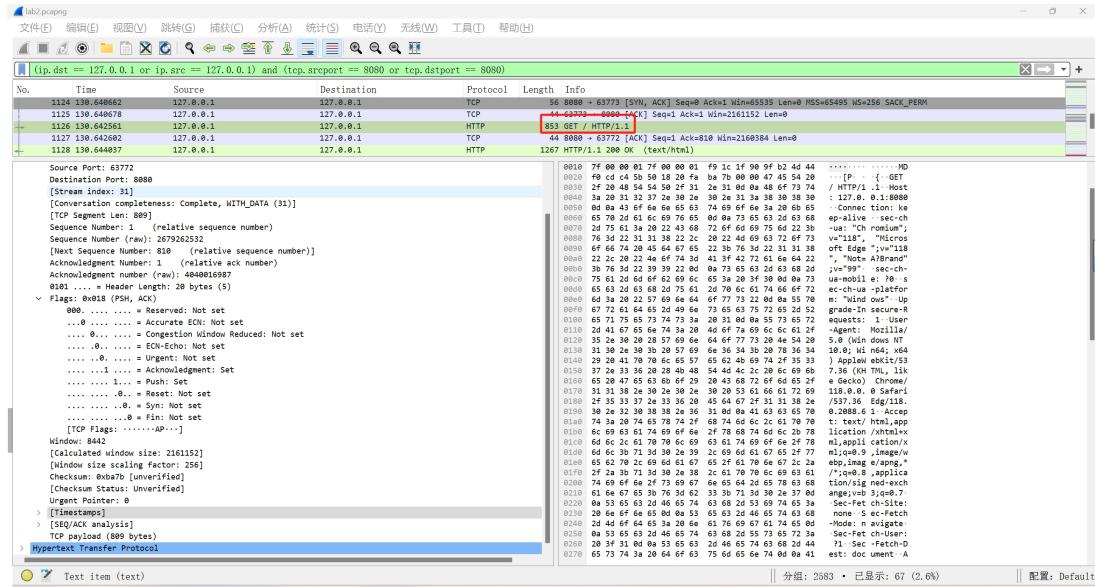
### (3) HTTP请求与响应

在本次实验数据中，可以看到在三次握手建立连接与四次挥手关闭连接中间有四次HTTP请求与响应。

127.0.0.1	TCP	56 63773 → 8080 [SYN] Seq=0 Win=0 MSS=65495 WS=256 SACK_PERM	
127.0.0.1	TCP	56 8080 + 63773 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM	
127.0.0.1	HTTP	853 GET / HTTP/1.1	
127.0.0.1	TCP	44 8080 + 63773 [ACK] Seq=1 Ack=0 Win=2161152 Len=0	
127.0.0.1	HTTP	1267 HTTP/1.1 200 OK (text/html)	
127.0.0.1	HTTP	44 63772 → 8080 [ACK] Seq=810 Ack=1224 Win=2159872 Len=0	
127.0.0.1	HTTP	787 GET /myself.0781f0bb.jpg HTTP/1.1	
127.0.0.1	TCP	44 8080 + 63772 [ACK] Seq=1224 Ack=1553 Win=2150616 Len=0	
127.0.0.1	HTTP	18044 HTTP/1.1 200 OK (image/jpeg)	
127.0.0.1	TCP	44 63772 → 8080 [ACK] Seq=1553 Ack=19224 Win=2141952 Len=0	
127.0.0.1	HTTP	767 GET /self-introduction.d913604ac.m4a HTTP/1.1	
127.0.0.1	TCP	44 8080 + 63772 [ACK] Seq=19224 Ack=2276 Win=2158649 Len=0	
127.0.0.1	MP4	29260	
127.0.0.1	TCP	44 63772 → 8080 [ACK] Seq=2276 Ack=48940 Win=2112256 Len=0	
127.0.0.1	HTTP	779 GET /favicon.ico HTTP/1.1	
127.0.0.1	TCP	44 8080 + 63772 [ACK] Seq=48940 Ack=3011 Win=2158080 Len=0	
127.0.0.1	HTTP	4804 HTTP/1.1 200 OK (image/x-icon)	
127.0.0.1	TCP	44 63772 → 8080 [ACK] Seq=3011 Ack=53701 Win=2107392 Len=0	
127.0.0.1	TCP	44 63772 → 8080 [FIN, ACK] Seq=3011 Ack=53701 Win=2107392 Len=0	
127.0.0.1	TCP	44 8080 + 63772 [ACK] Seq=53701 Ack=3012 Win=2158080 Len=0	

下面以第一轮请求与响应为例进行分析。

#### 客户端向服务器端发送请求



#### • 请求行

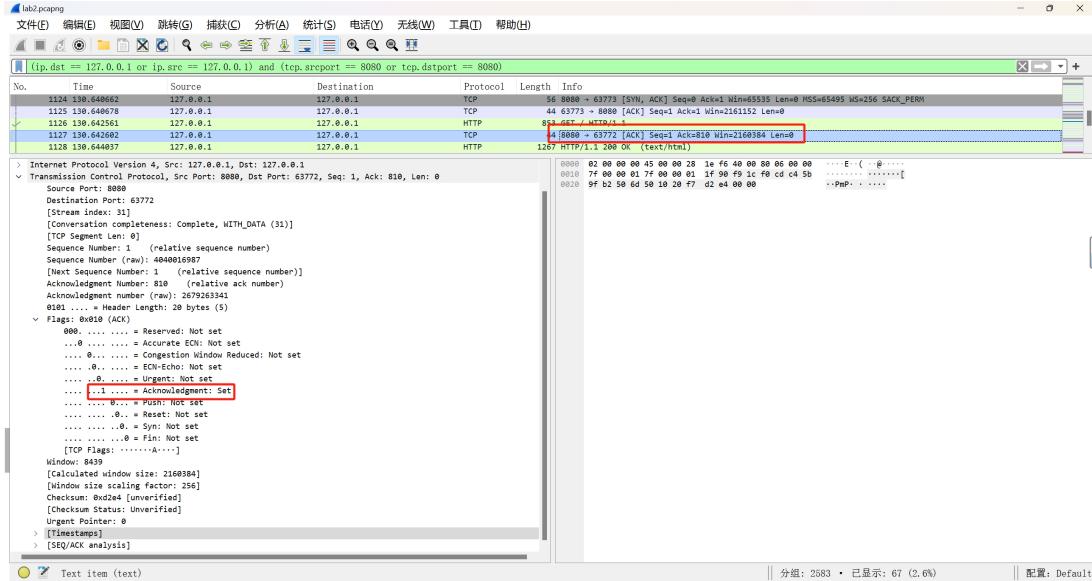
- 请求方式：GET
- 请求URI：/
- 协议版本：HTTP/1.1

#### • 请求头

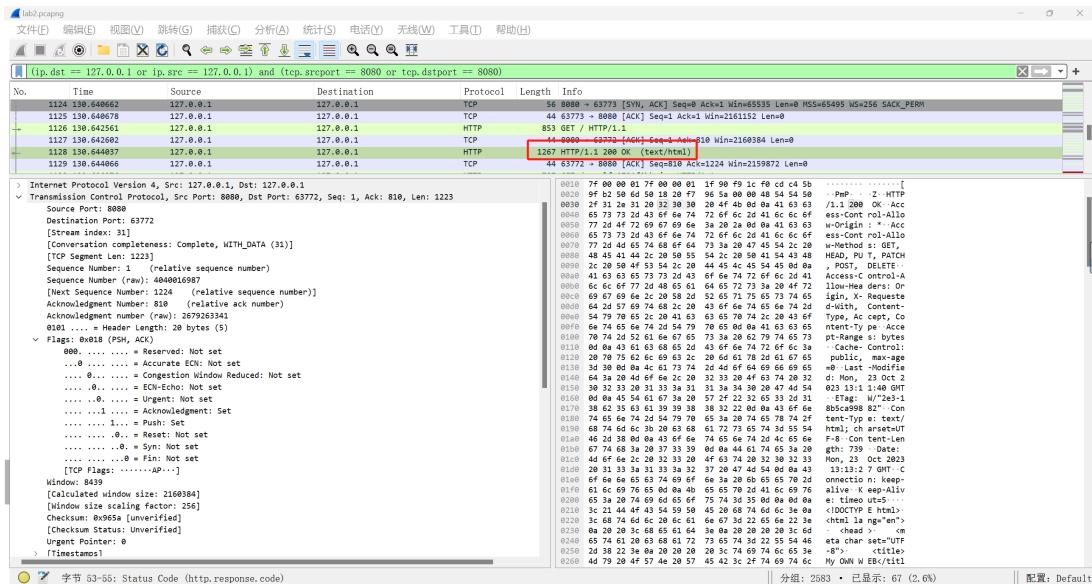
- Host
- Connection
- User-Agent

.....

## 服务器端回复ACK表示收到请求



## 服务器端向客户端发送响应报文



### • 状态行

- 协议版本: HTTP/1.1
- 状态码: 200, 表示OK

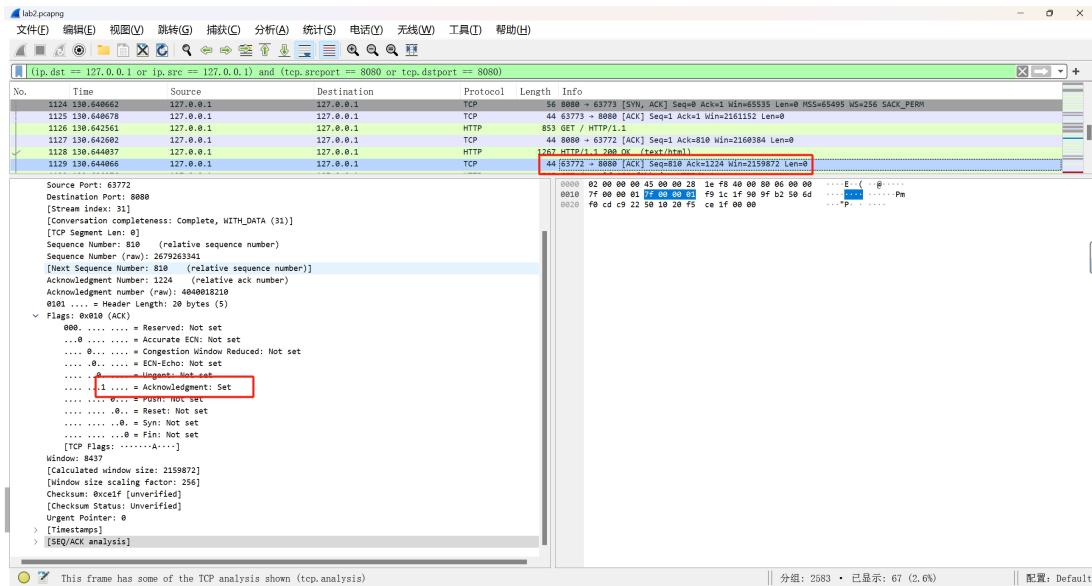
### • 响应头部

- Cache-Control
- Last-Modified
- Connection
- ...

### • 响应体

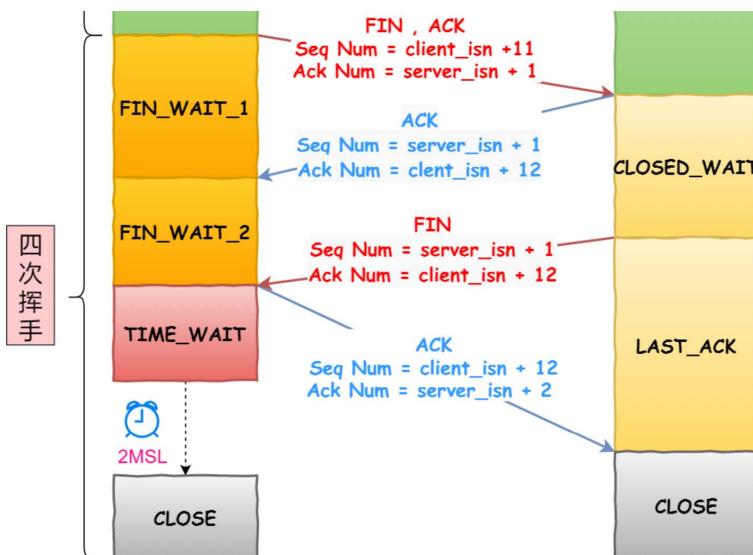
- index.html的内容

## 客户端回复ACK表示收到请求



## 3. TCP四次挥手关闭连接

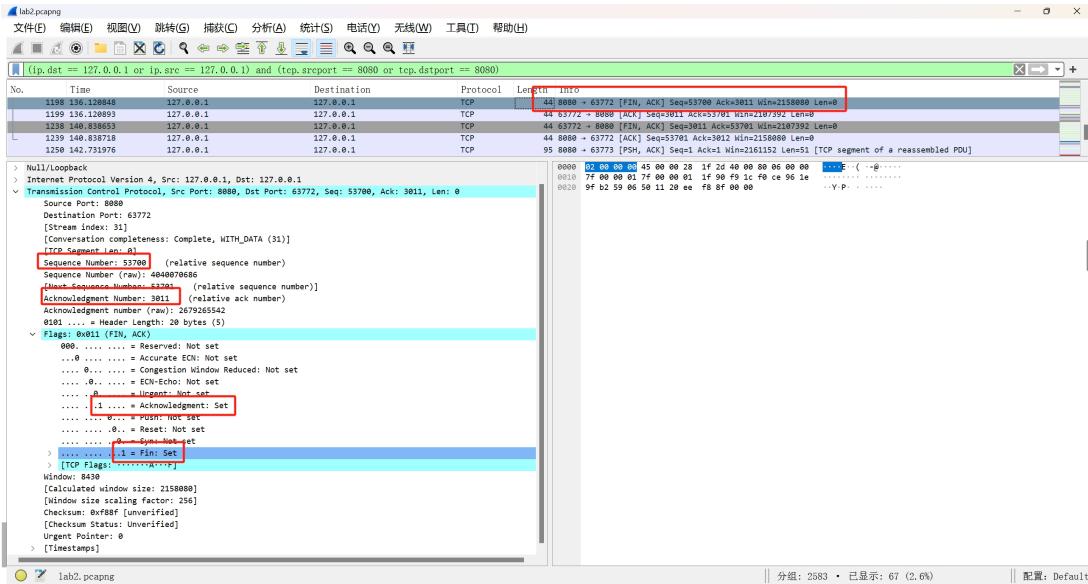
TCP使用四次挥手关闭连接，客户端和服务端分别释放连接



在wireshark中可以看到四次挥手的过程。

1198 136.120848	127.0.0.1	127.0.0.1	TCP	44 8080 → 63772 [FIN, ACK] Seq=53700 Ack=3011 Win=2158080 Len=0
1199 136.120893	127.0.0.1	127.0.0.1	TCP	44 63772 → 8080 [ACK] Seq=3011 Ack=53701 Win=2107392 Len=0
1238 140.838653	127.0.0.1	127.0.0.1	TCP	44 63772 → 8080 [FIN, ACK] Seq=3011 Ack=53701 Win=2107392 Len=0
1239 140.838718	127.0.0.1	127.0.0.1	TCP	44 8080 → 63772 [ACK] Seq=53701 Ack=3012 Win=2158080 Len=0

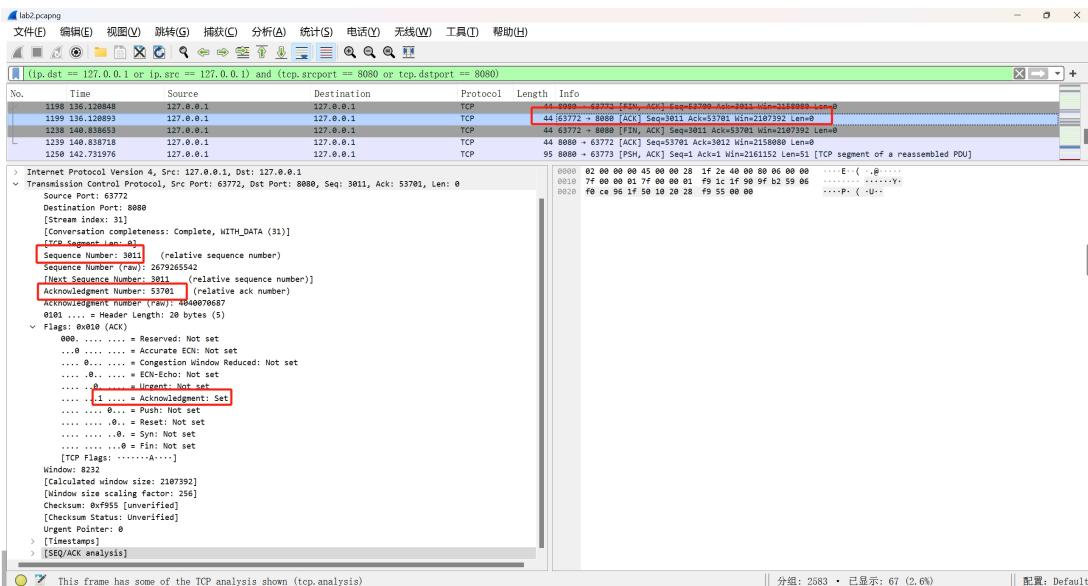
- 第一次挥手：主动关闭方发送一个FIN并进入FIN\_WAIT1状态



- 客户端向服务器端发送主动关闭连接报文
- 客户端进入FIN\_WAIT1状态，等待服务器端回复

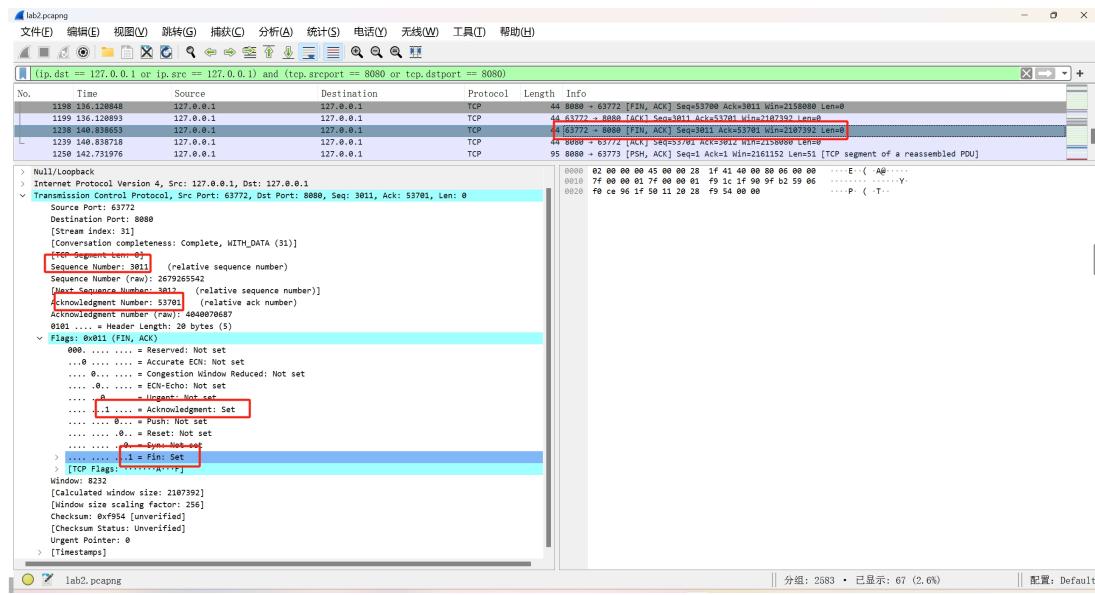
- Source Port: 8080
- Destination Port: 63772
- Seq = 53700
- Ack = 3011
- ACK、FIN：标志位

- 第二次挥手：被动关闭方接收到主动关闭方发送的FIN并发送ACK，此时被动关闭方进入CLOSE\_WAIT状态；主动关闭方收到被动关闭方的ACK后，进入FIN\_WAIT2状态



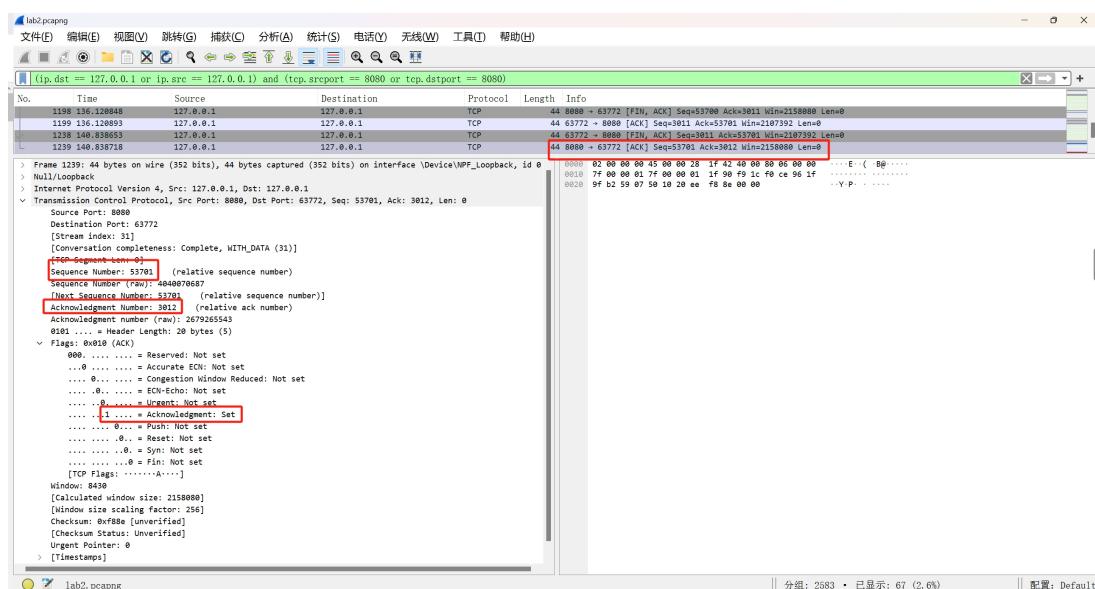
- 服务器端接收报文，并回复确认报文
  - 服务器端进入CLOSE\_WAIT状态；客户端收到服务器端的确认报文后进入FIN\_WAIT2状态
- Source Port: 63772
  - Destination Port: 8080
  - Seq = 3011

- Ack = 53701
- ACK : 标志位
- 第三次挥手：被动关闭方发送一个FIN并进入LAST\_ACK状态



- 服务器端在确认数据传输完毕后向客户端发送结束报文
- 服务器端进入LAST\_ACK状态，等待客户端最后确认
- Source Port: 8080
- Destination Port: 63772
- Seq = 3011
- Ack = 53701
- ACK、FIN : 标志位

- 第四次挥手：主动关闭方收到被动关闭方发送的FIN并发送ACK，此时主动关闭方进入TIME\_WAIT状态，经过2MSL时间后关闭连接；被动关闭方收到主动关闭方的ACK后，关闭连接



- 客户端回复确认报文，然后进入TIME\_WAIT状态，经过2MSL时间后关闭连接（MSL是数据分节在网络中存活的最长时间）
- 服务器端接收到确认报文后关闭连接

- Source Port: 63772
- Destination Port: 8080
- Seq = 53701
- Ack = 3012
- ACK : 标志位

比较主流的文章都说关闭TCP会话是四次挥手，但是实际上为了提高效率通常合并第二、三次的挥手，即三次挥手。

## 四、问题思考

---

### 1. 为何是三次握手而不是两次或四次握手？

失效的连接请求是指，若客户端向服务端发送的连接请求丢失，客户端等待应答超时后就会再次发送连接请求，此时，上一个连接请求就是失效的。三次握手可以防止失效的连接请求报文段被服务端接收从而产生错误。

若建立连接只需两次握手，客户端并没有太大的变化，仍然需要获得服务端的应答后才进入ESTABLISHED状态，而服务端在收到连接请求后就进入ESTABLISHED状态。此时如果网络拥塞，客户端发送的连接请求迟迟到不了服务端，客户端便超时重发请求，如果服务端正确接收并确认应答，双方便开始通信，通信结束后释放连接。此时，如果那个失效的连接请求抵达了服务端，由于只有两次握手，服务端收到请求就会进入ESTABLISHED状态，等待发送数据或主动发送数据。但此时的客户端早已进入CLOSED状态，服务端将会一直等待下去，这样浪费服务端连接资源。

但是是可以四次甚至更多次，但选择三次的原因是因为三次握手足以，无须过多浪费资源。

### 2. 为何是四次挥手而不是两次或三次挥手？（以客户端主动申请断开连接为例）

第一次挥手表示客户端发送了一个FIN的包，表示客户端已发送数据完毕，但是服务端这个时候可能还有数据没有发送完成，先发送给客户端一个ACK的包，等待自己的数据发送完成才能向客户端发送一个FIN的包，表示自己的数据也已发送完成。这样中间就必须为两次来发送ACK和FIN。

### 3. 304状态码

在抓包时有时会遇到304状态码，该状态码的含义是目前请求的信息与之前请求的内容相比没有改动，此时客户端从缓存读取即可，无须由服务器端再发送。

### 4. keep-alive连接状态是什么？

在get请求的请求头部分有个connection字段，该字段定义了其连接状态。其中，keep-alive意图在于短时间内连接复用，希望可以在短时间内在同一个连接上进行多次请求/响应，避免每次通信建立连接的繁琐与资源浪费。