
南開大學
Nankai University

# 计算机网络实验二

## 配置Web服务器

## 编写简单页面，分析交互过程

姓名：刘伟

学号：**2013029**

专业：物联网工程

## 一、实验要求

（1）搭建Web服务器（自由选择系统），并制作简单的Web页面，包含简单文本信息（至少包含专业、学号、姓名）和自己的LOGO。

（2）通过浏览器获取自己编写的Web页面，使用Wireshark捕获浏览器与Web服务器的交互过程，并进行简单的分析说明。

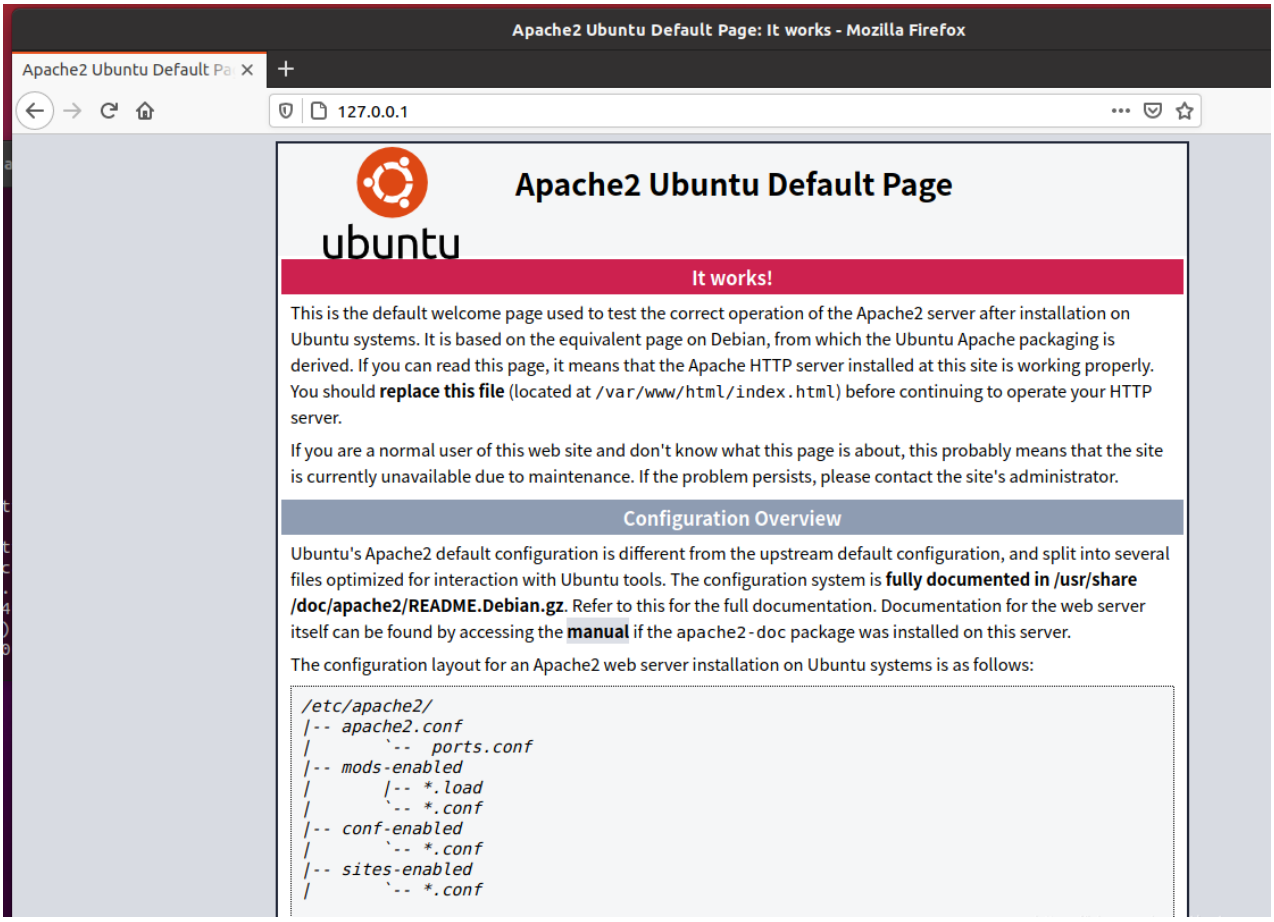
（3）提交实验报告。

## 二、搭建Web服务器

### 实验选择的**Web**服务器搭建环境是：**Vmware**虚拟机上的**Ubuntu**系统

1. 下载apache2：sudo apt-get install apache2
2. 由于apache2默认配置有index.html文件

下载apache2后默认打开127.0.0.1web服务器的网页界面：



3. 对index.html进行修改，自行设计内容，制作简易界面

- 按要求设定个人web服务器界面，编写index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>hello</title>
</head>
<body>
    <h1 align="center">计算机网络</h1>
    <h2 align="center">专业：物联网工程</h2>
    <h2 align="center">姓名：刘伟</h2>
    <h2 align="center">学号：2013029</h2>
    <p style="text-align: center;">
        <img src="logo.png" alt="图片未加载"/>
    </p>
</body>
</html>
```

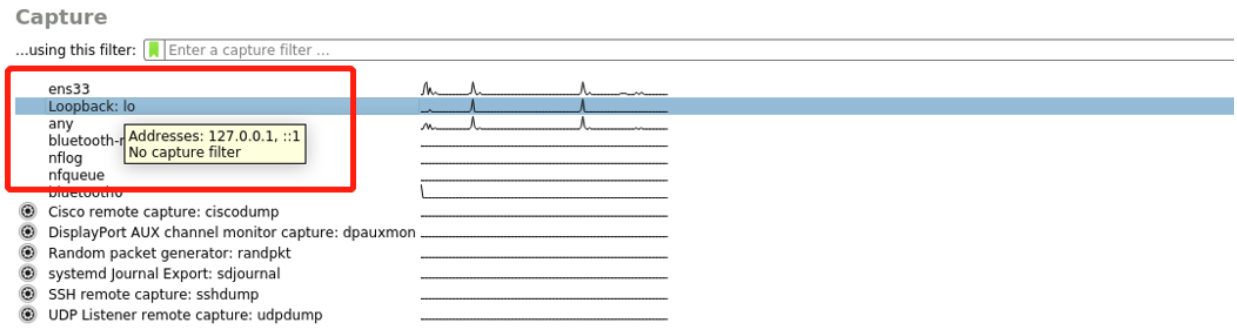
- 当在Ubuntu环境下打开浏览器，输入网址：127.0.0.1后，Web服务器展示界面如下：

## 三、Wireshark抓包分析浏览器与Web服务器的交互过程

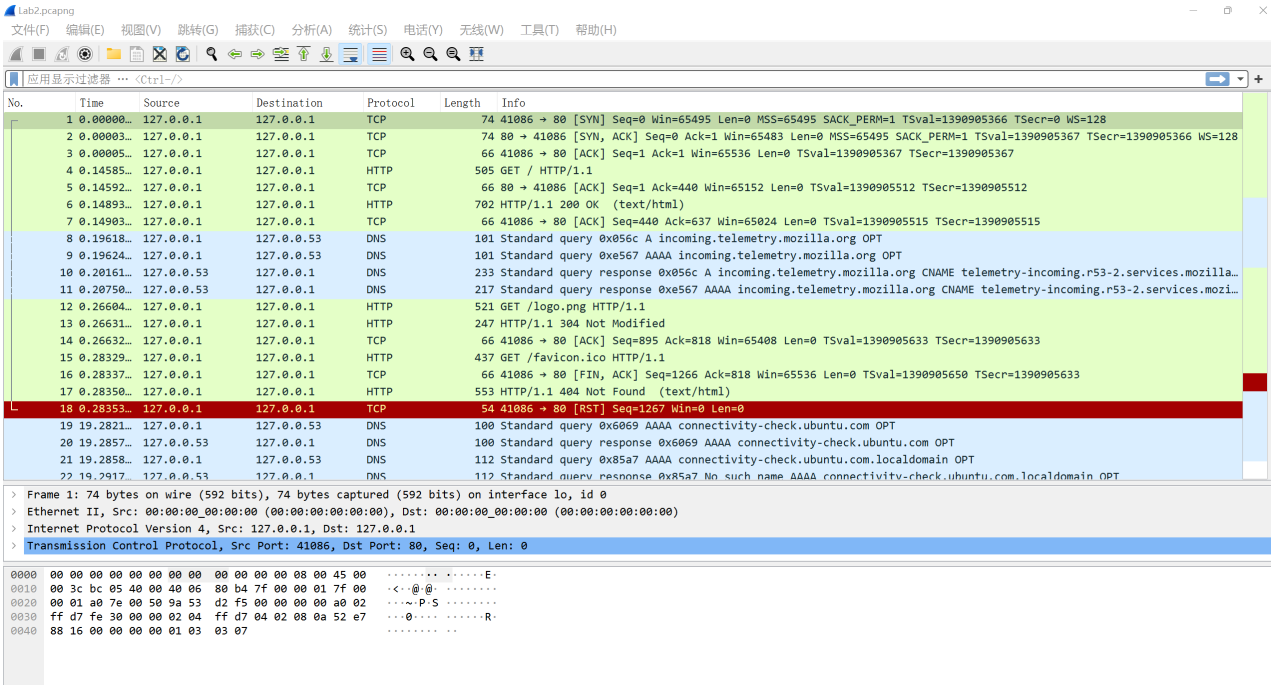
打开**Wireshark**，选择端口进行数据捕获抓包

由于该实验打开的Web服务器地址是：**127.0.0.1**

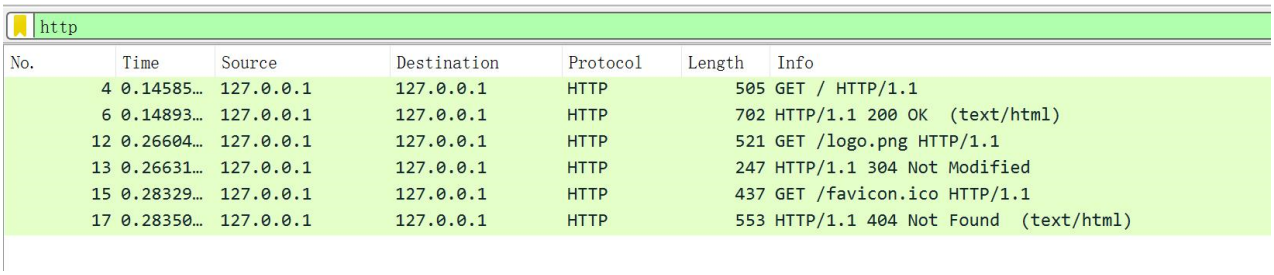
故Wireshark进行端口选择的时候应该选取：抓取本地回环包的网络接口 **loopback:lo**




**Wireshark**开始数据抓取后，使用浏览器访问网页，进而通过抓取内容分析交互过程



利用过滤器分析协议为**HTTP**的数据包



http

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 4 | 0.14585… | 127.0.0.1 | 127.0.0.1 | HTTP | 505 | GET / HTTP/1.1 |
| 6 | 0.14893… | 127.0.0.1 | 127.0.0.1 | HTTP | 702 | HTTP/1.1 200 OK (text/html) |
| 12 | 0.26604… | 127.0.0.1 | 127.0.0.1 | HTTP | 521 | GET /logo.png HTTP/1.1 |
| 13 | 0.26631… | 127.0.0.1 | 127.0.0.1 | HTTP | 247 | HTTP/1.1 304 Not Modified |
| 15 | 0.28329… | 127.0.0.1 | 127.0.0.1 | HTTP | 437 | GET /favicon.ico HTTP/1.1 |
| 17 | 0.28350… | 127.0.0.1 | 127.0.0.1 | HTTP | 553 | HTTP/1.1 404 Not Found (text/html) |

- No.4、No.12、No.15数据包：HTTP请求报文
- No.5、No.13、No.17数据包：HTTP响应报文

分析请求报文的内容

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 4 | 0.14585… | 127.0.0.1 | 127.0.0.1 | HTTP | 505 | GET / HTTP/1.1 |
| 6 | 0.14893… | 127.0.0.1 | 127.0.0.1 | HTTP | 702 | HTTP/1.1 200 OK (text/html) |
| 12 | 0.26604… | 127.0.0.1 | 127.0.0.1 | HTTP | 521 | GET /logo.png HTTP/1.1 |
| 13 | 0.26631… | 127.0.0.1 | 127.0.0.1 | HTTP | 247 | HTTP/1.1 304 Not Modified |
| 15 | 0.28329… | 127.0.0.1 | 127.0.0.1 | HTTP | 437 | GET /favicon.ico HTTP/1.1 |
| 17 | 0.28350… | 127.0.0.1 | 127.0.0.1 | HTTP | 553 | HTTP/1.1 404 Not Found (text/html) |

```
GET / HTTP/1.1\r\n                                                    请求行
  [Expert Info (Chat/Sequence): GET / HTTP/1.1\r\n]
    [GET / HTTP/1.1\r\n]
    [Severity level: Chat]
    [Group: Sequence]
  Request Method: GET
  Request URI: /
  Request Version: HTTP/1.1
Host: 127.0.0.1\r\n
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:105.0) Gecko/20100101 Firefox/105.0\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8\r\n
Accept-Language: en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate, br\r\n
Connection: keep-alive\r\n                                            请求头
Upgrade-Insecure-Requests: 1\r\n
Sec-Fetch-Dest: document\r\n
Sec-Fetch-Mode: navigate\r\n
Sec-Fetch-Site: none\r\n
Sec-Fetch-User: ?1\r\n
\r\n
[Full request URI: http://127.0.0.1/]
[HTTP request 1/3]
[Response in frame: 6]
[Next request in frame: 12]
```

请求报文由请求行、请求头、请求体组成。

GET / HTTP/1.1\r\n 被称为 请求行 ，由三个部分组成：请求方法、请求URI、HTTP版本（彼此之间空格隔开，注意末尾需要加上回车字符与换行字符）。

- 请求方法字段给出了请求类型：GET
- 请求URI字段给出了请求的资源位置
- HTTP版本字段给出采用的HTTP协议版本

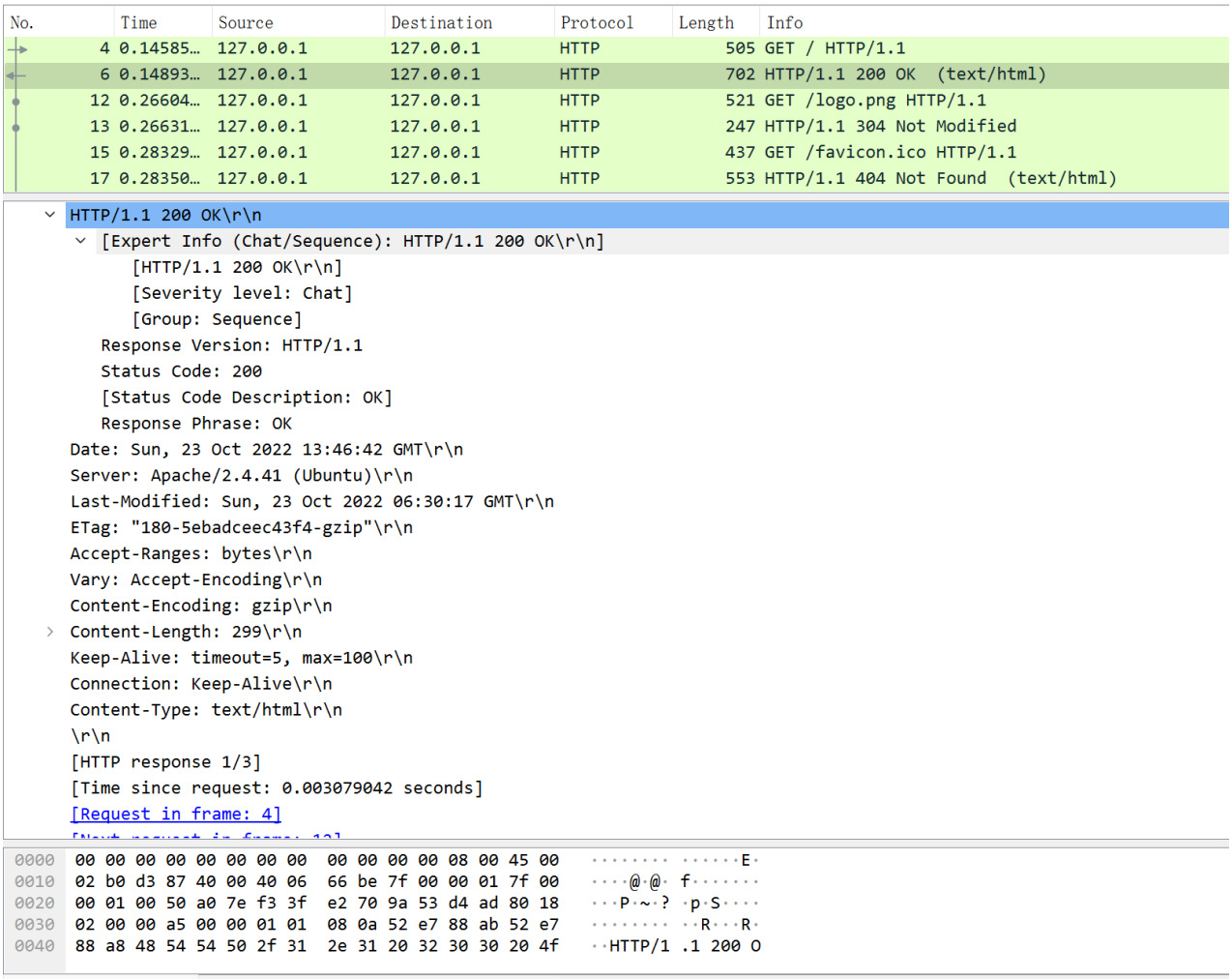
在请求行下紧接的便是： 请求头 。

请求头部分主要用于描述请求正文，说明请求源、连接类型、以及可能存有的cookie信息等等。

注意该部分每一行包含“键-值”对，并以\r\n结尾。

独立的回车换行指明请求头，紧接的便是 请求体 。

## 分析响应报文的内容



```
No.   Time      Source      Destination  Protocol  Length  Info
    4 0.14585…  127.0.0.1   127.0.0.1    HTTP         505 GET / HTTP/1.1
    6 0.14893…  127.0.0.1   127.0.0.1    HTTP         702 HTTP/1.1 200 OK  (text/html)
   12 0.26604…  127.0.0.1   127.0.0.1    HTTP         521 GET /logo.png HTTP/1.1
   13 0.26631…  127.0.0.1   127.0.0.1    HTTP         247 HTTP/1.1 304 Not Modified
   15 0.28329…  127.0.0.1   127.0.0.1    HTTP         437 GET /favicon.ico HTTP/1.1
   17 0.28350…  127.0.0.1   127.0.0.1    HTTP         553 HTTP/1.1 404 Not Found  (text/html)

HTTP/1.1 200 OK\r\n
  [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
      [HTTP/1.1 200 OK\r\n]
      [Severity level: Chat]
      [Group: Sequence]
    Response Version: HTTP/1.1
    Status Code: 200
    [Status Code Description: OK]
    Response Phrase: OK
  Date: Sun, 23 Oct 2022 13:46:42 GMT\r\n
  Server: Apache/2.4.41 (Ubuntu)\r\n
  Last-Modified: Sun, 23 Oct 2022 06:30:17 GMT\r\n
  ETag: "180-5ebadceec43f4-gzip"\r\n
  Accept-Ranges: bytes\r\n
  Vary: Accept-Encoding\r\n
  Content-Encoding: gzip\r\n
> Content-Length: 299\r\n
  Keep-Alive: timeout=5, max=100\r\n
  Connection: Keep-Alive\r\n
  Content-Type: text/html\r\n
  \r\n
  [HTTP response 1/3]
  [Time since request: 0.003079042 seconds]
  [Request in frame: 4]

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E·
0010  02 b0 d3 87 40 00 40 06  66 be 7f 00 00 01 7f 00   ····@·@· f·······
0020  00 01 00 50 a0 7e f3 3f  e2 70 9a 53 d4 ad 80 18   ···P·~·? ·p·S····
0030  02 00 00 a5 00 00 01 01  08 0a 52 e7 88 ab 52 e7   ········ ··R···R·
0040  88 a8 48 54 54 50 2f 31  2e 31 20 32 30 30 20 4f   ··HTTP/1 .1 200 O
```

响应报文由响应行（状态码和解释）、响应头、响应体组成。

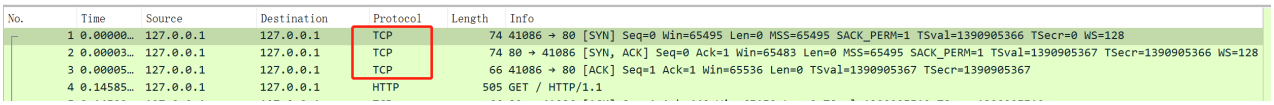
响应行给出响应报文的HTTP协议版本号、状态码、状态码的描述+回车换行符。

- 该报文中 状态码 为200 OK：表示请求成功，被请求的对象包含在该响应的数据部分。
- 在No.13的报文中， 状态码 则为304 Not Modified，表明此次请求为条件请求。由于在实验截图之前服务端已经有过访问Web服务器的经历，之后访问的时候，客户端缓存了目标资源但不确定该缓存资源是否是最新版本的时候, 就会发送一个条件请求。
- 在No.15的报文中，由于No.14请求的是favicon.ico，请求图标区分网站。由于本次实验的Web服务端并未搭建图标的信息，故返回的状态码为404 Not Found。

响应头则包含一些服务器用来处理请求的软件信息及其版本、连接方式等信息。

响应体：服务器返回给客户端的文本信息。

## 交互过程中**TCP**的三次握手

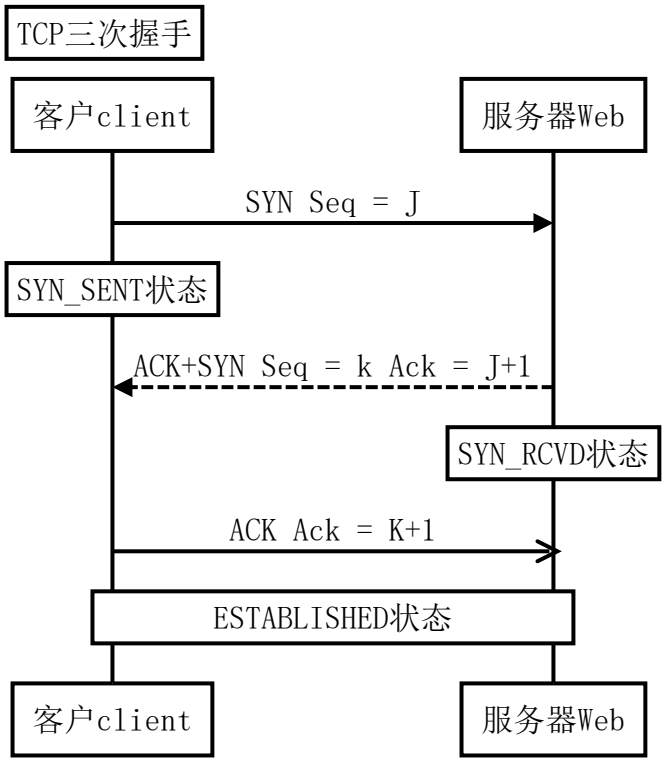


| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 0.00000… | 127.0.0.1 | 127.0.0.1 | TCP | 74 | 41086 → 80 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1390905366 TSecr=0 WS=128 |
| 2 | 0.00003… | 127.0.0.1 | 127.0.0.1 | TCP | 74 | 80 → 41086 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1390905367 TSecr=1390905366 WS=128 |
| 3 | 0.00005… | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 41086 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1390905367 TSecr=1390905367 |
| 4 | 0.14585… | 127.0.0.1 | 127.0.0.1 | HTTP | 505 | GET / HTTP/1.1 |

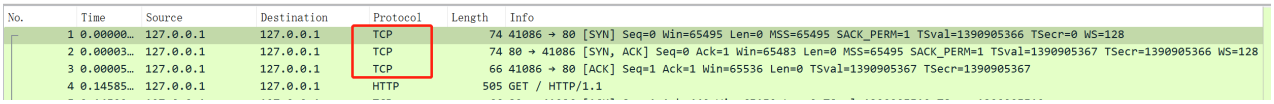
Wireshark捕获的数据包中可以看出在No.4：GET的HTTP请求报文之前，有过三次TCP的数据包传送。这三次TCP协议的数据包传递过程被称为**“TCP**三次握手**”**。

### **TCP**三次握手

过程分析：




1. 第一次握手：客户端发送 `syn` 包。Client将标志位SYN置为1，随机产生一个值seq=J，并将该数据包发送给Server，Client进入SYN_SENT状态，等待Server确认。
2. 第二次握手：服务器返回客户端 `SYN+ACK段` 。Server收到Client发来的 `SYN` 数据包后，由标志位SYN=1知道Client请求建立连接。Server将标志位SYN和ACK都置为1，ack=J+1，随机产生一个值seq=K，并将该数据包发送给Client以确认连接请求，Server进入SYN_RCVD状态。
3. 第三次握手：客户端收到服务器发送的 `SYN+ACK` 段，给服务器响应 `ACK` 段。Client收到确认后，检查ack是否为J+1，ACK是否为1，如果正确则将标志位ACK置为1，ack=K+1，并将该数据包发送给Server，Server检查正确则连接建立成功，Client和Server进入ESTABLISHED状态。



| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 0.00000… | 127.0.0.1 | 127.0.0.1 | TCP | 74 | 41086 → 80 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1390905366 TSecr=0 WS=128 |
| 2 | 0.00003… | 127.0.0.1 | 127.0.0.1 | TCP | 74 | 80 → 41086 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1390905367 TSecr=1390905366 WS=128 |
| 3 | 0.00005… | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 41086 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1390905367 TSecr=1390905367 |
| 4 | 0.14585… | 127.0.0.1 | 127.0.0.1 | HTTP | 505 | GET / HTTP/1.1 |

- 从本次Wireshark实际捕获到的信息来进行分析:

```
Transmission Control Protocol, Src Port: 41086, Dst Port: 80, Seq: 0, Len: 0
    Source Port: 41086
    Destination Port: 80
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 2589184757
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    1010 .... = Header Length: 40 bytes (10)
  > Flags: 0x002 (SYN)
    Window: 65495
    [Calculated window size: 65495]
```

```
Transmission Control Protocol, Src Port: 80, Dst Port: 41086, Seq: 0, Ack: 1, Len: 0
    Source Port: 80
    Destination Port: 41086
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 4081050223
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 2589184758
    1010 .... = Header Length: 40 bytes (10)
  > Flags: 0x012 (SYN, ACK)
    Window: 65483
    [Calculated window size: 65483]
```

```
Transmission Control Protocol, Src Port: 41086, Dst Port: 80, Seq: 1, Ack: 1, Len: 0
    Source Port: 41086
    Destination Port: 80
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 2589184758
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 4081050224
```

1. 客户端口41086向80发送一条SYN包，令seq=0。客户端client进入SYN_SENT状态。
2. 服务器80端口接收到SYN数据包后，确认SYN。令ack = 0+1 = 1，seq = 0。向client发送 SYN+ACK的数据包，Server进入SYN_RCVD状态。
3. 客户端收取到（SYN，ACK）数据包后，向服务端发送确认包，令ack = 0+1 = 1。连接建立好，Client和Server进入ESTABLISHED状态。

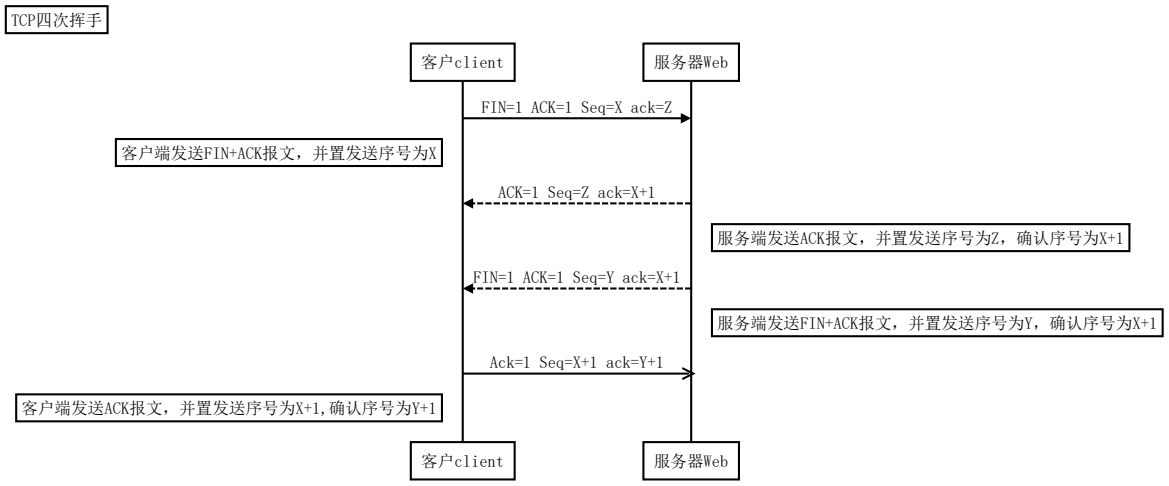
交互过程中**TCP**的四次挥手

**(**补充：由于实验第一次抓包过程未在意挥手断联的数据包。以下为后续二次捕获数据，故端口号出现不同**)**

```
    16 6.00283… 127.0.0.1       127.0.0.1       TCP             66 37576 → 80 [FIN, ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=22695529 TSecr=22689741
    17 6.00321… 127.0.0.1       127.0.0.1       TCP             66 80 → 37576 [FIN, ACK] Seq=1 Ack=2 Win=65536 Len=0 TSval=22695529 TSecr=22695529
    18 6.00324… 127.0.0.1       127.0.0.1       TCP             66 37576 → 80 [ACK] Seq=2 Ack=2 Win=65536 Len=0 TSval=22695529 TSecr=22695529
```

理论来说应该捕获四条TCP的数据包，可实际操作过程中Wireshark只捕获到断开连接过程中的三条数据包。（具体原因下文进行相关解释分析！）

过程分析：




1. 第一次挥手：Client发送一个FIN，用来关闭Client到Server的数据传送，Client进入FIN_WAIT_1状态。
2. 第二次挥手：Server收到FIN后，发送一个ACK给Client，确认序号为收到序号+1（与SYN相同，一个FIN占用一个序号），Server进入CLOSE_WAIT状态。
3. 第三次挥手：Server发送一个FIN，用来关闭Server到Client的数据传送，Server进入LAST_ACK状态。
4. 第四次挥手：Client收到FIN后，Client进入TIME_WAIT状态，接着发送一个ACK给Server，确认序号为收到序号+1，Server进入CLOSED状态，完成四次挥手。

```
16 6.00283… 127.0.0.1       127.0.0.1       TCP             66 37576 → 80 [FIN, ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=22695529 TSecr=22689741
17 6.00321… 127.0.0.1       127.0.0.1       TCP             66 80 → 37576 [FIN, ACK] Seq=1 Ack=2 Win=65536 Len=0 TSval=22695529 TSecr=22695529
18 6.00324… 127.0.0.1       127.0.0.1       TCP             66 37576 → 80 [ACK] Seq=2 Ack=2 Win=65536 Len=0 TSval=22695529 TSecr=22695529
```

- 从本次Wireshark实际捕获到的信息来进行分析:

```
Transmission Control Protocol, Src Port: 37576, Dst Port: 80, Seq: 1, Ack: 1, Len: 0
    Source Port: 37576
    Destination Port: 80
    [Stream index: 1]
    [TCP Segment Len: 0]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 1177801627
    [Next Sequence Number: 2    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 2652007561
```

```
Transmission Control Protocol, Src Port: 80, Dst Port: 37576, Seq: 1, Ack: 2, Len: 0
    Source Port: 80
    Destination Port: 37576
    [Stream index: 1]
    [TCP Segment Len: 0]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 2652007561
    [Next Sequence Number: 2    (relative sequence number)]
    Acknowledgment Number: 2    (relative ack number)
    Acknowledgment number (raw): 1177801628
```

```
Transmission Control Protocol, Src Port: 37576, Dst Port: 80, Seq: 2, Ack: 2, Len: 0
    Source Port: 37576
    Destination Port: 80
    [Stream index: 1]
    [TCP Segment Len: 0]
    Sequence Number: 2    (relative sequence number)
    Sequence Number (raw): 1177801628
    [Next Sequence Number: 2    (relative sequence number)]
    Acknowledgment Number: 2    (relative ack number)
    Acknowledgment number (raw): 2652007562
```

1. 客户端发起断开连接请求，向服务器发送FIN报文，指定序列号，进入 `FIN_WAIT_1` 状态。

2. 服务器发起断开连接请求，因为这里服务器端没有东西需要发送给客户端了，所以也要关闭连接，发送**FIN**信号。

   理论上四次挥手：服务器收到客户端发来的**FIN**报文后，应该先向客户端发送**ACK**应答报文，进入 `CLOSE_WAIT` 状态，TCP处在半关闭状态，客户端到服务器的连接释放。客户端收到来自服务器的ACK应答报文段后，进入 `FIN_WAIT_2` 状态。紧接之后，服务器也打算断开连接，向客户端发送**FIN**报文段，之后服务器进入**LASK_ACK(**最后确认**)**状态，等待客户端的确认。

3. 客户端收到来自服务器的连接释放**(FIN)**报文段后，会向服务器发送一个**ACK**应答报文段，以连接释放(FIN)报文段的确认序号 ack 作为ACK应答报文段的序列号 seq，以连接释放(FIN)报文段的序列号 seq+1作为确认序号ack。

三次挥手现象的发生是由于：客户端关闭连接，服务器**Web**没有数据发送时，会采取马上关闭连接的方式，也就将第二步的**ack**与第三步的**fin**合并为一步了。

# 具体TCP Wireshark抓包内容分析补充

以捕获的**TCP**挥手的数据包为例进行分析（挥手过程中服务端合并了**ACK**与**FIN**的数据包俩次发送

```
16 6.00283…  127.0.0.1        127.0.0.1        TCP              66 37576 → 80 [FIN, ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=22695529 TSecr=22689741
17 6.00321…  127.0.0.1        127.0.0.1        TCP              66 80 → 37576 [FIN, ACK] Seq=1 Ack=2 Win=65536 Len=0 TSval=22695529 TSecr=22695529
```

- 客户端发送FIN报文深层分析：

```
    16 6.00283…  127.0.0.1        127.0.0.1        TCP              66 37576 → 80 [FIN, ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=22695529 TSecr=22689741
Transmission Control Protocol, Src Port: 37576, Dst Port: 80, Seq: 1, Ack: 1, Len: 0
    Source Port: 37576
    Destination Port: 80
    [Stream index: 1]
    [TCP Segment Len: 0]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 1177801627
    [Next Sequence Number: 2    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 2652007561
    1000 .... = Header Length: 32 bytes (8)
  Flags: 0x011 (FIN, ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): 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
    › .... .... ...1 = Fin: Set
      [TCP Flags: ·····A···F]
    Window: 512
    [Calculated window size: 65536]
    [Window size scaling factor: 128]
```

> source port : 37576  源端口37576
> destination port : 80  目的端口80
> sequence number : 1  Seq = 1  发送序号为 1
> acknowledge number : 1  Ack = 1确认序号为 1
>
> Flags:
> Reserved 保留位：未设定，值全为零。预留给以后使用
> Nonce sum：该标签用来保护不受发送者发送的突发的恶意隐藏报文的侵害。
> Congestion Window Reduced：发送者在接收到一个带有ECE flag包时，将会使用CWR flag。
> ECN-Echo：ECN表示Explicit Congestion Notification。表示TCP peer有ECN能力。
> Urgent：通知接收端处理在处理其他包前优先处理接收到的紧急报文
> Acknowledgment：表示包已经被成功接收。
> Push：通知接收端处理接收的报文，而不是将报文缓存到buffer中。
> Reset：重置连接标志，用于重置由于主机崩溃或其他原因而出现错误的连接。
> (SYN)Synchronisation：表示三次握手建立连接的第一步，在建立连接时发送者发送的第一个包中设置flag值为SYN。
> (FIN)Finished：表示发送者以及发送完数据。通常用在发送者发送完数据的最后一个包中。

在上述实例中，源端口与目的端口表明了双方的端口信息，确定序号与发送序号蕴含了服务端下次回传信息的Ack的预期值。Acknowledgment与FIN被设置为1，表示该数据包为（FIN，ACK）数据包。表示客户段发送完数据，想要断开连接。

- 服务端发送FIN，ACK数据包


大体上的信息内容与客户端发送的类似，标志位都一样。需要注意的是，这里服务端收到FIN后，发送一个ACK给Client，确认序号为应该等于收到的FIN报文中的Seq序号+1！

至于未提及到的ACK、SYN数据包，它们的Flags的标识是不一样的，分别是为0X010、0X002。