

# 计算机网络实验报告

实验名称：配置Web服务器，编写简单页面，分析交互过程

学号：2012682          姓名：韩佳迅

计算机网络实验报告

实验名称：配置Web服务器，编写简单页面，分析交互过程

学号：2012682          姓名：韩佳迅

一、WEB服务器

二、Wireshark 抓包

(一) 使用流程

(二) 结果分析

- 1. Wireshark 面板结构
  - 2. TCP数据段格式
  - 3. TCP三次握手
    - (1) 第一次握手：
    - (2) 第二次握手
    - (3) 第三次握手
  - 4. HTTP报文格式
    - (1) 请求报文：向Web服务器请求一个动作
    - (2) 响应报文：将请求的结果返回给客户端
  - 5. HTTP请求应答
    - (1) 第一轮请求应答：  
客户端先向服务器端发送HTTP请求报文  
服务器端发送响应报文
    - (2) 第二轮请求应答：
  - 6. 四次挥手
    - (1) 第一次挥手
    - (2) 第二次挥手
    - (3) 第三次挥手
    - (4) 第四次挥手
- 拓展：三次握手时可能引起的SYN泛洪攻击


## 一、WEB服务器


本实验使用的是 node.js 中的 http-server，开启本地web服务器。

- 安装 node.js
- 在html文件所在的文件夹下，使用命令行输入http-server，即可开启服务。

« 实验 > Lab2 > myserver

名称

 index.html

 logo1.jpg

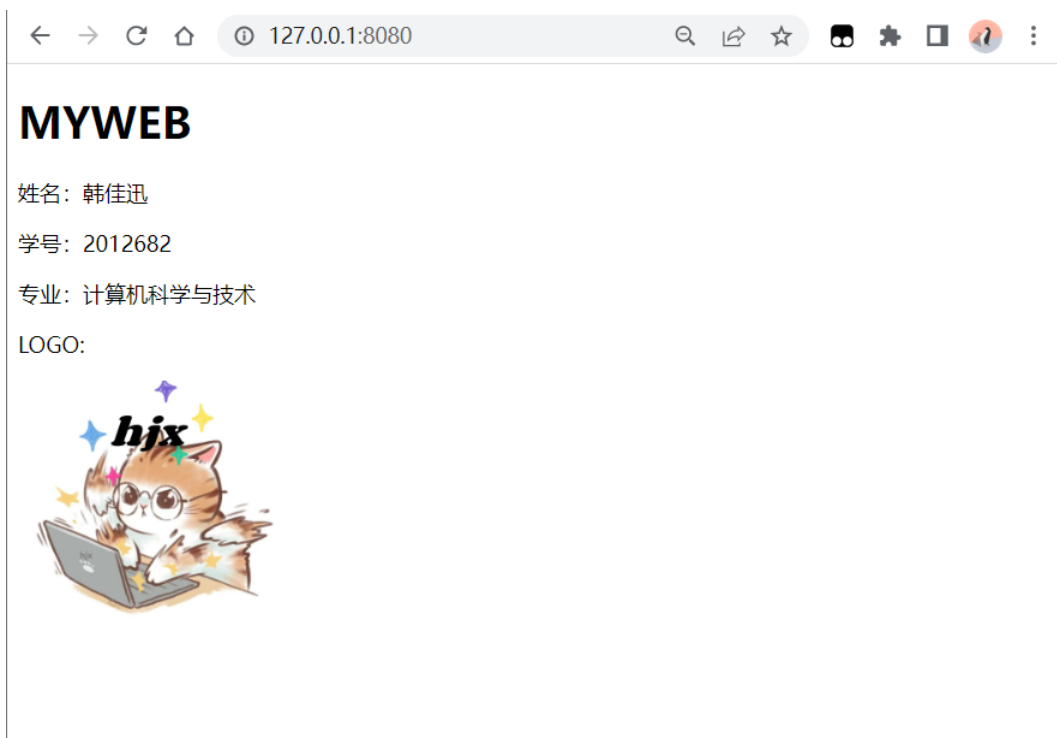
```
D:\learning\my_大三上\计算机网络\实验\Lab2\myserver>http-server
Starting up http-server, serving ./

http-server version: 14.1.1

http-server settings:
CORS: disabled
Cache: 3600 seconds
Connection Timeout: 120 seconds
Directory Listings: visible
AutoIndex: visible
Serve GZIP Files: false
Serve Brotli Files: false
Default File Extension: none

Available on:
  http://192.168.47.1:8080
  http://192.168.134.1:8080
  http://10.136.123.60:8080
  http://127.0.0.1:8080
Hit CTRL-C to stop the server
```

- 然后用浏览器打开我们的web服务器，在url输入 **127.0.0.1:8080**，如图：



## 二、Wireshark 抓包

### (一) 使用流程

- 打开 **Wireshark**，选择 **Adapter for loopback traffic capture** 进行捕获
- 设置过滤器，选择：
  - 目的ip、源ip = 127.0.0.1
  - 端口号 = 8080 和 52927（8080是确定的，52927是不确定的）
- 过滤出抓包结果如下：

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	52927 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
2	0.000060	127.0.0.1	127.0.0.1	TCP	56	8080 → 52927 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
3	0.000098	127.0.0.1	127.0.0.1	TCP	44	52927 → 8080 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
4	14.900775	127.0.0.1	127.0.0.1	HTTP	844	GET / HTTP/1.1
5	14.900822	127.0.0.1	127.0.0.1	TCP	44	8080 → 52927 [ACK] Seq=1 Ack=801 Win=2160384 Len=0
6	14.911704	127.0.0.1	127.0.0.1	HTTP	736	HTTP/1.1 200 OK (text/html)
7	14.911819	127.0.0.1	127.0.0.1	TCP	44	52927 → 8080 [ACK] Seq=801 Ack=693 Win=2160640 Len=0
8	14.919698	127.0.0.1	127.0.0.1	HTTP	626	GET /logo1.jpg HTTP/1.1
9	14.919735	127.0.0.1	127.0.0.1	TCP	44	8080 → 52927 [ACK] Seq=693 Ack=1383 Win=2159872 Len=0
10	14.922848	127.0.0.1	127.0.0.1	TCP	65539	8080 → 52927 [ACK] Seq=693 Ack=1383 Win=2159872 Len=65495 [TCP segment of a reassembled PDU]
11	14.922888	127.0.0.1	127.0.0.1	TCP	395	8080 → 52927 [PSH, ACK] Seq=66188 Ack=1383 Win=2159872 Len=351 [TCP segment of a reassembled PDU]
12	14.922962	127.0.0.1	127.0.0.1	TCP	44	52927 → 8080 [ACK] Seq=1383 Ack=66539 Win=2161152 Len=0
13	14.923231	127.0.0.1	127.0.0.1	TCP	65539	8080 → 52927 [ACK] Seq=66539 Ack=1383 Win=2159872 Len=65495 [TCP segment of a reassembled PDU]
14	14.923260	127.0.0.1	127.0.0.1	TCP	85	8080 → 52927 [PSH, ACK] Seq=132034 Ack=1383 Win=2159872 Len=41 [TCP segment of a reassembled PDU]
15	14.923299	127.0.0.1	127.0.0.1	TCP	44	52927 → 8080 [ACK] Seq=1383 Ack=132075 Win=2161152 Len=0
16	14.923466	127.0.0.1	127.0.0.1	TCP	65539	8080 → 52927 [ACK] Seq=132075 Ack=1383 Win=2159872 Len=65495 [TCP segment of a reassembled PDU]
17	14.923498	127.0.0.1	127.0.0.1	TCP	85	8080 → 52927 [PSH, ACK] Seq=197570 Ack=1383 Win=2159872 Len=41 [TCP segment of a reassembled PDU]
18	14.923535	127.0.0.1	127.0.0.1	TCP	44	52927 → 8080 [ACK] Seq=1383 Ack=197611 Win=2161152 Len=0
19	14.923798	127.0.0.1	127.0.0.1	TCP	65539	8080 → 52927 [ACK] Seq=197611 Ack=1383 Win=2159872 Len=65495 [TCP segment of a reassembled PDU]
20	14.923825	127.0.0.1	127.0.0.1	TCP	85	8080 → 52927 [PSH, ACK] Seq=263106 Ack=1383 Win=2159872 Len=41 [TCP segment of a reassembled PDU]
21	14.923861	127.0.0.1	127.0.0.1	TCP	44	52927 → 8080 [ACK] Seq=1383 Ack=263147 Win=2161152 Len=0
22	14.924187	127.0.0.1	127.0.0.1	HTTP	22379	HTTP/1.1 200 OK (JPEG JFIF image)
23	14.924225	127.0.0.1	127.0.0.1	TCP	44	52927 → 8080 [ACK] Seq=1383 Ack=285482 Win=2138880 Len=0
24	19.938846	127.0.0.1	127.0.0.1	TCP	44	8080 → 52927 [FIN, ACK] Seq=285482 Ack=1383 Win=2159872 Len=0
25	19.938881	127.0.0.1	127.0.0.1	TCP	44	52927 → 8080 [ACK] Seq=1383 Ack=285483 Win=2138880 Len=0
26	29.569502	127.0.0.1	127.0.0.1	TCP	44	52927 → 8080 [FIN, ACK] Seq=1383 Ack=285483 Win=2138880 Len=0
27	29.569661	127.0.0.1	127.0.0.1	TCP	44	8080 → 52927 [ACK] Seq=285483 Ack=1384 Win=2159872 Len=0

## (二) 结果分析

### 1. Wireshark 面板结构

```

> Frame 7: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface \Device\NPF_{Loopback}, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 52927, Dst Port: 8080, Seq: 801, Ack: 693, Len: 0

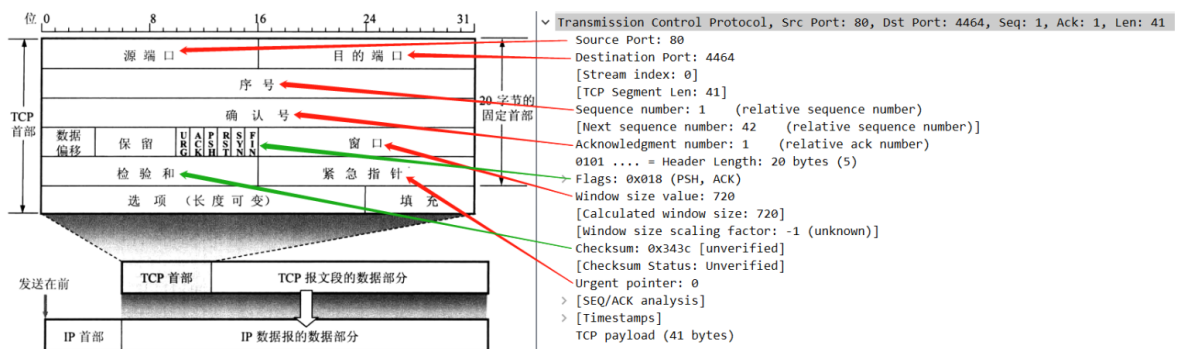
```

**Frame：**物理层的数据帧概况

**Internet Protocol Version 4：**互联网IP包头部信息

**Transmission Control Protocol：**传输层的数据段头部信息，此处是TCP

### 2. TCP数据段格式

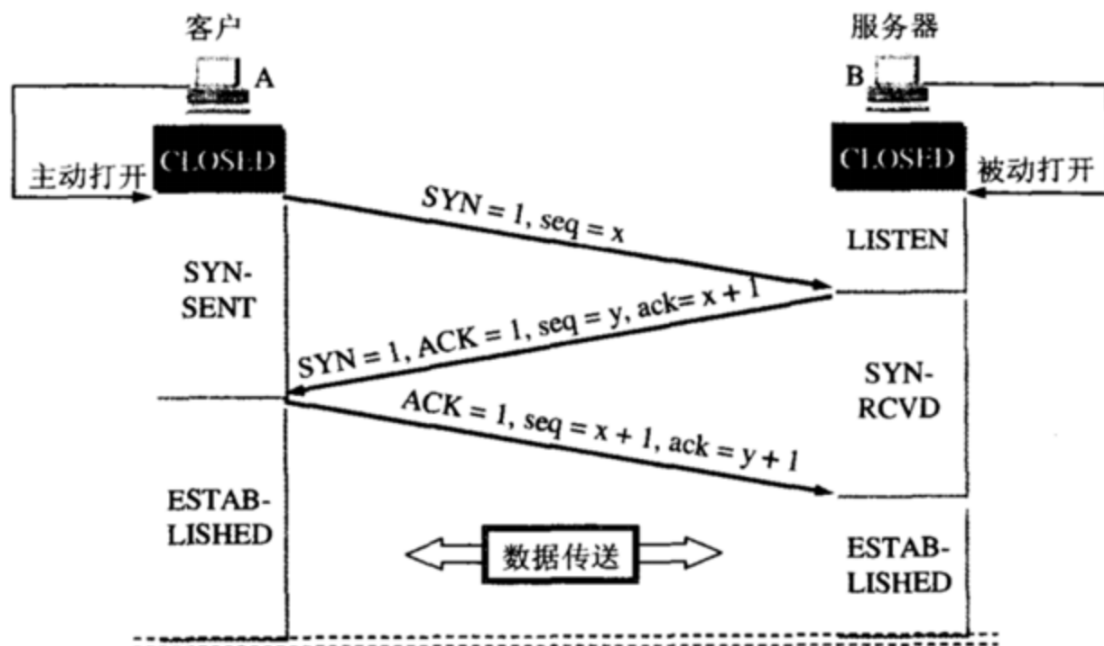


- 源端口： 16 bits，发送方的TCP端口号
- 目的端口： 16 bits，接收方的TCP端口号
- 序号： 32 bits，数据部分第一个字节的序列号
- 确认号： 32 bits，期望接收到对方下一个数据段中的数据部分的第一个字节序列号
- 数据偏移： 4bits，从数据段开头到第一个数据字节的长度，也就是TCP头部的长度
- 保留： 6 bits（未用）
- 标志位 Flags：
  - URG：1 bit，紧急指针控制位，置为1时表示有紧急数据。
  - ACK：1 bit，确认控制位，指示TCP数据段中的确认号字段是否有效，置为1时才表示有效。

- PSH: 1 bit, 推控制位, 指示是否需要立即把收到的该数据段提交给应用程序。置为1时要求接收端尽快把数据段提交给应用程序, 置为0时没有这个要求, 可以先缓存起来。
- RST: 1 bit, 重置控制位, 用于重置, 释放一个已经混乱的传输连接。
- SYN: 1 bit, 同步控制位, 用来在传输连接建立时同步传输连接序号。
- FIN: 1 bit, 最后控制位, 用于释放一个传输连接。
- 窗口大小: 16 bits, 指示发送此TCP数据段的主机上用来存储传入数据段的窗口大小, 也即发送者当前还可以接收的最大字节数
- 检验和: 校验和是指对数据段头, 数据和伪头部这三部分进行校验。
- 紧急指针: 16 bits, 仅当前面的URG控制位置为1时才有意义, 它指出本数据段中为紧急数据的字节数。当窗口大小为0时, 也可以发送紧急数据, 因为紧急数据无须缓存。
- 可选项: (数据偏移\*4 - 20) bits, 它可以包括窗口缩放选项, 最大段长度 (MSS), 选择性确认, 时间戳等。
- 数据: 应用层进程提交的数据, 作为TCP数据段的数据部分。

### 3. TCP三次握手

- TCP 建立连接的过程叫做**握手**, 握手需要在客户和服务端之间交换三个TCP 报文段, 称之为**三次握手**, 采用**三次握手**主要是为了防止已失效的连接请求报文段突然又传送到了, 因而产生错误。
  - 确认存在: 使得TCP双方能够知道对方的存在。
  - 协商参数: TCP双方协商一些参数。
  - 分配资源: 使得TCP双方能够对运输实体资源进行分配。
- 主动发起TCP连接建立称为**TCP客户**;  
被动等待TCP连接建立的应用进程称为**TCP服务器**。
- 三次握手示意图:



- 在 Wireshark 抓包文件里, 三次握手对应这三个包:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	52927 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
2	0.000060	127.0.0.1	127.0.0.1	TCP	56	8080 → 52927 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
3	0.000098	127.0.0.1	127.0.0.1	TCP	44	52927 → 8080 [ACK] Seq=1 Ack=1 Win=2161152 Len=0

## (1) 第一次握手:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	52927 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM

```
> Frame 1: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface \Device\NPF_{...}, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▼ Transmission Control Protocol, Src Port: 52927, Dst Port: 8080, Seq: 0, Len: 0
  Source Port: 52927
  Destination Port: 8080
  [Stream index: 0]
  [Conversation completeness: Complete, WITH_DATA (31)]
  [TCP Segment Len: 0]
  Sequence Number: 0 (relative sequence number)
  Sequence Number (raw): 1596886089
  [Next Sequence Number: 1 (relative sequence number)]
  Acknowledgment Number: 0
  Acknowledgment number (raw): 0
  1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x002 (SYN)
    Window: 65535
    [Calculated window size: 65535]
    Checksum: 0x9d22 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP), No-Operation (NOP), SACK permitted
  > [Timestamps]
```

从上图分析，我们主要关注这个数据包的以下值：

- Source Port: 52927 (客户端)
- Destination Port: 8080 (服务器)
- Sequence Number: 0
  - 这是relative相对的序列号
  - 实际序列号: 1596886089
- Flags: SYN (同步控制位，用来在传输连接建立时同步传输连接序号)

起初TCP两端都处于 CLOSED 关闭状态

(TCP服务器进程是被动等待来自TCP客户端进程的连接请求，因此称为被动打开连接)

(TCP客户进程主动发起建立连接，因此称为主动打开连接)

- TCP**客户**进程向TCP服务器进程发送**连接请求报文段**
  - 标志位: SYN 置为 1
  - seq num: 随机产生一个值 seq=x (x一般是随机数)，指定客户端的初始化序列号 (ISN)，在Wireshark中，其显示的是相对值，即0。
  - 无应用层数据
    - TCP规定SYN被设置为1的报文段不能携带数据但要消耗掉一个序号。
- 随后**客户端**进入 **SYN-SENT** 阶段，等待服务器端确认

## (2) 第二次握手

```
2 0.000060 127.0.0.1 127.0.0.1 TCP 56 8080 → 52927 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
> Frame 2: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface \Device\NPF_{...} id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
√ Transmission Control Protocol, Src Port: 8080, Dst Port: 52927, Seq: 0, Ack: 1, Len: 0
  Source Port: 8080
  Destination Port: 52927
  [Stream index: 0]
  [Conversation completeness: Complete, WITH_DATA (31)]
  [TCP Segment Len: 0]
  Sequence Number: 0 (relative sequence number)
  Sequence Number (raw): 839827435
  [Next Sequence Number: 1 (relative sequence number)]
  Acknowledgment Number: 1 (relative ack number)
  Acknowledgment number (raw): 1596886090
  1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x012 (SYN, ACK)
  Window: 65535
  [Calculated window size: 65535]
  Checksum: 0xab17 [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
  > Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP), No-Operation (NOP), SACK permitted
  > [Timestamps]
  > [SEQ/ACK analysis]
```

从上图分析，我们主要关注这个数据包的以下值：

- Source Port: 8080 (服务器)
- Destination Port: 52927 (客户端)
- Sequence Number: 0
  - 这是relative相对的序列号，虽然为0，但是实际值和第一次握手的实际序列号不同。
  - 实际序列号: 839827435
- Acknowledgment Number: 1
  - 这个也是相对值
  - 实际值: 1596886090
- Flags: ACK (确认控制位)、SYN (同步控制位，用来在传输连接建立时同步传输连接序号)
- 服务器端为该TCP连接分配缓存和变量
- TCP**服务器**进程收到数据包后由标志位 SYN=1 得知客户端请求建立连接，向TCP客户进程发送**确认报文段**:
  - 标志位: SYN 和 ACK 都置为 1
  - seq num: 随机产生一个值 seq=y (y一般是随机数) 指定服务器端的初始化序列号 (ISN)，在Wireshark中，显示的是相对值，即0。

**注意:** 虽然这里的 seq num 和第一次握手的 seq num 都显示为 0，但这只是显示了相对值，其实际值并不相等 (本实验中，第一次握手的 seq = 1596886089，第二次的 seq = 839827435)
  - ack num: 客户端所发送的 seq num + 1 (= x + 1)
    - 这是对TCP**客户**进程所选择的初始序号的确认。
    - 加1是因为第一次握手发送的连接请求报文段没有数据部分，所以服务器端期望收到客户端数据部分的下一个字节序列号只用加1即可。
    - 本实验中，第二次握手的 ack 实际值 = 1596886090 = 客户端发送的 seq num + 1 = 1596886089 + 1
  - 无应用层数据
    - TCP规定SYN被设置为1的报文段不能携带数据但要消耗掉一个序号。
- 此后，**服务器端**进入 **SYN-RCVD** 状态。

### (3) 第三次握手

3 0.000098	127.0.0.1	127.0.0.1	TCP	44 52927 → 8080 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
> Frame 3: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface \Device\NPF_{Loopback}, id 0				
> Null/Loopback				
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1				
√ Transmission Control Protocol, Src Port: 52927, Dst Port: 8080, Seq: 1, Ack: 1, Len: 0				
Source Port: 52927				
Destination Port: 8080				
[Stream index: 0]				
[Conversation completeness: Complete, WITH_DATA (31)]				
[TCP Segment Len: 0]				
Sequence Number: 1 (relative sequence number)				
Sequence Number (raw): 1596886090				
[Next Sequence Number: 1 (relative sequence number)]				
Acknowledgment Number: 1 (relative ack number)				
Acknowledgment number (raw): 839827436				
0101 .... = Header Length: 20 bytes (5)				
> Flags: 0x010 (ACK)				
Window: 8442				
[Calculated window size: 2161152]				
[Window size scaling factor: 256]				
Checksum: 0xc514 [unverified]				
[Checksum Status: Unverified]				
Urgent Pointer: 0				
> [Timestamps]				
> [SEQ/ACK analysis]				

从上图分析，我们主要关注这个数据包的以下值：

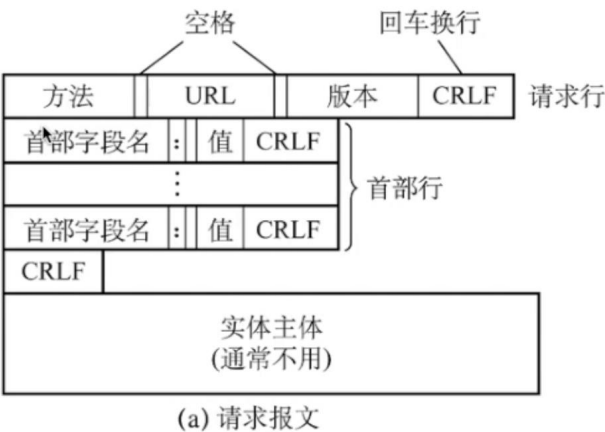
- Source Port: 52927 (客户端)
- Destination Port: 8080 (服务器)
- Sequence Number: 1 (相对值)
  - 实际序列号: 1596886090
- Acknowledgment Number: 1 (相对值)
  - 实际值: 839827436
- Flags: ACK (确认控制位)
- 客户端为该TCP连接分配缓存和变量
- TCP**客户**进程收到确认后，检查 ack 是否为 x+1，ACK 是否为 1，如果正确则再发送一个确认报文段给TCP服务器进程：
  - 标志位: ACK 置为 1
  - seq num: 置为 x + 1
    - 加1是因为就TCP**客户**进程而言，其发送的第一个TCP报文段的seq num为x，并且不携带数据，因此第二个报文段的序号为x + 1 (也就是数据部分第一个字节的序列号)
  - ack num: 服务器端所发送的 seq num + 1 (= y + 1)
    - 这是对TCP**服务器**进程所选择的初始序号的确认。
    - 本实验中，第三次握手的 ack 实际值 = 839827436 = 服务器端发送的 seq num + 1 = 839827435 + 1
  - 可以携带数据
- **客户端**进入 ESTABLISHED 状态
- **服务器端**收到这个包，检查 ack 是否为 y+1，ACK 是否为 1，如果正确则连接建立成功，进入 ESTABLISHED 状态，完成三次握手，随后 Client 和 Server 就可以开始传输数据。

## 4. HTTP报文格式

**HTTP报文：**是HTTP应用程序之间发送的数据块。这些数据块以一些文本形式的元信息开头，这些信息描述了报文的内容及含义，后面跟着可选的数据部分。这些报文都是在客户端、服务器和代理之间流动。

**HTTP报文的流动方向：**一次HTTP请求，HTTP报文会从“客户端”到“服务器”，在服务器工作完成之后，报文又会从“服务器”流到“客户端”。

### (1) 请求报文：向Web服务器请求一个动作



#### ① 请求行

请求行包括：

1. 方法：常见的有get, post, put, delete
2. 路径：服务器地址后面的部分
3. http的版本

例如：GET /users HTTP/1.1

#### ② 请求头 Headers

为请求报文添加了一些附加信息，由“名/值”对组成，每行一对，名和值之间使用冒号分隔

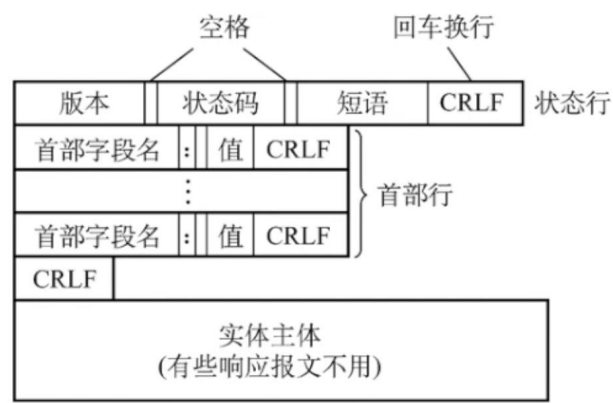
请求头	说明
Host	接受请求的服务器地址，可以是IP:端口号，也可以是域名
User-Agent	发送请求的应用程序名称
Connection	指定与连接相关的属性，如Connection:Keep-Alive
Accept-Charset	通知服务端可以发送的编码格式
Accept-Encoding	通知服务端可以发送的数据压缩格式
Accept-Language	通知服务端可以发送的语言

#### ③ 主体

报文主体包含了要发给服务器的数据，是主要数据部分，它可以是任意数据类型的数据。



(2) 响应报文：将请求的结果返回给客户端



(b) 响应报文

① 状态行

状态包括三个部分：

- 1. http版本：常用的是1.1
- 2. 状态码：
  - 200：响应成功
  - 304：缓存文件未过期，还可以继续使用，无需向服务器端获取
  - 400：客户端语法错误，无法被服务器识别
  - 403：请求失败
  - 404：请求资源不存在
  - 500：服务器内部错误
- 3. 状态信息

例如：HTTP/1.1 200 OK

② 响应头部

与请求头部类似，为响应报文添加了一些附加信息

响应头	说明
Server	服务器应用程序软件的名称和版本
Content-Type	响应正文的类型（是图片还是二进制字符串）
Content-Length	响应正文长度
Content-Charset	响应正文使用的编码
Content-Encoding	响应正文使用的数据压缩格式
Content-Language	响应正文使用的语言

③ 主体

服务器发送给客户端的内容，也是客户端所请求的内容。

例如：



- Accept: 客户可以识别的内容类型列表
- Accept-Encoding: 客户可识别的数据编码
- Accept-Language: 浏览器支持的语言类型

**HTTP请求报文的TCP端: PSH、ACK**

然后，服务器端回复ACK表示收到。

## 服务器端发送响应报文

```

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
  accept-ranges: bytes\r\n
  cache-control: max-age=3600\r\n
  last-modified: Mon, 24 Oct 2022 11:45:37 GMT\r\n
  etag: W/"1407374883590725-374-2022-10-24T11:45:37.254Z"\r\n
> content-length: 374\r\n
  content-type: text/html; charset=UTF-8\r\n
  Date: Mon, 24 Oct 2022 11:46:32 GMT\r\n
  Connection: keep-alive\r\n
  Keep-Alive: timeout=5\r\n
\r\n
[HTTP response 1/2]
[Time since request: 0.010929000 seconds]
[Request in frame: 4]
[Next request in frame: 8]
[Next response in frame: 22]
[Request URI: http://127.0.0.1:8080/]
File Data: 374 bytes
v Line-based text data: text/html (27 lines)
<!DOCTYPE html>\r\n
<html>\r\n
<head>\r\n
  <title>\r\n
    mypage\r\n
  </title>\r\n
</head>\r\n
</html>\r\n
\r\n

```

- 状态行:
  - HTTP版本: 1.1
  - 状态码: 200 (请求成功)
  - 状态信息
- 响应头:
  - cache-control: 缓存机制
  - last-modified: 请求资源的最后修改时间
  - content-length: 消息主体的大小
  - content-type: 告诉客户端实际返回的内容类型
  - connection: 连接状态为保持连接
  - File Data: 响应报文的大小
- 主体:
 

Line-based text data: text/html (27lines)

自己写的html文档

**HTTP响应报文的TCP端: PSH、ACK**

然后，客户端回复ACK表示收到。

## (2) 第二轮请求应答：

- 与第一轮类似，不过这次请求和应答的是图片 JPG 文件
- 与第一轮不同的是，HTTP响应报文没有一次性全部传过去，而是分成了多个段：
  - 基于TCP在传输消息时，对于上面的应用层如果出于某些原因（如超过MSS）TCP Segment不能一次包含全部的应用层PDU，而要把一个完整消息分成多个段，就会将除了最后一个分段（segment）的所有其他分段都打上 **TCP segment of a reassembled PDU**。
  - 使用PSH标识：当接收段收到PSH=1的报文段，就尽快地交付给接收应用进程,而不再等到整个缓冲都填满了再向上交互

### 补充：传输过程中的seq和ack：

5	14.900822	127.0.0.1	127.0.0.1	TCP	44 8080 → 52927 [ACK] Seq=1 Ack=801 Win=2160384 Len=0
6	14.911704	127.0.0.1	127.0.0.1	HTTP	736 HTTP/1.1 200 OK (text/html)
7	14.911819	127.0.0.1	127.0.0.1	TCP	44 52927 → 8080 [ACK] Seq=801 Ack=693 Win=2160640 Len=0
8	14.919698	127.0.0.1	127.0.0.1	HTTP	626 GET /Logo1.jpg HTTP/1.1
9	14.919735	127.0.0.1	127.0.0.1	TCP	44 8080 → 52927 [ACK] Seq=693 Ack=1383 Win=2159872 Len=0
10	14.922848	127.0.0.1	127.0.0.1	TCP	65539 8080 → 52927 [ACK] Seq=693 Ack=1383 Win=2159872 Len=65495 [TCP segment of a reassembled PDU]
11	14.922888	127.0.0.1	127.0.0.1	TCP	395 8080 → 52927 [PSH, ACK] Seq=66188 Ack=1383 Win=2159872 Len=351 [TCP segment of a reassembled PDU]
12	14.922962	127.0.0.1	127.0.0.1	TCP	44 52927 → 8080 [ACK] Seq=1383 Ack=66539 Win=2161152 Len=0
13	14.923231	127.0.0.1	127.0.0.1	TCP	65539 8080 → 52927 [ACK] Seq=66539 Ack=1383 Win=2159872 Len=65495 [TCP segment of a reassembled PDU]
14	14.923260	127.0.0.1	127.0.0.1	TCP	85 8080 → 52927 [PSH, ACK] Seq=132034 Ack=1383 Win=2159872 Len=41 [TCP segment of a reassembled PDU]
15	14.923299	127.0.0.1	127.0.0.1	TCP	44 52927 → 8080 [ACK] Seq=1383 Ack=132075 Win=2161152 Len=0 132075 + 65495 = 197570
16	14.923466	127.0.0.1	127.0.0.1	TCP	65539 8080 → 52927 [ACK] Seq=132075 Ack=1383 Win=2159872 Len=65495 [TCP segment of a reassembled PDU]
17	14.923498	127.0.0.1	127.0.0.1	TCP	85 8080 → 52927 [PSH, ACK] Seq=197570 Ack=1383 Win=2159872 Len=41 [TCP segment of a reassembled PDU]
18	14.923535	127.0.0.1	127.0.0.1	TCP	44 52927 → 8080 [ACK] Seq=1383 Ack=197611 Win=2161152 Len=0 197570 + 41 = 197611
19	14.923798	127.0.0.1	127.0.0.1	TCP	65539 8080 → 52927 [ACK] Seq=197611 Ack=1383 Win=2159872 Len=65495 [TCP segment of a reassembled PDU]
20	14.923825	127.0.0.1	127.0.0.1	TCP	85 8080 → 52927 [PSH, ACK] Seq=263106 Ack=1383 Win=2159872 Len=41 [TCP segment of a reassembled PDU]
21	14.923861	127.0.0.1	127.0.0.1	TCP	44 52927 → 8080 [ACK] Seq=1383 Ack=263147 Win=2161152 Len=0
22	14.924187	127.0.0.1	127.0.0.1	HTTP	22379 HTTP/1.1 200 OK (JPEG JFIF image)
23	14.924225	127.0.0.1	127.0.0.1	TCP	44 52927 → 8080 [ACK] Seq=1383 Ack=285482 Win=2138880 Len=0
24	19.938846	127.0.0.1	127.0.0.1	TCP	44 8080 → 52927 [FIN, ACK] Seq=285482 Ack=1383 Win=2159872 Len=0
25	19.938881	127.0.0.1	127.0.0.1	TCP	44 52927 → 8080 [ACK] Seq=1383 Ack=285483 Win=2138880 Len=0
26	29.569502	127.0.0.1	127.0.0.1	TCP	44 52927 → 8080 [FIN, ACK] Seq=1383 Ack=285483 Win=2138880 Len=0
27	29.569661	127.0.0.1	127.0.0.1	TCP	44 8080 → 52927 [ACK] Seq=285483 Ack=1384 Win=2159872 Len=0

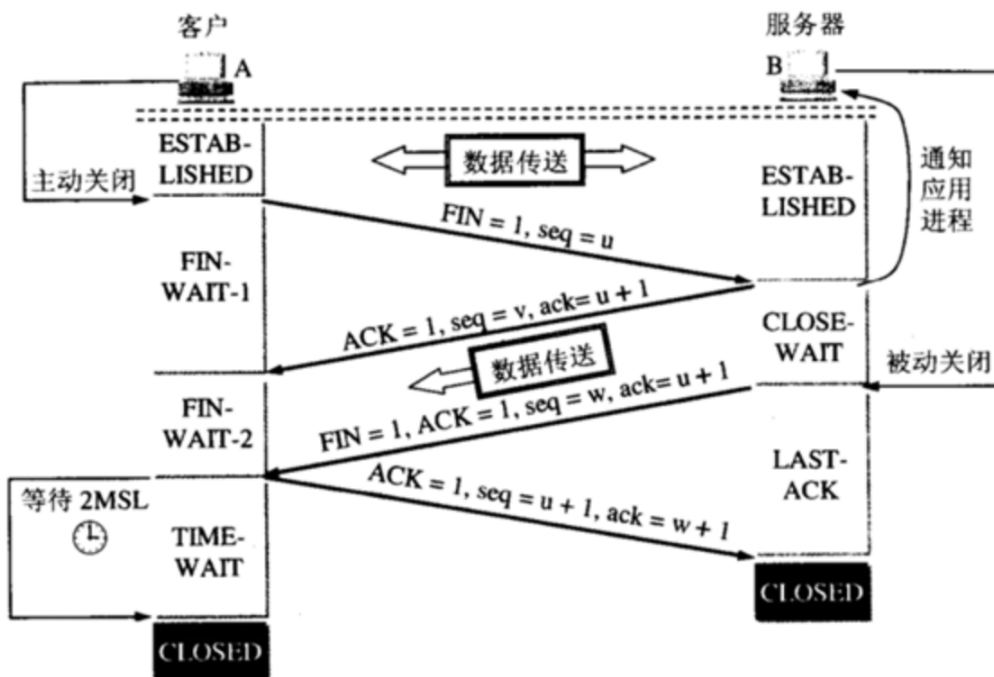
如上图：

- 蓝色框起来的部分：  
当 A -> B 端的数据分两次发送时，第二次发送的seq = 第一次发送的 seq + len  
A -> B 的 ack = 上一报文中 B -> A 的 seq + len
- 黄色框起来的部分：  
A -> B 的 seq = 上一个报文中 B -> A 的 ack
- 这都是因为 seq 表示数据部分第一个字节的序列号，ack 表示期望接收到对方下一个数据段的第一个字节序列号。且上面的传输过程没有失序、丢失等异常问题。

## 6. 四次挥手

24	19.938846	127.0.0.1	127.0.0.1	TCP	44 8080 → 52927 [FIN, ACK] Seq=285482 Ack=1383 Win=2159872 Len=0
25	19.938881	127.0.0.1	127.0.0.1	TCP	44 52927 → 8080 [ACK] Seq=1383 Ack=285483 Win=2138880 Len=0
26	29.569502	127.0.0.1	127.0.0.1	TCP	44 52927 → 8080 [FIN, ACK] Seq=1383 Ack=285483 Win=2138880 Len=0
27	29.569661	127.0.0.1	127.0.0.1	TCP	44 8080 → 52927 [ACK] Seq=285483 Ack=1384 Win=2159872 Len=0

- TCP释放连接的过程叫做**挥手**，断开连接需要发送四个包。由于TCP连接是全双工的，因此每个方向都必须单独进行关闭。
- 参与TCP连接的两个进程中的任何一个**都可以终止连接**。连接结束后，主机中的资源（缓存和变量）将被释放。在我们的抓包文件中，是服务器发起的终止连接。
- 四次握手示意图（这里是以客户端发起的终止连接）：



## (1) 第一次挥手

```

24 19.938846 127.0.0.1 127.0.0.1 TCP 44 8080 → 52927 [FIN, ACK] Seq=285482 Ack=1383 Win=2159872 Len=0
> Frame 24: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface \Device\NPF_{...}, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 8080, Dst Port: 52927, Seq: 285482, Ack: 1383, Len: 0
  Source Port: 8080
  Destination Port: 52927
  [Stream index: 0]
  [Conversation completeness: Complete, WITH_DATA (31)]
  [TCP Segment Len: 0]
  Sequence Number: 285482 (relative sequence number)
  Sequence Number (raw): 840112917
  [Next Sequence Number: 285483 (relative sequence number)]
  Acknowledgment Number: 1383 (relative ack number)
  Acknowledgment number (raw): 1596887472
  0101 .... = Header Length: 20 bytes (5)
> Flags: 0x011 (FIN, ACK)
  Window: 8437
  [Calculated window size: 2159872]
  [Window size scaling factor: 256]
  Checksum: 0x6485 [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
> [Timestamps]

```

从上图分析，我们主要关注这个数据包的以下值：

- Source Port: 8080 (服务器)
- Destination Port: 52927 (客户端)
- Sequence Number: 285482 (相对值)
  - 实际序列号: 840112917
- Acknowledgment Number: 1383 (相对值)
  - 实际值: 1596887472
- Flags: ACK (确认控制位)、FIN
- A 向 B 发送连接释放报文段，主动关闭TCP连接：
  - 标志位: FIN 置为 1
  - seq num: u
- A 进入FIN\_WAIT\_1状态，停止发送数据，等待客户端回复。

## (2) 第二次挥手

25	19.938881	127.0.0.1	127.0.0.1	TCP	44	52927 → 8080 [ACK] Seq=1383 Ack=285483 Win=2138880 Len=0
> Frame 25: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface \Device\NPF_{Loopback}, id 0						
> Null/Loopback						
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1						
√ Transmission Control Protocol, Src Port: 52927, Dst Port: 8080, Seq: 1383, Ack: 285483, Len: 0						
Source Port: 52927						
Destination Port: 8080						
[Stream index: 0]						
[Conversation completeness: Complete, WITH_DATA (31)]						
[TCP Segment Len: 0]						
Sequence Number: 1383 (relative sequence number)						
Sequence Number (raw): 1596887472						
[Next Sequence Number: 1383 (relative sequence number)]						
Acknowledgment Number: 285483 (relative ack number)						
Acknowledgment number (raw): 840112918						
0101 .... = Header Length: 20 bytes (5)						
> Flags: 0x010 (ACK)						
Window: 8355						
[Calculated window size: 2138880]						
[Window size scaling factor: 256]						
Checksum: 0x64d7 [unverified]						
[Checksum Status: Unverified]						
Urgent Pointer: 0						
> [Timestamps]						
> [SEQ/ACK analysis]						

从上图分析，我们主要关注这个数据包的以下值：

- Source Port: 52927 (客户端)
- Destination Port: 8080 (服务器)
- Sequence Number: 1383 (相对值)
  - 实际序列号: 1596887472
- Acknowledgment Number: 285483 (相对值)
  - 实际值: 840112918
- Flags: ACK
- **B** 回复确认报文段：
  - 标志位: ACK 置为 1
  - seq num: v
  - ack num: u+1
    - 是对 **A** 端发来的连接释放报文段的回复
- **B** 进入CLOSE-WAIT关闭等待状态
- **A** 到 **B** 方向的连接就释放了，处于半关闭状态
- 当 **A** 收到这个确认报文段之后，进入FIN-WAIT-2 状态

## (3) 第三次挥手

```

26 29.569502      127.0.0.1      127.0.0.1      TCP      44 [52927 → 8080 [FIN, ACK] Seq=1383 Ack=285483 Win=2138880 Len=0
> Frame 26: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface \Device\NPF_{Loopback, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▼ Transmission Control Protocol, Src Port: 52927, Dst Port: 8080, Seq: 1383, Ack: 285483, Len: 0
    Source Port: 52927
    Destination Port: 8080
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 1383      (relative sequence number)
    Sequence Number (raw): 1596887472
    [Next Sequence Number: 1384      (relative sequence number)]
    Acknowledgment Number: 285483      (relative ack number)
    Acknowledgment number (raw): 840112918
    0101 .... = Header Length: 20 bytes (5)
> Flags: 0x011 (FIN, ACK)
    Window: 8355
    [Calculated window size: 2138880]
    [Window size scaling factor: 256]
    Checksum: 0x64d6 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
> [Timestamps]

```

从上图分析，我们主要关注这个数据包的以下值：

- Source Port: 8080 (服务器)
- Destination Port: 52927 (客户端)
- Sequence Number: 285483 (相对值)
  - 实际序列号: 840112918
- Acknowledgment Number: 1384 (相对值)
  - 实际值: 1596887473
- Flags: ACK
- **A** 回复一个确认报文段：
  - 标志位: ACK 置为 1
  - seq num:  $u + 1$
  - ack num:  $w + 1$ 
    - 是对 **B** 端发来的连接释放报文段的回复
- **A** 再等待 2 msl 之后，彻底关闭连接

## 拓展：三次握手时可能引起的SYN泛洪攻击

TCP SYN泛洪发生在OSI第四层，这种方式利用TCP协议的三次握手。攻击者发送TCP SYN，SYN是TCP三次握手中的第一个数据包，而当服务器返回ACK后，该攻击者就不对其进行再确认，那这个TCP连接就处于挂起状态，也就是所谓的半连接状态，服务器收不到再确认的话，还会重复发送ACK给攻击者。这样更加会浪费服务器的资源。攻击者就对服务器发送非常大量的这种TCP连接，由于每一个都没法完成三次握手，所以在服务器上，这些TCP连接会因为挂起状态而消耗CPU和内存，最后服务器可能死机，就无法为正常用户提供服务了。