实验二：分析浏览器与Web服务器的交互过程

物联网工程_2111194_胡博程

一、实验要求

摘自学院网站。

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）、自己的LOGO、自我介绍的音频信息。页面不要太复杂，包含要求的基本信息即可。
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. 使用HTTP，不要使用HTTPS。
4. 提交实验报告。

二、web服务器搭建与页面编写

1、web服务器搭建

使用web服务器`http-server`，需要提前下载`nodejs`和`npm`

在cmd中使用命令`npm install -g http-server`安装`http-server`

输入命令`http-server --version`，正常显示版本，说明安装成功

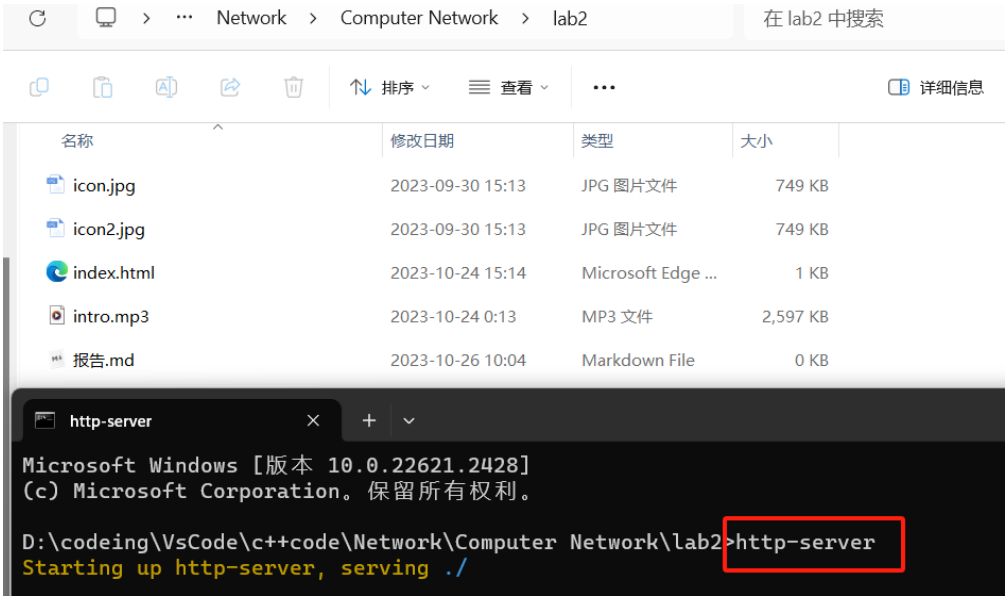
2、web页面编写

编写一个简单的HTML文件`index.html`，包含文本信息、图片资源和音频资源，源码如下。

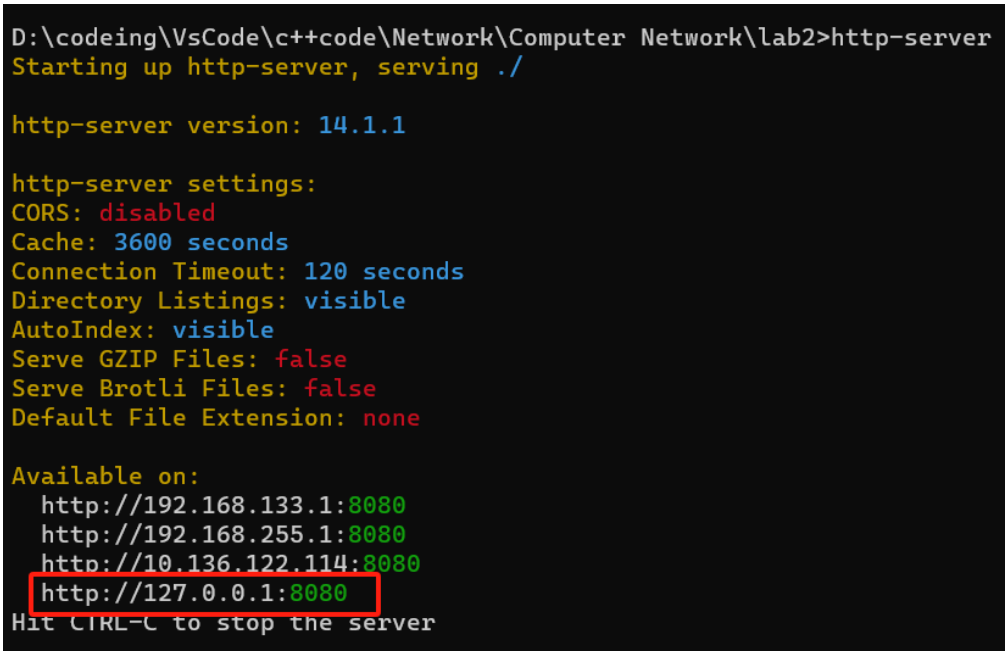
```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>个人介绍</title>
    <link rel="icon" href="icon2.jpg" type="image/jpeg">
</head>

<body>
    <h1>我的个人介绍</h1>
    <p>专业：物联网工程</p>
    <p>学号：2111194</p>
    <p>姓名：胡博程</p>
    <img src="icon.jpg" alt="我的LOGO" width="300">
    <audio controls>
        <source src="intro.mp3" type="audio/mp3">
        你的浏览器不支持音频标签。
```

```
    </audio>
</body>

</html>
```

在index.html文件所在目录下打开cmd，并输入命令`http-server`打开web服务器



命令行输出提示信息并给出可用的IP地址与端口



使用本地回环地址127.0.0.1的8080端口，在浏览器中打开网页，呈现效果如下。

我的个人介绍

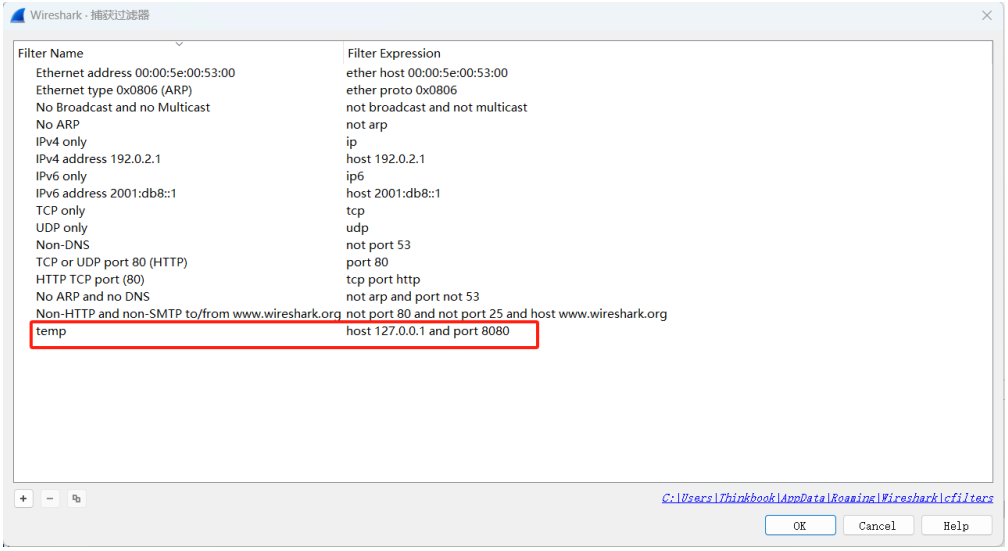
专业：物联网工程

学号：2111194

姓名：胡博程

三、wireshark抓包操作流程

- 打开wireshark，选择 Adapter for loopback traffic capture （对应本地回环地址127.0.0.1）进行捕获
- 使用捕获过滤器过滤出我们想用的数据包
  - 需要手动添加捕获过滤器规则
  - 随后将新的捕获规则应用于所选接口



> 最好是使用浏览器的无痕浏览模式，这样浏览器在推出后会自动删除关于已访问网站的缓存信息，保证每次抓包都是从本机获取资源，让我们可以分析交互过程。

- 开始捕获数据包
- 打开http-server服务，并用浏览器访问访问 `127.0.0.1:8080`
- 数据包不再变化，停止捕获并保存数据包

四、wireshark数据包分析

数据包捕获概览如下

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	50417 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
2	0.000058	127.0.0.1	127.0.0.1	TCP	56	8080 → 50417 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
3	0.000104	127.0.0.1	127.0.0.1	TCP	44	50417 → 8080 [ACK] Seq=1 Ack=1 Win=327424 Len=0
4	0.000472	127.0.0.1	127.0.0.1	HTTP	702	GET / HTTP/1.1
5	0.000503	127.0.0.1	127.0.0.1	TCP	44	8080 → 50417 [ACK] Seq=1 Ack=659 Win=2160640 Len=0
6	0.017604	127.0.0.1	127.0.0.1	HTTP	983	HTTP/1.1 200 OK (text/html)
7	0.017634	127.0.0.1	127.0.0.1	TCP	44	50417 → 8080 [ACK] Seq=659 Ack=940 Win=326400 Len=0
8	0.028542	127.0.0.1	127.0.0.1	HTTP	625	GET /icon.jpg HTTP/1.1
9	0.028589	127.0.0.1	127.0.0.1	TCP	44	8080 → 50417 [ACK] Seq=940 Ack=1240 Win=2159872 Len=0
10	0.034219	127.0.0.1	127.0.0.1	TCP	65539	8080 → 50417 [ACK] Seq=940 Ack=1240 Win=2159872 Len=65495 [TCP segment of a reassembled PDU]
11	0.034240	127.0.0.1	127.0.0.1	TCP	418	8080 → 50417 [PSH, ACK] Seq=66435 Ack=1240 Win=2159872 Len=374 [TCP segment of a reassembled PDU]
12	0.034301	127.0.0.1	127.0.0.1	TCP	44	50417 → 8080 [ACK] Seq=1240 Ack=66809 Win=327424 Len=0
13	0.034561	127.0.0.1	127.0.0.1	TCP	56	50418 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
14	0.034614	127.0.0.1	127.0.0.1	TCP	56	8080 → 50418 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
15	0.034651	127.0.0.1	127.0.0.1	TCP	44	50418 → 8080 [ACK] Seq=1 Ack=1 Win=327424 Len=0
16	0.034706	127.0.0.1	127.0.0.1	TCP	65539	8080 → 50417 [ACK] Seq=66809 Ack=1240 Win=2159872 Len=65495 [TCP segment of a reassembled PDU]
17	0.034715	127.0.0.1	127.0.0.1	TCP	85	8080 → 50417 [PSH, ACK] Seq=132304 Ack=1240 Win=2159872 Len=41 [TCP segment of a reassembled PDU]
18	0.034773	127.0.0.1	127.0.0.1	TCP	44	50417 → 8080 [ACK] Seq=1240 Ack=132345 Win=327424 Len=0
19	0.035354	127.0.0.1	127.0.0.1	HTTP	584	GET /intro.mp3 HTTP/1.1
20	0.035385	127.0.0.1	127.0.0.1	TCP	44	8080 → 50418 [ACK] Seq=1 Ack=541 Win=2160640 Len=0
21	0.035534	127.0.0.1	127.0.0.1	TCP	65539	8080 → 50417 [ACK] Seq=132345 Ack=1240 Win=2159872 Len=65495 [TCP segment of a reassembled PDU]
22	0.035543	127.0.0.1	127.0.0.1	TCP	85	8080 → 50417 [PSH, ACK] Seq=197840 Ack=1240 Win=2159872 Len=41 [TCP segment of a reassembled PDU]

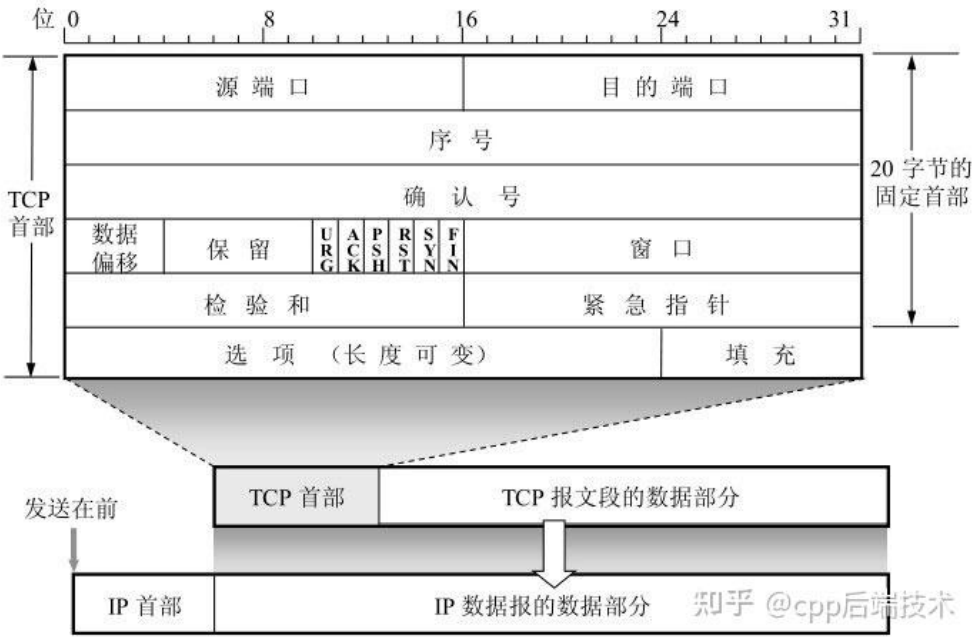
通览数据包，可以得出Wireshark捕获浏览器与Web服务器的**交互过程如下**

TCP协议三次握手建立连接----->客户端使用GET请求获取网页内容----->服务器向客户端发送HTTP响应状态码（200or206）----->四次挥手关闭连接

接下来逐个分析数据包来理解本机打开web页面的时候与web服务器的交互过程。

1、TCP头部与http报文格式分析

TCP 头格式



源端口和目的端口字段：各占 2 字节（16位）。端口是运输层与应用层的服务接口。运输层的复用和分用功能都要通过端口才能实现。

序列号：在建立连接时由计算机生成的随机数作为其初始值，通过 SYN 包传给接收端主机，每发送一次数据，就「累加」一次该「数据字节数」的大小。用来解决网络包乱序问题。

确认应答号：指下一次「期望」收到的数据的序列号，发送端收到这个确认应答以后可以认为在这个序号以前的数据都已经被正常接收。用来解决丢包的问题。

控制位：

- ACK：该位为 1 时，「确认应答」的字段变为有效，TCP 规定除了最初建立连接时的 SYN 包之外该位必须设置为 1 。
- RST：该位为 1 时，表示 TCP 连接中出现异常必须强制断开连接。
- SYN：该位为 1 时，表示希望建立连接，并在其「序列号」的字段进行序列号初始值的设定。

- FIN：该位为 1 时，表示今后不会再有数据发送，希望断开连接。当通信结束希望断开连接时，通信双方的主机之间就可以相互交换 FIN 位为 1 的 TCP 段。
- 紧急 URG —— 当 URG = 1 时，表明紧急指针字段有效。它告诉系统此报文段中有紧急数据，应尽快传送(相当于高优先级的数据)。

数据偏移（即首部长度）：占 4 位，它指出 TCP 报文段的数据起始处距离 TCP 报文段的起始处有多远，也就是TCP首部的长度。“数据偏移”的单位是 32 位字（以 4 字节为计算单位），最大1111表示15x4=60个字节，即表示TCP首部最大长度为60个字节，因此“选项”部分最多40个字节。

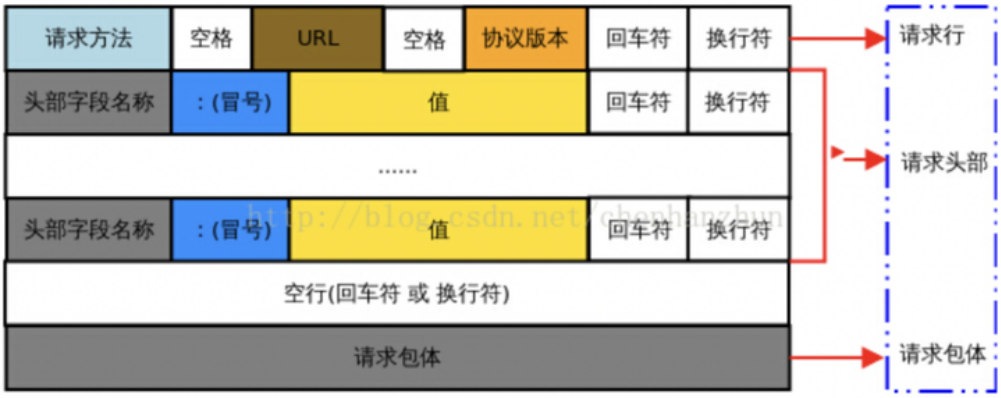
保留字段：占 6 位，保留为今后使用，但目前应置为 0。

窗口字段：占 2 字节，用来让对方设置发送窗口的依据，单位为字节。

检验和：占 2 字节。检验和字段检验的范围包括首部和数据这两部分。在计算检验和时，要在TCP 报文段的前面加上 12 字节的伪首部。

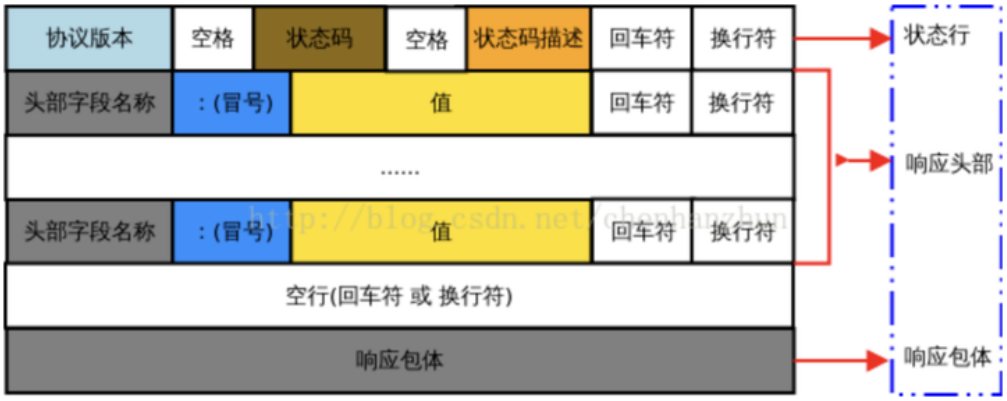
紧急指针字段： 占 16 位，指出在本报文段中紧急数据共有多少个字节。

对于HTTP请求报文，由**请求行**、**请求头**、**空行**和**请求包体**组成



- **请求行**：请求行是请求报文的第一行，它包括：
  - 请求方法：如 `GET`、`POST`、`PUT`、`DELETE` 等。
  - 请求URI：请求的资源的标识符，如 `/index.html`。
  - HTTP版本：例如 `HTTP/1.1` 或 `HTTP/2`。
- **请求头**：这部分是一系列的键值对，提供有关请求的附加信息。常见的请求头有：
  - `Host`：指定请求资源的主机和端口号。
  - `User-Agent`：描述发出请求的用户代理（通常是浏览器）的信息。
  - `Accept`：告诉服务器客户端能够处理的媒体类型。
  - `Content-Type`：在请求中有数据发送时（如POST请求），表示数据的媒体类型。
  - `Content-Length`：表示请求正文的长度（以字节为单位）。
  - `Cookie`：发送之前存储在客户端的cookie。
- **空行**：请求头和请求正文之间的空行，表示头部结束。
- **请求正文**：对于某些请求方法（如 `POST` 或 `PUT`），请求正文包含要发送给服务器的数据。例如，当您在网页上填写表单并提交时，表单数据将作为请求正文发送

对于HTTP响应报文，由**请求行**、**请求头**、**空行**和**请求包体**组成

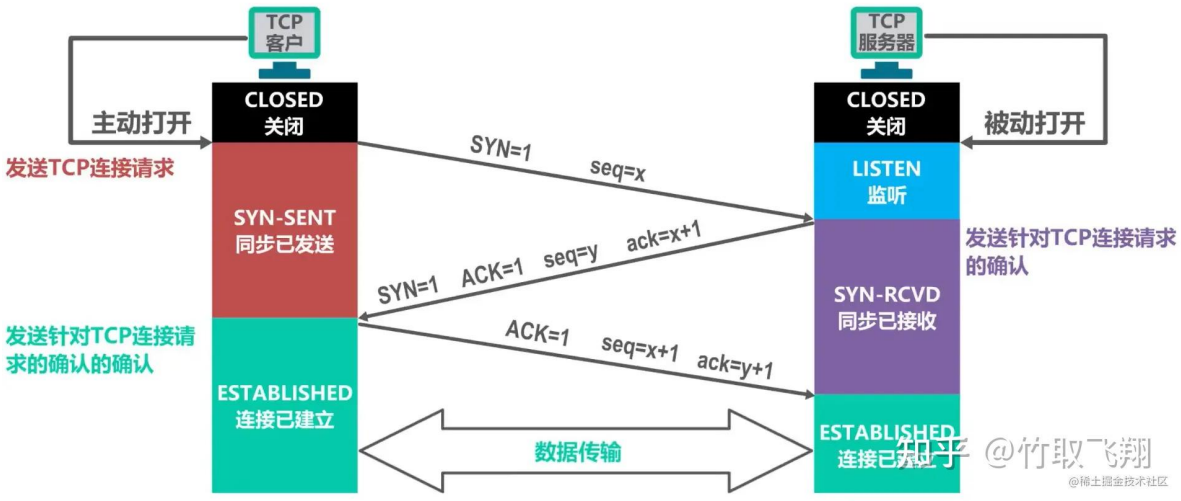


- **状态行**：这是响应报文的第一行，包括以下信息：
  - HTTP版本：例如`HTTP/1.1`或`HTTP/2`
  - 状态码：一个三位数字，表示请求的结果。例如`200`表示“OK”，`404`表示“未找到”。
  - 状态文本：对状态码的简短描述，例如“OK”或“Not Found”。
- **响应头**：这部分包含了一系列的键值对，描述了响应的各种属性和设置。常见的响应头包括：
  - `Content-Type`：表示响应正文的媒体类型，例如`text/html`或`application/json`。
  - `Content-Length`：表示响应正文的长度（以字节为单位）。
  - `Set-Cookie`：用于设置浏览器中的cookie。
  - `Cache-Control`：提供缓存指令，告诉浏览器如何缓存响应内容。
  - `Location`：在某些类型的响应（如302重定向）中使用，指示浏览器导航到的新URL。
- **空行**：响应头和响应正文之间的一个空行，表示头部信息的结束。
- **响应正文**：这部分包含了实际的响应内容，例如HTML页面、图像、CSS或JavaScript文件等。不是所有的HTTP响应都有响应正文，例如一些重定向或无内容的响应。

2、三次握手

（1）三次握手介绍

三次握手是为了建立TCP可信连接，主要目的是为了防止已失效的连接请求突然传到服务器，从而产生错误。通过三次握手，双方都能确认自己和对方的序列号是正确的，确保连接的可靠性。其实意图如下图。（后文会详细验证三次握手中序列号和验证号的关系）



（2）分析三次握手数据包

对应的数据包为前三个数据包

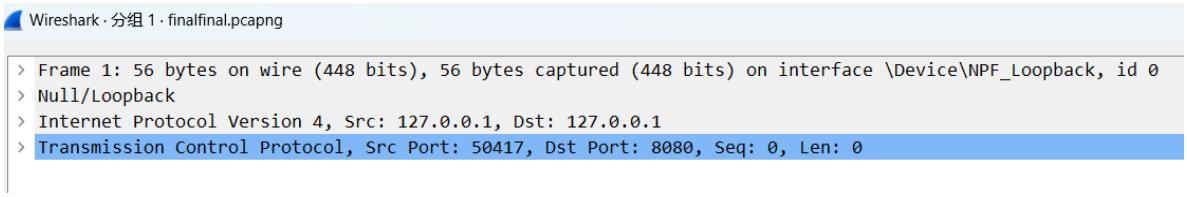
```
1 0.000000    127.0.0.1    127.0.0.1    TCP    56 50417 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
2 0.000058    127.0.0.1    127.0.0.1    TCP    56 8080 → 50417 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
3 0.000104    127.0.0.1    127.0.0.1    TCP    44 50417 → 8080 [ACK] Seq=1 Ack=1 Win=327424 Len=0
```

1. **SYN（同步序列编号）：**客户端发送一个SYN （同步序列号）到服务器端，询问是否可以打开一个连接。
2. **SYN + ACK（同步确认）：**服务器端响应一个 SYN 包来确认客户端的 SYN。同时，服务器也发送一个 ACK 包确认客户端的 SYN 已经被接收。
3. **ACK（确认）：**客户端再次发送一个 ACK 包给服务器，确认服务器的 SYN 也已被接收。

完成这三步后，客户端和服务器端的连接被成功建立，之后就可以开始传输数据了。

（3）逐个分析数据包

打开wireshark的数据包详情，图示如下



可以发现数据包大致分为四层，这四层模型基于TCP/IP模型，可以看到Frame (数据链路层/物理层的表示)、Null/Loopback（本地环境，相当于链路层的一种特例）、网络层 (例如 IPv4)、传输层 (例如 TCP)。还有一层为应用层（将在后续数据包中出现，如HTTP)

第一次握手——客户端发送一个 SYN 包到服务器端

```
Transmission Control Protocol, Src Port: 50417, Dst Port: 8080, Seq: 0, Len: 0
    Source Port: 50417
    Destination Port: 8080
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 3862177286
    [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: 0x862d [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]
```

Source Port: `50417`：这是数据包的发送方的端口号。通常，源端口是随机选择的，而目标端口则对应于特定的服务或应用程序。

Destination Port: `8080`：这是数据包的接收方的端口号。端口`8080`通常与Web服务（例如某些HTTP服务器）相关。

[Stream index: 0]：流索引，在Wireshark中，流索引是为TCP连接分配的唯一标识符。跟踪同一个TCP会话中的数据包。

[TCP Segment Len: 0]：这是TCP数据包中数据部分的长度。这里长度为0，说明这个数据包没有携带任何应用数据，一般代表这个数据包是一个连接控制包（如SYN, ACK等）。

Sequence Number: 0 (relative sequence number)：序列号用于对TCP数据包进行排序，以便在接收端能正确地重组数据流。这里的相对序列号是Wireshark提供的，使分析变得更加简单。

Sequence Number (raw): 3862177286：这是原始的序列号，没有进行任何转换。

Next Sequence Number: 1 (relative sequence number)：这是下一个预期的TCP序列号。由于这是一个SYN包并且没有数据，所以下一个序列号仅增加1。

Acknowledgment Number: 0：确认号表示发送方最后成功接收的数据字节的下一个序列号。因为这是一个SYN包（连接的开始），所以确认号为0。

Header Length: 32 bytes (8)：这表示TCP头部的长度为32字节。它是基于32位字长计算的，所以是8个32位字。

Flags: 0x002 (SYN)：TCP头中的标志字段，用于指示数据包的特定特性或目的。这里SYN标志为1，表示这是一个连接请求。

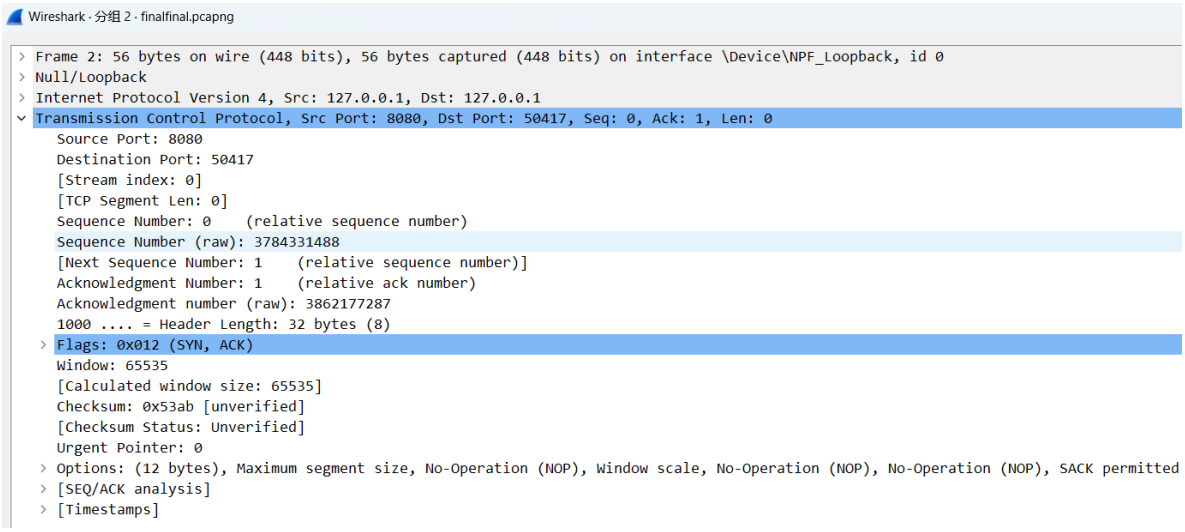
Window: 65535：这是TCP窗口大小，表示接收方在需要下一个ACK之前可以接收的字节数。

Checksum: 0x862d [unverified]：校验和用于检查数据包是否在传输过程中被损坏。"unverified"表示Wireshark没有验证这个校验和。

Urgent Pointer: 0：紧急指针用于指示TCP数据中的紧急数据。但在这个包中，它没有被使用。

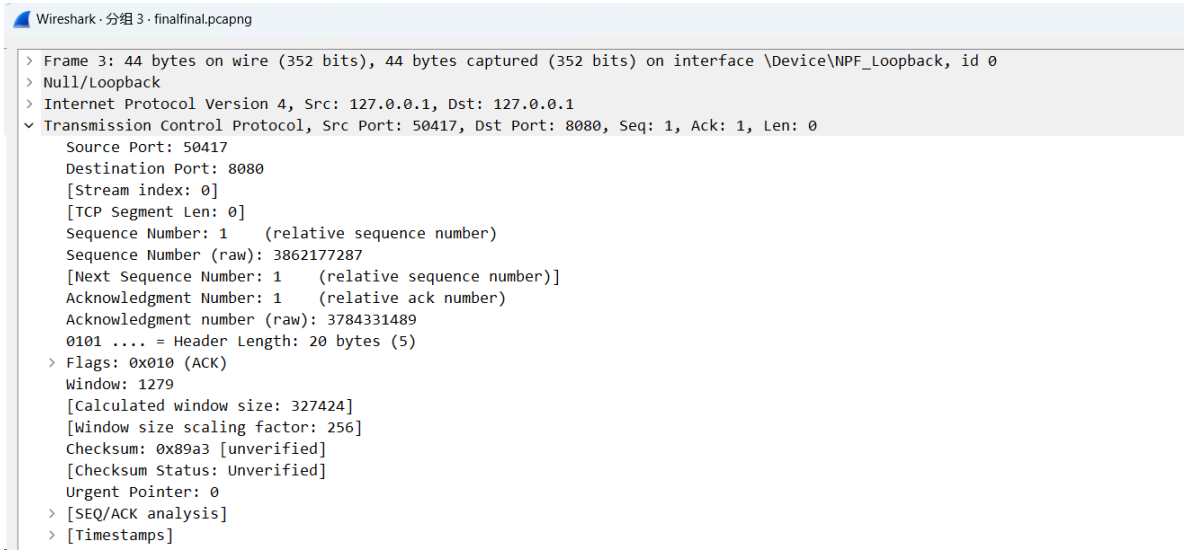
Options：这部分包含了TCP头的可选字段。在您的数据包中，有关于最大段大小、窗口缩放以及其他一些选项

第二次握手——SYN + ACK（同步确认）



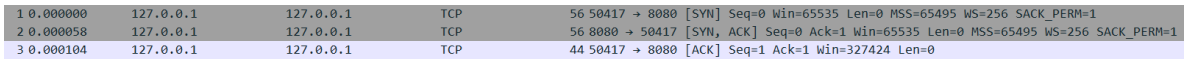
端口 `8080` 向端口 `50417` 发送数据包，`Flags` 位为 0x012 (SYN, ACK)，SYN和ACK标志为1，表示这是一个对SYN请求的响应。

第三次握手——ACK（确认）



端口 `50417` 向端口 `8080` 发送数据包，`Flags` 为0x010 (ACK)，ACK标志为1，表示这是一个纯粹的确认包，确认服务器的 SYN 也已被接收。

（4）验证序列号和确认号关系

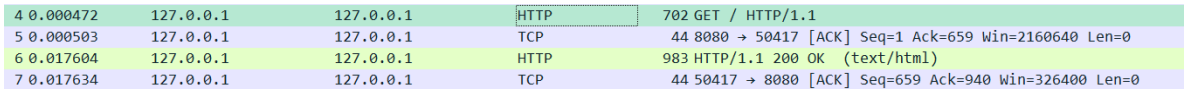


根据本部分对于三次握手的序列号和和确认号的关系，我对我捕获的数据包进行验证

在TCP三次握手中，按理说每个数据包的序列号会增加1（如果不携带数据）。也就是说，当一个方向的通信中发送了一个SYN或ACK标志的数据包，接收方会发送一个确认号，这个确认号是原始序列号加1。

在我捕获的数据包中，第一个数据包的序列号是0，第二个数据包的序列号是1（并且确认号是1），第三个数据包的序列号也是1（并且确认号是2）。这完美地符合TCP三次握手的序列号和确认号的预期关系。

3、对网页的GET请求与回复



- 数据包4是一个HTTP GET请求，内容类型是text/html，请求的事主网页的html，请求路径为 /，也就是首页。
- 数据包5是对之前的HTTP GET请求（数据包 4）的确认
- 数据包6是一个HTTP响应，状态码是200 OK，内容类型是text/html。这意味着请求成功，服务器返回了HTML内容
- 数据包7是对之前的HTTP GET请求（数据包 6）的确认

（1）分析数据包4——HTTP GET请求，请求页面本身

- **GET / HTTP/1.1**: 这是一个HTTP GET请求，请求根目录。使用的HTTP版本是1.1。
- **Host**: 请求的主机地址是127.0.0.1，端口号是8080。
- **Connection: keep-alive**: 这表示客户端希望保持TCP连接，不希望服务器在响应后立即关闭连接。
- **sec-ch-ua** 和 **sec-ch-ua-mobile**: 这些是新的HTTP头，用于帮助网站识别浏览器和其特性。
- **User-Agent**: 提供了关于发送请求的客户端（通常是浏览器）的详细信息。在这里，它表示使用的是基于Windows 10 x64的Chrome浏览器，版本号为118.0.0.0，并使用了AppleWebKit渲染引擎。
- **Accept**: 列出了客户端愿意接受的媒体类型。
- **Sec-Fetch-Site, Sec-Fetch-Mode, Sec-Fetch-User, Sec-Fetch-Dest**: 这些是Fetch metadata请求头，提供了关于请求的上下文信息。
- **Accept-Encoding**: 表示客户端支持的编码方式，如gzip、deflate等。
- **Accept-Language**: 指示客户端的首选语言，这里是中文。

（2）分析数据包5——ACK确认

类似于三次握手中的第三次握手，Flags为0x010，表示确认HTTP GET请求已收到

（3）分析数据包6——HTTP响应，状态码200

```
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
    accept-ranges: bytes\r\n
    cache-control: no-cache, no-store, must-revalidate\r\n
    last-modified: Tue, 24 Oct 2023 07:14:03 GMT\r\n
    etag: W/"1688849860926271-598-2023-10-24T07:14:03.707Z"\r\n
  > content-length: 598\r\n
    content-type: text/html; charset=UTF-8\r\n
    Date: Tue, 24 Oct 2023 14:46:40 GMT\r\n
    Connection: keep-alive\r\n
    Keep-Alive: timeout=5\r\n
    \r\n
    [HTTP response 1/2]
    [Time since request: 0.017132000 seconds]
    [Request in frame: 4]
    [Next request in frame: 8]
    [Next response in frame: 63]
    [Request URI: http://127.0.0.1:8080/icon.jpg]
    File Data: 598 bytes
Line-based text data: text/html (23 lines)
    <!DOCTYPE html>\r\n
    <html lang="en">\r\n
    \r\n
    <head>\r\n
        <meta charset="UTF-8">\r\n
```

HTTP状态表示信息：`HTTP/1.1 200 OK \r \n`,使用了HTTP/1.1协议,返回状态码为200 OK，表示请求成功。

HTTP响应头信息为：

- `accept-ranges`: bytes — 支持部分内容请求。
- `cache-control`: no-cache, no-store, must-revalidate — 指示缓存不能存储响应内容，每次都需要重新验证。
- `last-modified`: Tue, 24 Oct 2023 07:14:03 GMT — 响应的资源上次修改的日期和时间。
- `etag`: W/"1688849860962271-598-2023-10-24T07:14:03.707Z" — 资源的实体标签，用于验证资源的版本。
- `content-length`: 598 — 响应体的长度，以字节为单位。
- `content-type`: text/html; charset=UTF-8 — 响应的内容类型是HTML，并使用UTF-8字符集。
- `Date`: Tue, 24 Oct 2023 14:46:40 GMT — 响应生成的日期和时间。
- `Connection`: keep-alive — 连接将保持活跃状态。
- `Keep-Alive`: timeout=5 — 连接保持活跃的最大时长为5秒

其他信息包含：

- 这是一个HTTP响应序列中的第1个响应。
- 从发出请求到收到此响应的时间为0.017132000秒。
- 相关的HTTP请求在第4帧中。
- 下一个HTTP请求在第8帧中。
- 下一个HTTP响应在第63帧中。
- 请求的URI是：(http://127.0.0.1:8080/icon.jpg)
- 整个文件大小为598byte

响应主体包含：个人网站的html文档，共23lines

4、对logo图片（图片资源）的GET请求与回复

8	0.028542	127.0.0.1	127.0.0.1	HTTP	625 GET /icon.jpg HTTP/1.1
9	0.028589	127.0.0.1	127.0.0.1	TCP	44 8080 → 50417 [ACK] Seq=940 Ack=1240 Win=2159872 Len=0
10	0.034219	127.0.0.1	127.0.0.1	TCP	65539 8080 → 50417 [ACK] Seq=940 Ack=1240 Win=2159872 Len=65495 [TCP segment of a reassembled PDU]
11	0.034240	127.0.0.1	127.0.0.1	TCP	418 8080 → 50417 [PSH, ACK] Seq=66435 Ack=1240 Win=2159872 Len=374 [TCP segment of a reassembled PDU]
12	0.034301	127.0.0.1	127.0.0.1	TCP	44 50417 → 8080 [ACK] Seq=1240 Ack=66809 Win=327424 Len=0
13	0.034561	127.0.0.1	127.0.0.1	TCP	56 50418 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
14	0.034614	127.0.0.1	127.0.0.1	TCP	56 8080 → 50418 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
15	0.034651	127.0.0.1	127.0.0.1	TCP	44 50418 → 8080 [ACK] Seq=1 Ack=1 Win=327424 Len=0
16	0.034706	127.0.0.1	127.0.0.1	TCP	65539 8080 → 50417 [ACK] Seq=66809 Ack=1240 Win=2159872 Len=65495 [TCP segment of a reassembled PDU]
17	0.034715	127.0.0.1	127.0.0.1	TCP	85 8080 → 50417 [PSH, ACK] Seq=132304 Ack=1240 Win=2159872 Len=41 [TCP segment of a reassembled PDU]
18	0.034773	127.0.0.1	127.0.0.1	TCP	44 50417 → 8080 [ACK] Seq=1240 Ack=132345 Win=327424 Len=0
19	0.035354	127.0.0.1	127.0.0.1	HTTP	584 GET /intro.mp3 HTTP/1.1

与网页的传输过程类似，但是图片和音频这种较大资源的传输需要分段传输。概览分析如下：

- 数据包8和9分别是客户端发送给服务器端的HTTP GET请求和相应的ACK标识传递
- 数据包10到11是TCP包，它含有`[TCP segment of a reassembled PDU]`，这意味着这个数据包是一个更大数据传输中的一部分，它最终会被重组
- 数据包12为客户端向服务器发送的ACK，表示我确实收到了一部分图片的数据包
- 数据包13、数据包14和数据包15一个新的TCP三次握手过程，这个通常是因为浏览器有时会为同一个域名创建多个并行的TCP连接，以便同时下载多个资源（例如，图片、CSS、JS文件等）。这可以减少等待时间，由于在数据包19时开始传了音频数据，此时图片资源还未传输完毕，所以需要新建一个TCP链接传递音频资源。
- 同时我们发现TCP数据包的`SACK_PERM`标识为1,说明该TCP连接支持`SACK`，并请求对端也支持`SACK`
    - `SACK`允许接收方指定已接收的非连续的数据段。这是一种TCP的扩展机制，用于改进在存在丢包的情况下的性能。

- 在多个TCP连接、网络不稳定等情况下使用`SACK`有助于提高在出现丢包的情况下的数据传输效率。

最后在数据包63的部位发现客户端返回的HTTP响应，状态码为200，类型为JFIF image表名logo图片资源请求全部完成，如下图所示。

60	0.041155	127.0.0.1	127.0.0.1	TCP	65539	8080 → 50418 [ACK] Seq=262533 Ack=541 Win=2160640 Len=65495 [TCP segment of a reassembled PDU]
61	0.041169	127.0.0.1	127.0.0.1	TCP	85	8080 → 50418 [PSH, ACK] Seq=328028 Ack=541 Win=2160640 Len=41 [TCP segment of a reassembled PDU]
62	0.041205	127.0.0.1	127.0.0.1	TCP	44	50418 → 8080 [ACK] Seq=541 Ack=328069 Win=327424 Len=0
63	0.041677	127.0.0.1	127.0.0.1	HTTP	45671	HTTP/1.1 200 OK (JPEG JFIF image)

(2) 验证序列号和确认号的关系

8	0.028542	127.0.0.1	127.0.0.1	HTTP	625	GET /icon.jpg HTTP/1.1
9	0.028589	127.0.0.1	127.0.0.1	TCP	44	8080 → 50417 [ACK] Seq=940 Ack=1240 Win=2159872 Len=0
10	0.034219	127.0.0.1	127.0.0.1	TCP	65539	8080 → 50417 [ACK] Seq=940 Ack=1240 Win=2159872 Len=65495 [TCP segment of a reassembled PDU]
11	0.034240	127.0.0.1	127.0.0.1	TCP	418	8080 → 50417 [PSH, ACK] Seq=66435 Ack=1240 Win=2159872 Len=374 [TCP segment of a reassembled PDU]
12	0.034301	127.0.0.1	127.0.0.1	TCP	44	50417 → 8080 [ACK] Seq=1240 Ack=66809 Win=327424 Len=0
13	0.034561	127.0.0.1	127.0.0.1	TCP	56	50418 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
14	0.034614	127.0.0.1	127.0.0.1	TCP	56	8080 → 50418 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
15	0.034651	127.0.0.1	127.0.0.1	TCP	44	50418 → 8080 [ACK] Seq=1 Ack=1 Win=327424 Len=0
16	0.034706	127.0.0.1	127.0.0.1	TCP	65539	8080 → 50417 [ACK] Seq=66809 Ack=1240 Win=2159872 Len=65495 [TCP segment of a reassembled PDU]
17	0.034715	127.0.0.1	127.0.0.1	TCP	85	8080 → 50417 [PSH, ACK] Seq=132304 Ack=1240 Win=2159872 Len=41 [TCP segment of a reassembled PDU]
18	0.034773	127.0.0.1	127.0.0.1	TCP	44	50417 → 8080 [ACK] Seq=1240 Ack=132345 Win=327424 Len=0
19	0.035354	127.0.0.1	127.0.0.1	HTTP	584	GET /intro.mp3 HTTP/1.1

1. **数据包8 (HTTP GET 请求)**:
   - 类型：HTTP GET 请求
   - 序列号：940
   - 确认号：1240
   - 数据长度：625字节

   解析: 数据包8是一个HTTP GET请求。它从序列号940开始，并发送了625字节的数据。因此，下一个要发送的字节的序列号将是1565 (940 + 625)。同时，它的确认号为1240，表示它已经成功接收到对端发送的序列号为1239的数据，并期望接收序列号为1240的数据。

2. **数据包9 (TCP ACK)**:
   - 类型：TCP [ACK]
   - 序列号：940
   - 确认号：1240
   - 数据长度：0字节

   解析: 数据包9是一个纯ACK数据包，没有数据传输。它的序列号仍然是940，表示尚未发送更多数据。确认号仍然是1240，表示它仍然期望从对端接收序列号为1240的数据。

3. **数据包10 (TCP ACK，重组的PDU段)**:
   - 类型：TCP [ACK]，并且是一个重组的PDU段
   - 序列号：940
   - 确认号：1240
   - 数据长度：65495字节

   解析: 数据包10开始于序列号940，并发送了65495字节的数据。这意味着这个数据包结束于序列号66434。确认号仍然是1240，表示从对端期望接收的下一个字节的序列号仍然是1240。

4. **数据包11 (TCP PSH, ACK，重组的PDU段)**:
   - 类型：TCP [PSH, ACK]，并且是一个重组的PDU段
   - 序列号：66435
   - 确认号：1240
   - 数据长度：374字节

   解析: 数据包11紧接着数据包10的数据，开始于序列号66435，并发送了374字节的数据。这意味着这个数据包结束于序列号66808。确认号仍然是1240，表示从对端期望接收的下一个字节的序列号仍然是1240。

5. **数据包12 (TCP ACK)**:
   - 类型：TCP [ACK]
   - 序列号：1240

- 确认号：66809
- 数据长度：0字节

解析: 数据包12是对之前收到的数据包的确认。其序列号为1240，表示它准备发送从序列号1240开始的数据。它的确认号为66809，表示它已经接收到对端发送的序列号为66808的数据，并期望接收序列号为66809的数据。

(3) 分析图片类型的响应数据包

```
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
  accept-ranges: bytes\r\n
  cache-control: no-cache, no-store, must-revalidate\r\n
  last-modified: Sat, 30 Sep 2023 07:13:28 GMT\r\n
  etag: W/"1407374884215619-766523-2023-09-30T07:13:28.659Z"\r\n
  content-length: 766523\r\n
  content-type: image/jpeg\r\n
  Date: Tue, 24 Oct 2023 14:46:40 GMT\r\n
  Connection: keep-alive\r\n
  Keep-Alive: timeout=5\r\n
  \r\n
  [HTTP response 2/2]
  [Time since request: 0.013135000 seconds]
  [Prev request in frame: 4]
  [Prev response in frame: 6]
  [Request in frame: 8]
  [Request URI: http://127.0.0.1:8080/icon.jpg]
  File Data: 766523 bytes
JPEG File Interchange Format
  Marker: Start of Image (0xffd8)
  Marker segment: Reserved for application segments - 0 (0xFFE0)
  Marker segment: Reserved for application segments - 1 (0xFFE1)
  Marker segment: Reserved for application segments - 13 (0xFFED)
```

和前面对网页的响应回复的格式一样，数据包的HTTP部分分为状态表示部分、相应头部分、其他附加信息部分和主体部分，有较大不同的在**JPEG File Interchange Format**部分，是JPEG文件的元数据和一些关于图片格式的标记段信息

此外，我们还发现`arrival time`和`last modified time`是不一样的（如下图）

```
Frame 63: 45671 bytes on wire (365368 bits), 45671 bytes captured (365368 bits) on interface \Device\NPF_Loopback, id 0
  Interface id: 0 (\Device\NPF_Loopback)
  Encapsulation type: NULL/Loopback (15)
  Arrival Time: Oct 24, 2023 22:46:40.745460000 马来西亚半岛标准时间
  [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: 1698158800.745460000 seconds
  [Time delta from previous captured frame: 0.000472000 seconds]
  [Time delta from previous displayed frame: 0.000472000 seconds]
  [Time since reference or first frame: 0.041677000 seconds]
  Frame Number: 63
  Frame Length: 45671 bytes (365368 bits)
  Capture Length: 45671 bytes (365368 bits)
  [Frame is marked: False]
  [Frame is ignored: False]
  [Protocols in frame: null:ip:tcp:http:image-jfif]
  [Coloring Rule Name: HTTP]
  [Coloring Rule String: http || tcp.port == 80 || http2]
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 8080, Dst Port: 50417, Seq: 722169, Ack: 1240, Len: 45627
[23 Reassembled TCP Segments (766856 bytes): #10(65495), #11(374), #16(65495), #17(41), #21(65495), #22(41), #24(65495), #25(41), #27(65495), #28(
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
  accept-ranges: bytes\r\n
  cache-control: no-cache, no-store, must-revalidate\r\n
  last-modified: Sat, 30 Sep 2023 07:13:28 GMT\r\n
  etag: W/"1407374884215619-766523-2023-09-30T07:13:28.659Z"\r\n
```

- **Arrival Time**: 数据包的到达时间标记为 "Arrival Time: 2023-09-30 07:13:28.000472000"，表示该数据包在这个时间到达捕获设备。
- **HTTP Header 的 Last-Modified**: HTTP 头部有一个 "Last-Modified" 字段，显示的时间是 "Sat, 30 Sep 2023 07:13:28 GMT"，这表示资源在服务器上最后的修改时间。

5、对音频（音频资源）的GET请求与回复

19	0.035354	127.0.0.1	127.0.0.1	HTTP	584 GET /intro.mp3 HTTP/1.1
20	0.035385	127.0.0.1	127.0.0.1	TCP	44 8080 → 50418 [ACK] Seq=1 Ack=541 Win=2160640 Len=0
21	0.035534	127.0.0.1	127.0.0.1	TCP	65539 8080 → 50417 [ACK] Seq=132345 Ack=1240 Win=2159872 Len=65495 [TCP segment of a reassembled PDU]
22	0.035543	127.0.0.1	127.0.0.1	TCP	85 8080 → 50417 [PSH, ACK] Seq=197840 Ack=1240 Win=2159872 Len=41 [TCP segment of a reassembled PDU]
23	0.035592	127.0.0.1	127.0.0.1	TCP	44 50417 → 8080 [ACK] Seq=1240 Ack=197881 Win=327424 Len=0
24	0.037557	127.0.0.1	127.0.0.1	TCP	65539 8080 → 50417 [ACK] Seq=197881 Ack=1240 Win=2159872 Len=65495 [TCP segment of a reassembled PDU]
25	0.037571	127.0.0.1	127.0.0.1	TCP	85 8080 → 50417 [PSH, ACK] Seq=263376 Ack=1240 Win=2159872 Len=41 [TCP segment of a reassembled PDU]
26	0.037619	127.0.0.1	127.0.0.1	TCP	44 50417 → 8080 [ACK] Seq=1240 Ack=263417 Win=327424 Len=0
27	0.038399	127.0.0.1	127.0.0.1	TCP	65539 8080 → 50417 [ACK] Seq=263417 Ack=1240 Win=2159872 Len=65495 [TCP segment of a reassembled PDU]
28	0.038409	127.0.0.1	127.0.0.1	TCP	85 8080 → 50417 [PSH, ACK] Seq=328912 Ack=1240 Win=2159872 Len=41 [TCP segment of a reassembled PDU]
29	0.038440	127.0.0.1	127.0.0.1	TCP	44 50417 → 8080 [ACK] Seq=1240 Ack=328953 Win=327424 Len=0
30	0.038667	127.0.0.1	127.0.0.1	TCP	65539 8080 → 50417 [ACK] Seq=328953 Ack=1240 Win=2159872 Len=65495 [TCP segment of a reassembled PDU]
31	0.038678	127.0.0.1	127.0.0.1	TCP	85 8080 → 50417 [PSH, ACK] Seq=394448 Ack=1240 Win=2159872 Len=41 [TCP segment of a reassembled PDU]
32	0.038722	127.0.0.1	127.0.0.1	TCP	44 50417 → 8080 [ACK] Seq=1240 Ack=394489 Win=327424 Len=0

具体的情况和传输图片资源相同，可以发现客户端每收到两个含有PDU标识的TCP数据包之后就发送一个ACK给服务器，这可能是依据TCP接收缓冲区大小确定的，当数据包到达并放入接收缓冲区后，系统可能会决定立即发送ACK以告诉发送方它已成功接收了这些数据包，客户端可能选择在接收到一定数量的数据包后发送ACK。

最后在167号数据包的位置捕获到了对音频资源的响应回复，如下图

163	0.050055	127.0.0.1	127.0.0.1	TCP	44 50418 → 8080 [ACK] Seq=541 Ack=2490757 Win=65280 Len=0
164	0.059948	127.0.0.1	127.0.0.1	TCP	44 [TCP Window Update] 50418 → 8080 [ACK] Seq=541 Ack=2490757 Win=327424 Len=0
165	0.060033	127.0.0.1	127.0.0.1	TCP	65539 8080 → 50418 [ACK] Seq=2490757 Ack=541 Win=2160640 Len=65495 [TCP segment of a reassembled PDU]
166	0.060046	127.0.0.1	127.0.0.1	TCP	65539 8080 → 50418 [PSH, ACK] Seq=2556252 Ack=541 Win=2160640 Len=65495 [TCP segment of a reassembled PDU]
167	0.060055	127.0.0.1	127.0.0.1	HTTP	37943 HTTP/1.1 206 Partial Content (audio/mpeg)

`206` 状态码是HTTP协议中的一个响应状态码，称为 `Partial Content`（部分内容）。当客户端发送一个带有 `Range` 头部的请求，要求从服务器获取资源的一部分（而不是整个资源）时，服务器会返回 `206` 状态码。遇到如下几种情况，通常会出现 `206` 状态码：

- **断点续传**：当用户下载大文件时，如果因为某种原因（如网络中断）下载被中断，使用断点续传可以从已下载部分继续下载，而不是重新开始。在这种情况下，客户端将请求从上次中断的位置开始的文件部分，服务器会响应 `206` 状态码，并仅提供请求的部分内容
- **多媒体流**：当播放视频或音频流时，客户端可能只请求从某个时间点开始的媒体内容，例如用户跳过视频的前几分钟。这允许快速地定位到媒体流的特定部分，而无需下载整个资源。
- **并行下载**：某些下载工具可以分割文件并发起多个并行请求，每个请求下载文件的一个部分。这样可以加速下载速度。每个这样的请求都可能导致一个 `206` 响应。

6、对icon图片（图片资源、网页小图标）的GET请求与回复

icon图片资源是浏览器中网页标签左侧的小icon图片。

一般本地web服务器会自动检索文件夹中名为favicon.ico的图片文件，并使用GET方法获取图片资源放到网页中，如果没有设置则会捕获到HTTP/1.1 404 Not Found的响应数据包，同时我们也可以使用html中的linked标签引用新的图标资源

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>个人介绍</title>
    <link rel="icon" href="icon2.jpg" type="image/jpeg">
</head>
```

捕获的请求和响应如下，和logo图片资源的请求响应过程一样

170	0.114722	127.0.0.1	127.0.0.1	HTTP	626 GET /icon2.jpg HTTP/1.1
171	0.114787	127.0.0.1	127.0.0.1	TCP	44 8080 → 50418 [ACK] Seq=2659646 Ack=1123 Win=2160128 Len=0
172	0.121321	127.0.0.1	127.0.0.1	TCP	65539 8080 → 50418 [ACK] Seq=2659646 Ack=1123 Win=2160128 Len=65495 [TCP segment of a reassembled PDU]
173	0.121340	127.0.0.1	127.0.0.1	TCP	418 8080 → 50418 [PSH, ACK] Seq=2725141 Ack=1123 Win=2160128 Len=374 [TCP segment of a reassembled PDU]
174	0.121402	127.0.0.1	127.0.0.1	TCP	44 50418 → 8080 [ACK] Seq=1123 Ack=2725515 Win=327424 Len=0

206	0.123737	127.0.0.1	127.0.0.1	HTTP	45671 HTTP/1.1 200 OK (JPEG JFIF image)
207	0.123781	127.0.0.1	127.0.0.1	TCP	44 50418 → 8080 [ACK] Seq=1123 Ack=3426502 Win=281600 Len=0

7、四次挥手关闭连接

（1）四次挥手介绍

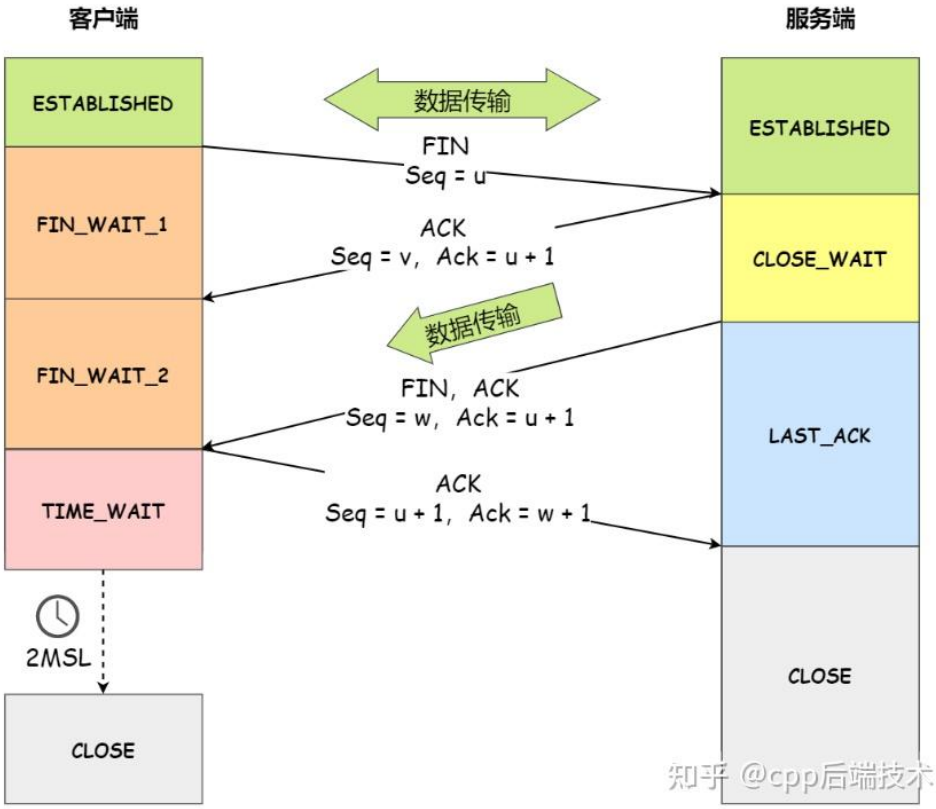
TCP四次挥手过程，用于终止TCP连接，目的是确保双方都能正常地关闭连接，以及确保双方都已发送和接收所有的数据。

TCP四次挥手的详细步骤如下：

1. **FIN**: 客户端（或一方）决定关闭连接，并发送一个FIN标志的数据包给服务器。此时，客户端进入`FIN_WAIT_1`状态。这表明客户端已经完成了它的数据发送任务，但仍可以接收数据。
2. **ACK**: 服务器收到客户端的FIN数据包后，它会发送一个ACK标志的数据包作为对FIN数据包的确认。此时，客户端进入`FIN_WAIT_2`状态，而服务器则进入`CLOSE_WAIT`状态。服务器向客户端发送ACK后，还需要等待向客户端发送完剩余的数据。
3. **FIN**: 当服务器发送完所有数据后，它会发送另一个带FIN标志的数据包给客户端，表示服务器也准备好关闭连接了。此时，服务器进入`LAST_ACK`状态。
4. **ACK**: 客户端收到服务器的FIN数据包后，它也会回复一个ACK数据包，并进入`TIME_WAIT`状态。这个状态会持续一段时间（通常是约2倍的最大段生命周期），确保服务器收到了客户端的ACK数据包。一旦这个时间过去，客户端会进入`CLOSED`状态，连接正式关闭。

> 需要注意的是，尽管通常是客户端先发起关闭连接的请求，但这个过程同样可以由服务器端先发

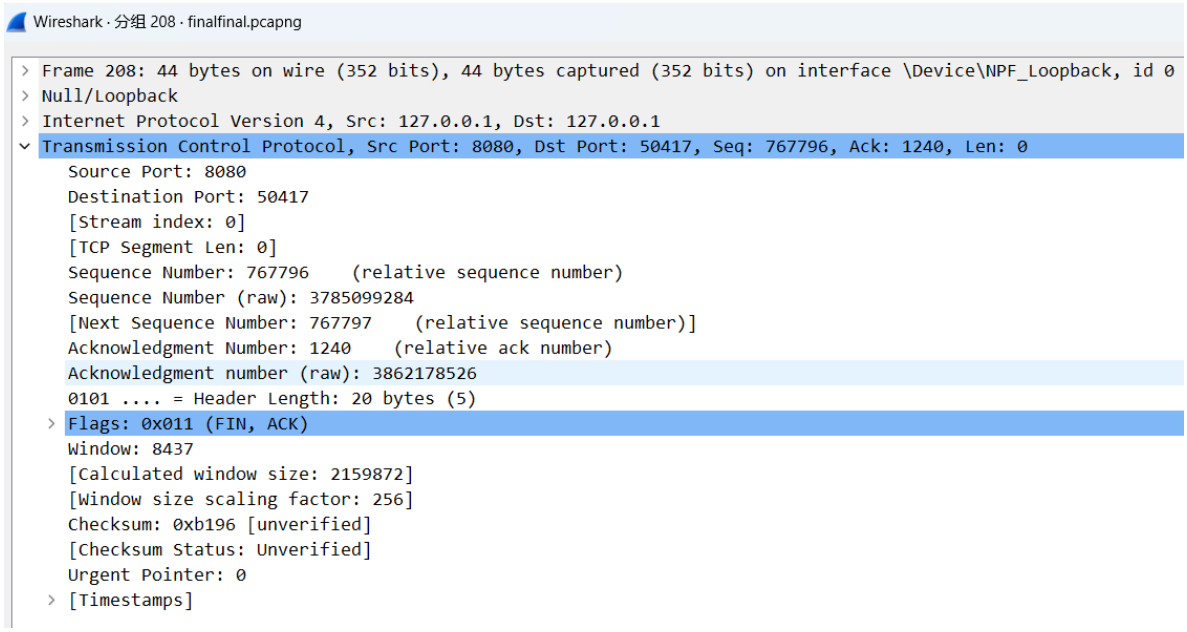
详细图解如下：



（2）数据包解析

207	0.123781	127.0.0.1	127.0.0.1	TCP	44 50418 → 8080 [ACK] Seq=1123 Ack=3426502 Win=281600 Len=0
208	5.056407	127.0.0.1	127.0.0.1	TCP	44 8080 → 50417 [FIN, ACK] Seq=767796 Ack=1240 Win=2159872 Len=0
209	5.056452	127.0.0.1	127.0.0.1	TCP	44 50417 → 8080 [ACK] Seq=1240 Ack=767797 Win=281600 Len=0
210	5.133963	127.0.0.1	127.0.0.1	TCP	44 8080 → 50418 [FIN, ACK] Seq=3426502 Ack=1123 Win=2160128 Len=0
211	5.133986	127.0.0.1	127.0.0.1	TCP	44 50418 → 8080 [ACK] Seq=1123 Ack=3426503 Win=281600 Len=0

第一次挥手



Source Port (源端口): `8080`：这是发送数据包的设备使用的端口。

Destination Port (目的端口): `50417` 这是接收数据包的设备使用的端口。

TCP Segment Len (TCP段长度): `0` 表示这个数据包不包含任何TCP数据负载，只是用于控制。

Sequence Number (序列号): `767796` 这是发送方为这个段指定的序列号。

Next Sequence Number (下一个序列号): `767797` 这是发送方的下一个期望的序列号。

Acknowledgment Number (确认号): `1240` 这是接收方期望的下一个序列号。

Flags (标志位): `0x011 (FIN, ACK)` 这里设置了 `FIN` 和ACK标志。

- `FIN` 表示发送方已经发送完毕，希望关闭连接。
- `ACK` 表示这个段也作为之前接收的数据的确认。

Window (窗口大小): `8437`：这表示接收方现在可以接收的数据的数量。

Calculated window size (计算的窗口大小): `2159872`：这可能是考虑了窗口缩放因子后的窗口大小。

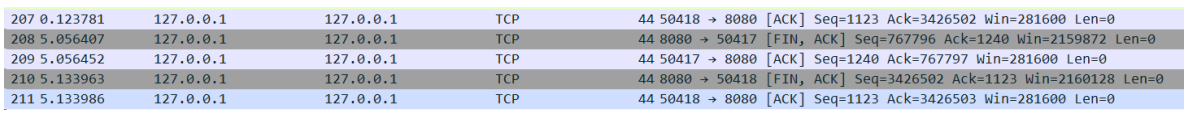
Window size scaling factor (窗口大小缩放因子): `256`：用于调整窗口大小，允许更大的窗口值。

Checksum (校验和): `0xb196`：用于检查数据包在传输过程中是否出现错误。

Checksum Status (校验和状态): `Unverified`：表示该校验和未经验证。

Urgent Pointer: `0`：当URG标志被设置时，这表示紧急数据的结束位置，但在这里它的值为0，所以不包含紧急数

（3）验证序列号和确认号关系



第一次挥手：发送方发出关闭连接的请求，它的序列号是`767796`。这个序列号表示发送方发送数据的顺序，即这是发送方发送的第`767796`个字节。

第二次挥手：接收方回应发送方，告知已经收到关闭请求。它的序列号是`1240`，表示这是接收方发送的第`1240`个字节。它的确认号是`767797`，这表示接收方期望从发送方接收的下一个字节是第`767797`个字节，也就是确认了发送方的序列号`767796` + 1。

第三次挥手：发送方再次发送关闭连接的请求，但这次是向另一个端口`50418`。它的序列号是`3426502`，表示这是发送方发送的第`3426502`个字节。确认号`1123`是发送方期望从接收方接收的下一个字节的序列号。

第四次挥手：接收方最后回应发送方，确认已经收到关闭请求。它的序列号是`1123`，表示这是接收方发送的第`1123`个字节。它的确认号是`3426503`，这表示接收方期望从发送方接收的下一个字节是第`3426503`个字节，也就是确认了发送方的序列号`3426502` + 1。

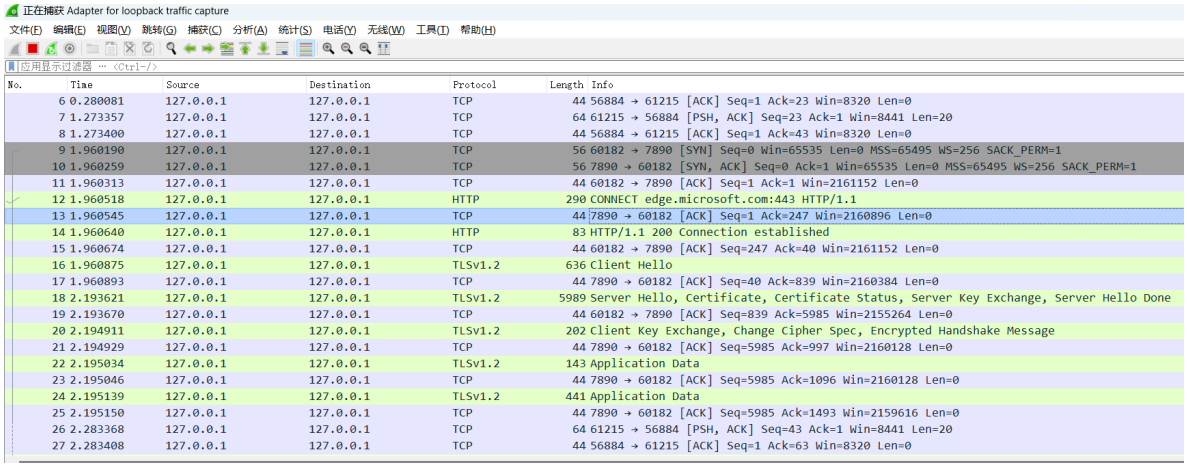
总结：在四次挥手的过程中，确认号总是对方的序列号+1。这种机制确保了双方都正确接收到了对方的数据。当一方发送数据时，它提供一个序列号；当另一方回应时，它的确认号就是这个序列号+1，表示它已经收到了那个序列号的数据，并期望接收下一个序列号的数据。这样，双方都能确保数据的可靠传输，并且都知道对方已经正确地接收到数据。

四、问题与解决

1、抓包分析过程中抓不到http数据包，而是TLSV数据包

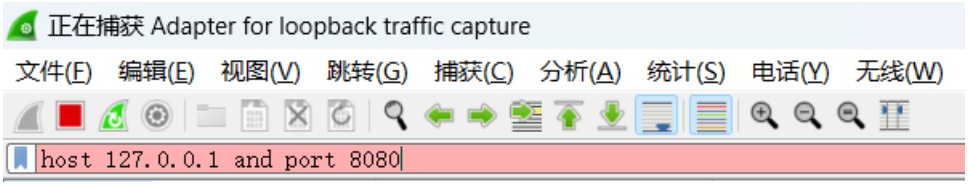
Destination	Protocol	Length	Info
127.0.0.1	HTTP	294	CONNECT clientservices.googleapis.com:443 HTTP/1.1
127.0.0.1	HTTP	83	HTTP/1.1 200 Connection established
127.0.0.1	TLSv1	657	Client Hello
127.0.0.1	HTTP	274	CONNECT accounts.google.com:443 HTTP/1.1
127.0.0.1	HTTP	83	HTTP/1.1 200 Connection established
127.0.0.1	TLSv1	647	Client Hello
127.0.0.1	TLSv1.3	4904	Server Hello, Change Cipher Spec, Application Data
127.0.0.1	TLSv1.3	118	Change Cipher Spec, Application Data
127.0.0.1	TLSv1.3	136	Application Data
127.0.0.1	TLSv1.3	393	Application Data
127.0.0.1	TLSv1.3	4533	Server Hello, Change Cipher Spec, Application Data
127.0.0.1	TLSv1.3	118	Change Cipher Spec, Application Data
127.0.0.1	TLSv1.3	136	Application Data
127.0.0.1	TLSv1.3	1617	Application Data

2、抓包捕获到的数据包内容繁杂，很不纯净，无法分析



No.	Time	Source	Destination	Protocol	Length	Info
6	0.280081	127.0.0.1	127.0.0.1	TCP	44	56884 → 61215 [ACK] Seq=1 Ack=23 Win=8320 Len=0
7	1.273357	127.0.0.1	127.0.0.1	TCP	64	61215 → 56884 [PSH, ACK] Seq=23 Ack=1 Win=8441 Len=20
8	1.273400	127.0.0.1	127.0.0.1	TCP	44	56884 → 61215 [ACK] Seq=1 Ack=43 Win=8320 Len=0
9	1.960190	127.0.0.1	127.0.0.1	TCP	56	60182 → 7890 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
10	1.960259	127.0.0.1	127.0.0.1	TCP	56	7890 → 60182 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
11	1.960313	127.0.0.1	127.0.0.1	TCP	44	60182 → 7890 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
12	1.960518	127.0.0.1	127.0.0.1	HTTP	290	CONNECT edge.microsoft.com:443 HTTP/1.1
13	1.960545	127.0.0.1	127.0.0.1	TCP	44	7890 → 60182 [ACK] Seq=1 Ack=247 Win=2160896 Len=0
14	1.960640	127.0.0.1	127.0.0.1	HTTP	83	HTTP/1.1 200 Connection established
15	1.960674	127.0.0.1	127.0.0.1	TCP	44	60182 → 7890 [ACK] Seq=247 Ack=40 Win=2161152 Len=0
16	1.960875	127.0.0.1	127.0.0.1	TLSv1.2	636	Client Hello
17	1.960893	127.0.0.1	127.0.0.1	TCP	44	7890 → 60182 [ACK] Seq=40 Ack=839 Win=2160384 Len=0
18	2.193621	127.0.0.1	127.0.0.1	TLSv1.2	5989	Server Hello, Certificate, Certificate Status, Server Key Exchange, Server Hello Done
19	2.193670	127.0.0.1	127.0.0.1	TCP	44	60182 → 7890 [ACK] Seq=839 Ack=5985 Win=2155264 Len=0
20	2.194911	127.0.0.1	127.0.0.1	TLSv1.2	202	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
21	2.194929	127.0.0.1	127.0.0.1	TCP	44	7890 → 60182 [ACK] Seq=5985 Ack=997 Win=2160128 Len=0
22	2.195034	127.0.0.1	127.0.0.1	TLSv1.2	143	Application Data
23	2.195046	127.0.0.1	127.0.0.1	TCP	44	7890 → 60182 [ACK] Seq=5985 Ack=1096 Win=2160128 Len=0
24	2.195139	127.0.0.1	127.0.0.1	TLSv1.2	441	Application Data
25	2.195150	127.0.0.1	127.0.0.1	TCP	44	7890 → 60182 [ACK] Seq=5985 Ack=1493 Win=2159616 Len=0
26	2.283368	127.0.0.1	127.0.0.1	TCP	64	61215 → 56884 [PSH, ACK] Seq=43 Ack=1 Win=8441 Len=20
27	2.283408	127.0.0.1	127.0.0.1	TCP	44	56884 → 61215 [ACK] Seq=1 Ack=63 Win=8320 Len=0

可以使用wireshark的捕获过滤器过滤数据包，只捕获在8080端口上的数据包，只使用显示过滤器无法成功，因为显示过滤器不支持`port`语法，会报错



五、一些相关的思考

1、各个版本http协议的异同点

HTTP/1.1：

- 持久连接：默认使用持久连接（Connection: keep-alive），这意味着多个HTTP请求和响应可以在同一个连接中传输，减少了建立和关闭连接的开销。
- 管道机制：允许在一个连接上同时发送多个请求，但响应仍然是按照请求的顺序排队的。
- 添加了PUT、OPTIONS、DELETE等新的方法。
- 引入了Host头，允许在同一个IP地址上托管多个域名。
- 缓存处理得到加强，通过ETag、If-Modified-Since等头信息实现。
- 支持传输编码如chunked transfer。

HTTP/2：

- 使用二进制格式传输数据，而不是文本格式。
- 多路复用：同一个连接上可以并发多个请求和响应。
- 服务器推送：允许服务器在客户端需要之前"推送"资源。
- 首部压缩：使用HPACK算法压缩请求和响应的头部。

HTTP/3：

- 使用QUIC协议，这是一个基于UDP的多路复用和0-RTT连接的协议，解决了TCP的队头阻塞问题。
- 支持更好的并发和流量控制。
- 连接迁移：比如，当用户的网络从WiFi切换到4G时，连接可以不被中断。
- 持续的安全性和性能改进。

2、四次挥手完成后，为什么主动关闭方需要等待2倍的MSL时间

1. **确保最后一个ACK被接收**：
    - 当主动关闭方发送最后一个ACK给被动关闭方时，这个ACK可能会在网络中丢失。
    - 如果被动关闭方没有收到这个ACK，它会认为主动关闭方没有收到其发送的FIN，所以它会重新发送FIN。
    - 为了应对这种情况，主动关闭方需要维持TIME_WAIT状态，这样如果它再次收到被动关闭方的FIN，它可以再次发送ACK。
    - MSL的时间是为了确保这个ACK在网络中有足够的时间被被动关闭方接收。
2. **等待可能延迟的数据段消失**：
    - 在网络中，尤其是在互联网环境中，数据包可能会因为各种原因在路上延迟。有些数据包可能会被延迟，然后在一个很晚的时间点突然到达。
    - 假设我们的TCP连接终止后，立刻启动了一个新的连接，而这个新连接的端口号和IP地址与前一个连接相同。如果网络中还存在前一个连接的数据段，并且这个数据段突然到达，那么它可能会被错误地认为是新连接的数据，从而导致数据的混淆。
    - 为了避免这种情况，MSL确保在这段时间内，网络中的所有属于之前连接的数据段都已经消失。