计算机网络实验二报告

学号：2210737 姓名：阿斯雅

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）、自己的LOGO、自我介绍的音频信息。
>
> （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，使用`Wireshark`过滤器使其只显示HTTP协议。
>
> （3）现场演示。
>
> （4）提交HTML文档、`Wireshark`捕获文件和实验报告，对HTTP交互过程进行详细说明。

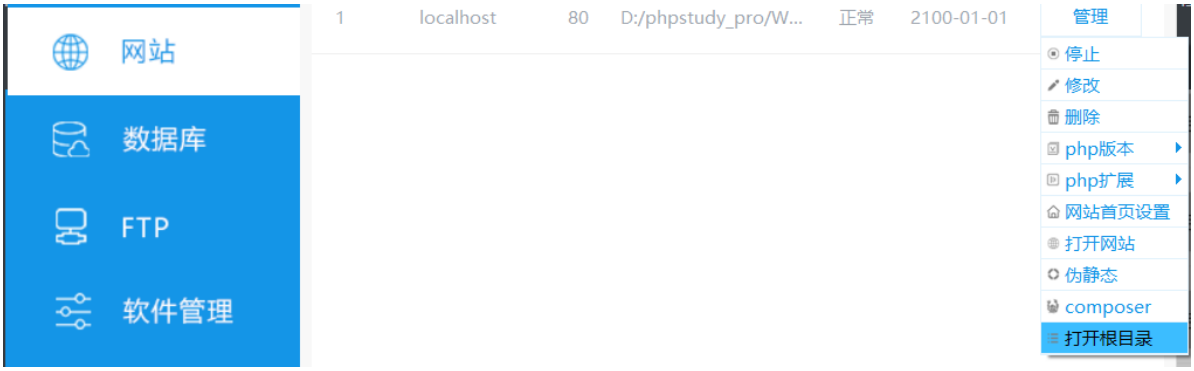
二、实验步骤

2.1、搭建本地web服务器

要想搭建一个本地 Web 服务器，我们需要配置好本地的 `PHP`、`MySQL`、`Apache` 等等一系列的组件。而通过查阅资料后发现 `phpstudy-pro` 工具很好地将这些组件整合在一起，简化了安装和配置的过程。使用 `phpstudy-pro`，用户只需下载并安装这个工具，它会自动配置好所需的环境，包括 `PHP` 版本的选择、`MySQL` 数据库的管理，以及 `Apache` 服务器的设置。该工具还提供了友好的用户界面，使得即使是初学者也能轻松上手。此外，`phpstudy-pro` 支持一键启动和停止服务，方便进行开发和调试。这大大减少了手动配置的复杂性，让开发者能够更专注于编写代码，而不是在环境设置上浪费时间。

但需要注意的是，在启动服务时，必须关闭本地的 `MySQL` 80 端口，以避免与该服务器发生冲突。确保端口不被占用，可以顺利启动 `phpstudy-pro` 的 `MySQL` 服务，确保环境正常运行。

在正确启动后我们就可以使用我们的浏览器去访问该服务器。那对应的URL是什么呢？可以是`http://127.0.0.1`，也可以是`http://localhost`。那我们服务器的资源都在哪里呢？我们可以在`phpstudy-pro`找到网站根目录选项，单击就会跳转到该服务器的网站根目录下面。



可以发现网站根目录在WWW文件夹里面。我们就可以在该目录下面创建我们的网页资源然后去访问。

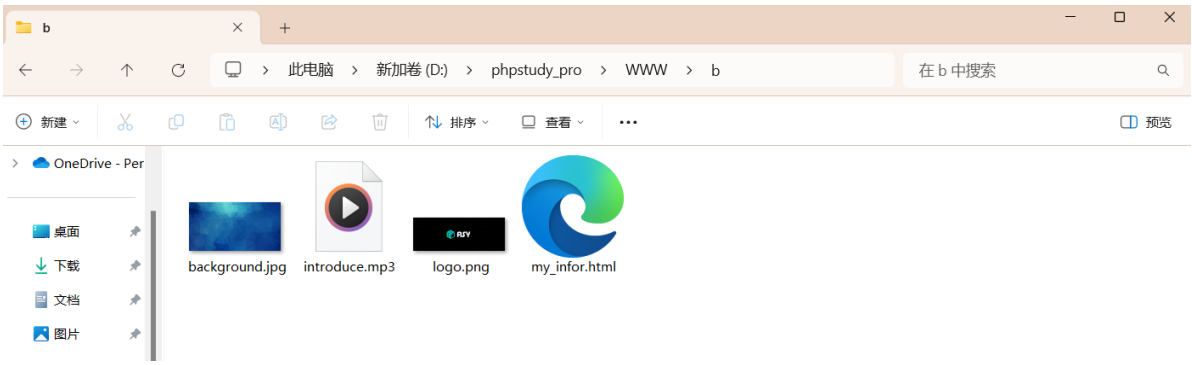
2.2、编写网页

因为本次实验只需要实现一些简单的文本信息，LOGO图片和音频，所以相应的HTML代码并不是很复杂，只是一些美化的代码比较复杂而已，在这里我只取一些核心代码进行解释。

```
<!DOCTYPE html>
<html lang="zh-CN"> <!-- 开始 HTML 文档，设置语言为中文（简体） -->
<head>
    <!-- 头部区域，可以包含标题、元信息、样式链接等 -->
</head>
<body>
    <header> <!-- 页眉区域 -->
        <h1>个人简介</h1> <!-- 主标题 -->
        <img src="logo.png" alt="我的独特 LOGO"> <!-- 显示网站的 LOGO 图片 -->
    </header>

    <div class="content"> <!-- 内容区域 -->
        <h2>个人信息</h2> <!-- 副标题 -->
        <div class="card"> <!-- 用于显示个人信息的卡片 -->
            <p><strong><span style="color: #333;">专业领域：</strong><span
style="color: #333;">信息安全</span></span></p> <!-- 显示专业领域 -->
            <p><strong><span style="color: #333;">学号：</strong><span
style="color: #333;">2210737</span></span></p> <!-- 显示学号 -->
            <p><strong><span style="color: #333;">姓名：</strong><span
style="color: #333;">阿斯雅</span></span></p> <!-- 显示姓名 -->
```

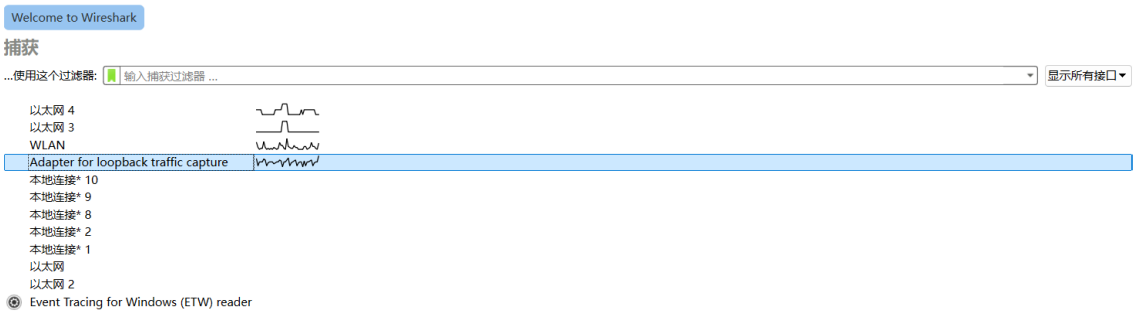
```
        </div>

        <h2>自我介绍</h2> <!-- 副标题 -->

        <a href="introduce.mp3">点击播放</a> <!-- 提供播放自我介绍音频的链接 -->
        <script type="text/javascript" src="http://mediaplayer.yahoo.com/js">
</script> <!-- 引入外部 JavaScript 播放器 -->
    </div>
</body>
</html>
```

要注意的是因为我们要访问的不只是该`html`文件，还有一些`LOGO`图片，自我介绍音频等文件，并且在代码中并没使用他们的绝对地址，所以我们需要把这些文件跟`html`文件放在一个目录下面，所以我把他们统一放在了一个`b`文件夹下面。相应的，我们访问的URL也变成`http://127.0.0.1/b/my_infor.html`。

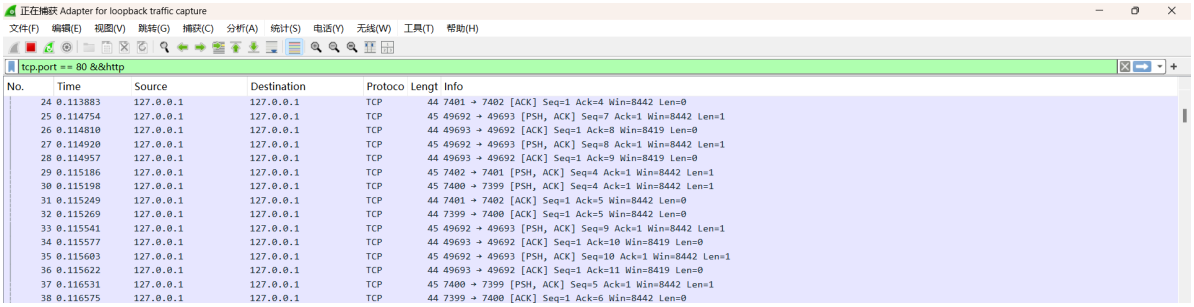


2.3、使用`wireshark`抓包

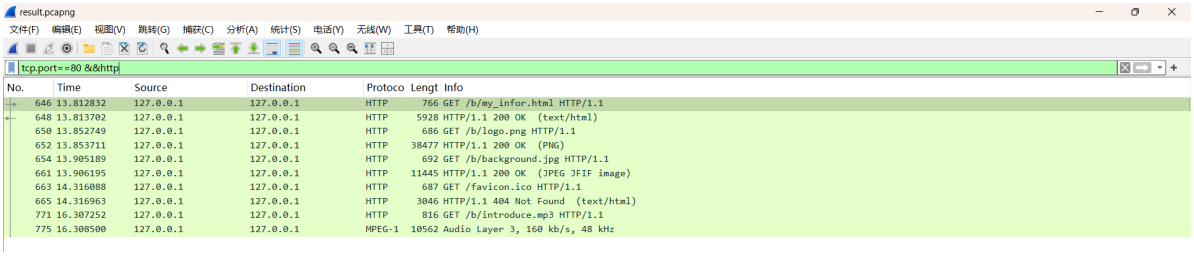
`Wireshark`是一个网络封包分析软件，可以通过该工具监控指定网卡的网络流量。因为在本次实验中我是在本地搭建服务，本地请求，所以需要通过该工具抓取本地127.0.0.1即抓取环回地址的包。具体来说的话我们需要在`Wireshark`中选择`Adapter for loopback traffic capture`网卡进行监控。



我们可以看到点击打开该网卡之后发现有很多条网络记录，那么怎么在这么多条记录中找到我们想要的呢？这就需要使用过滤器来实现。具体来说的话，因为本次实验只需要捕获到`http`协议就可，并且我们访问的端口是80端口，所以可以设置过滤器为：`tcp.port==80 && http`。



在进行上述操作之后发现那些原来的多条记录已经不见了，这时候我们再去访问我们的页面，就可以在`wireshark`中看到记录了。并且进一步可以知道我们浏览器的端口号：40451



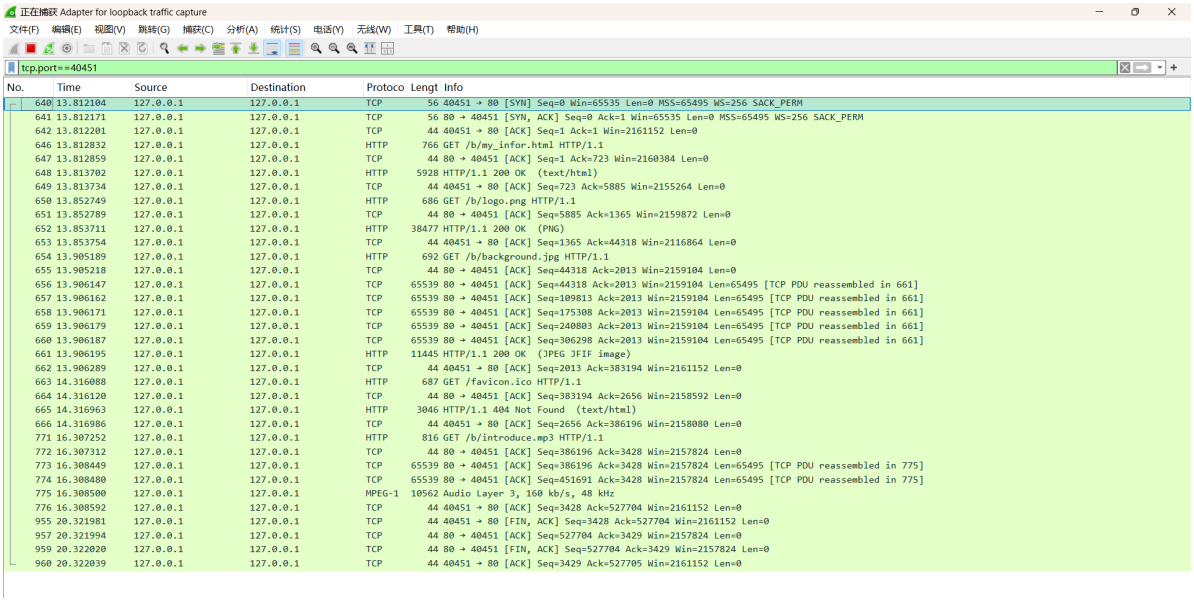
2.4、分析`http`交互过程

在捕获到网络流量之后，我们就可以对浏览器和服务器的`http`交互过程进行分析。

在应用层层面，浏览器和服务器使用`http`协议来进行交互，而我们都知道`http`协议有两种报文：请求报文和响应报文。

请求报文分为三大部分，分别是请求行，请求头和请求体。请求行里面表明了请求方法，请求URL和使用的`http`版本。而在请求体里面包含了一些状态信息。请求体则是使用put和post方法时需要使用的。相应的，响应报文同样分为响应行，响应头和响应体三大部分。其中稍微不一样的是响应行里面是`http`版本和状态码。

知道了这些，我们就可以对交互过程进行分析了。为了更为全面的分析，我们把过滤器条件设置为我们本地浏览器的端口`tcp.port==40451`，就可以得到如下的捕获记录。

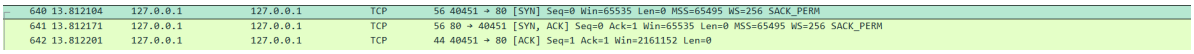


本次实验的交互过程可分为如下六个部分：

1. TCP三次握手客户端服务端进行连接
2. 客户端请求网页资源
3. 客户端请求logo图片资源
4. 客户端请求背景图资源
5. 客户端请求音频资源
6. TCP四次挥手客户端服务端断开连接

接下来我将根据wireshark记录分别解释每个部分。

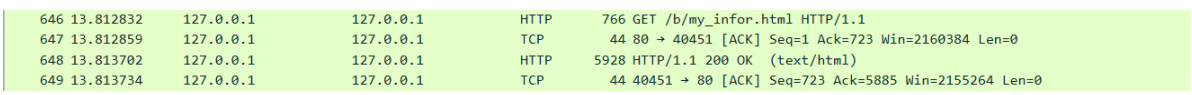
TCP三次握手



从上面捕获到的流量图可以看到，由于应用层协议HTTP是基于传输层TCP协议构建的，所以在数据传输之前，客户端与服务器必须首先通过三次握手建立可靠的连接。在本次实验中，客户端使用端口40451，服务器使用标准的HTTP端口80。整个三次握手过程如下：

1. **客户端发起连接**：客户端向服务器发送一个带有 `SYN`（同步）标志的TCP报文段，表示希望建立连接。这是三次握手的第一步，其中客户端会指定一个初始序列号，并将其放入报文中，作为后续数据传输的起点。
2. **服务器响应连接请求**：服务器收到客户端的 `SYN` 报文后，确认收到该请求，并回复一个带有 `SYN` 和 `ACK`（确认）标志的TCP报文段。此报文段包含服务器的初始序列号，同时通过 `ACK` 字段确认客户端的初始序列号。这一步标志着服务器同意建立连接。
3. **客户端确认连接**：客户端收到服务器的 `SYN-ACK` 报文段后，发送一个带有 `ACK` 标志的报文段进行最终确认，告知服务器它已收到服务器的响应。此时，双方的连接正式建立，可以开始传输应用层的数据。

请求网页资源



```
> GET /b/my_infor.html HTTP/1.1\r\n
  Host: 127.0.0.1\r\n
  Connection: keep-alive\r\n
  sec-ch-ua: "Chromium";v="130", "Microsoft Edge";v="130", "Not?A_Brand";v="99"\r\n
  sec-ch-ua-mobile: ?0\r\n
  sec-ch-ua-platform: "Windows"\r\n
  Upgrade-Insecure-Requests: 1\r\n
  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.0.0 Safari/537.36 Edg/130.0.0.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7\r\n
  Sec-Fetch-Site: none\r\n
  Sec-Fetch-Mode: navigate\r\n
  Sec-Fetch-User: ?1\r\n
  Sec-Fetch-Dest: document\r\n
  Accept-Encoding: gzip, deflate, br, zstd\r\n
  Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6\r\n
  \r\n
  [Response in frame: 648]
  [Full request URI: http://127.0.0.1/b/my_infor.html]
```

在三次握手结束，建立可靠性连接之后，客户端就可以向服务端请求资源。我们可以从上面记录看到本地`127.0.0.1`向本地服务器`127.0.0.1`使用GET方法请求了`b/my_infor.html`这个网页资源。我们可以双击该记录查看更详细的信息。可以看到请求行里面包含了请求`URL`和`Http`版本信息，而在请求头里面则是记录了一些请求的主机名，发出请求的客户端软件信息，指示客户端能够接受的语言类型等信息。而因为使用的是GET请求方法，所以就没有相应的请求体。

```
v Hypertext Transfer Protocol
  > HTTP/1.1 200 OK\r\n
    Date: Wed, 30 Oct 2024 13:34:31 GMT\r\n
    Server: Apache/2.4.39 (Win64) OpenSSL/1.1.1b mod_fcgid/2.3.9a mod_log_rotate/1.02\r\n
    Last-Modified: Fri, 25 Oct 2024 01:32:18 GMT\r\n
    ETag: "1840-6254316f0dc86"\r\n
    Accept-Ranges: bytes\r\n
```

接着可以看到服务器对客户端的GET请求进行了响应，发送一个`ACK`包，其中`Seq=1`和`Ack=731`表示确认了客户端的请求。之后服务器向客户端发送HTTP响应报文，而可以看到响应的状态码为`200 OK`，这表示客户端的请求成功，并且在响应体中包含了一些服务端的信息，如系统信息。而当客户端请求到资源后，也要给服务端发送`ACK`确认收到服务器的响应。

请求LOGO图片资源

```
650 13.852749    127.0.0.1       127.0.0.1       HTTP      686 GET /b/logo.png HTTP/1.1
651 13.852789    127.0.0.1       127.0.0.1       TCP        44 80 → 40451 [ACK] Seq=5885 Ack=1365 Win=2159872 Len=0
652 13.853711    127.0.0.1       127.0.0.1       HTTP    38477 HTTP/1.1 200 OK  (PNG)
653 13.853754    127.0.0.1       127.0.0.1       TCP        44 40451 → 80 [ACK] Seq=1365 Ack=44318 Win=2116864 Len=0
```

因为本次实验要请求的资源并不只是该网页资源，所以在得到了网页资源后，需要接着去请求网页里面嵌入的一些资源，如LOGO图片，背景图片和音频。其中请求LOGO图片的逻辑跟上述请求网页资源流程一模一样，所以就不再进行解释。

请求背景图片资源

```
654 13.905189    127.0.0.1       127.0.0.1       HTTP      692 GET /b/background.jpg HTTP/1.1
655 13.905218    127.0.0.1       127.0.0.1       TCP        44 80 → 40451 [ACK] Seq=44318 Ack=2013 Win=2159104 Len=0
656 13.906147    127.0.0.1       127.0.0.1       TCP     65539 80 → 40451 [ACK] Seq=44318 Ack=2013 Win=2159104 Len=65495 [TCP PDU reassembled in 661]
657 13.906162    127.0.0.1       127.0.0.1       TCP     65539 80 → 40451 [ACK] Seq=109813 Ack=2013 Win=2159104 Len=65495 [TCP PDU reassembled in 661]
658 13.906171    127.0.0.1       127.0.0.1       TCP     65539 80 → 40451 [ACK] Seq=175308 Ack=2013 Win=2159104 Len=65495 [TCP PDU reassembled in 661]
659 13.906179    127.0.0.1       127.0.0.1       TCP     65539 80 → 40451 [ACK] Seq=240803 Ack=2013 Win=2159104 Len=65495 [TCP PDU reassembled in 661]
660 13.906187    127.0.0.1       127.0.0.1       TCP     65539 80 → 40451 [ACK] Seq=306298 Ack=2013 Win=2159104 Len=65495 [TCP PDU reassembled in 661]
661 13.906195    127.0.0.1       127.0.0.1       HTTP    11445 HTTP/1.1 200 OK  (JPEG JFIF image)
662 13.906289    127.0.0.1       127.0.0.1       TCP        44 40451 → 80 [ACK] Seq=2013 Ack=383194 Win=2161152 Len=0
```

通过实验我发现，请求背景图片`background.jpgd`的过程与请求LOGO图片的方式稍有不同，因此在此详细介绍一下请求`background.jpg`的过程。首先，客户端依旧通过`GET`方法请求该资源，服务端接收到请求后发送一个`ACK`包，确认请求已收到，并开始数据传输。

而编号656到660的记录展示了TCP数据包的传输，其中每个数据包都包含`ACK`标志，表明已确认接收到数据包。这些数据包中包含了文件`background.jpg`的数据，且它们的长度（Len字段）较大，表明携带了大量数据。此外，这些包上标有“`PDU reassembled in 661`”标记，意味着这些数据包会在编号661的位置重新组装成一个完整的数据单元。

```
[6 Reassembled TCP Segments (338876 bytes): #656(65495), #657(65495), #658(65495), #659(65495), #660(65495), #661(11401)]
    [Frame: 656, payload: 0-65494 (65495 bytes)]
    [Frame: 657, payload: 65495-130989 (65495 bytes)]
    [Frame: 658, payload: 130990-196484 (65495 bytes)]
    [Frame: 659, payload: 196485-261979 (65495 bytes)]
    [Frame: 660, payload: 261980-327474 (65495 bytes)]
    [Frame: 661, payload: 327475-338875 (11401 bytes)]
    [Segment count: 6]
    [Reassembled TCP length: 338876]
    [Reassembled TCP Data […]: 485454502f312e3120323030204f4b0d0a446174653a205765642c203330204f63742032303234203135a30393a343620474d540d0a5365727665723a204170616…
```

双击编号661，我们确实可以发现在这里前面的656到661的数据被重新组装成一个大数据流。具体来说的话：

Reassembled TCP Segments：表示总共有338876字节的数据被重新组装。

Frame Numbers：列出了六个帧的编号，分别是#656、#657、#658、#659、#660和#661。

Payload：每个帧的有效载荷（payload）大小，其中前五个帧的payload大小都是65495字节，最后一个帧的payload大小是11401字节。

Frame Details：详细列出了每个帧的编号和它们对应的payload范围。例如，`Frame:656`的payload是从0到65494字节，`Frame:657`的payload是从65495到130989字节，以此类推。

> 我猜测这是因为我的背景图片文件过大的原因，所以才会在传输过程中被分成好几个段，然后在目的地进行重装。

而在传输完成后，服务端发送响应报文，状态码为200，表示请求成功。最后，客户端发送一个`ACK`包以确认完整数据单元的接收，整个资源请求过程结束

请求音频资源

```
771 16.307252    127.0.0.1       127.0.0.1       HTTP      816 GET /b/introduce.mp3 HTTP/1.1
772 16.307312    127.0.0.1       127.0.0.1       TCP        44 80 → 40451 [ACK] Seq=386196 Ack=3428 Win=2157824 Len=0
773 16.308449    127.0.0.1       127.0.0.1       TCP     65539 80 → 40451 [ACK] Seq=386196 Ack=3428 Win=2157824 Len=65495 [TCP PDU reassembled in 775]
774 16.308480    127.0.0.1       127.0.0.1       TCP     65539 80 → 40451 [ACK] Seq=451691 Ack=3428 Win=2157824 Len=65495 [TCP PDU reassembled in 775]
775 16.308500    127.0.0.1       127.0.0.1       MPEG-1  10562 Audio Layer 3, 160 kb/s, 48 kHz
776 16.308592    127.0.0.1       127.0.0.1       TCP        44 40451 → 80 [ACK] Seq=3428 Ack=527704 Win=2161152 Len=0
```

最后请求的是音频资源，大致流程跟请求上面的`background.jpg`步骤差不多，首先还是客户端使用GET方法请求`mp3`资源，然后服务端返回`ACK`包表示接收到请求，并把`mp3`文件分成几个数据包在775进行重组。其中有一点不一样的是，在记录775这里显示了`MP3`文件的数据传输，还提供了音频文件的详细信息。`MP3`文件使用`MPEG-1 Audio Layer 3`编码格式，数据包大小为10562字节。其比特率为`160 kb/s`，采样率为`48 kHz`。这表明服务器正在发送`mp3`文件的实际音频数据，该数据包经过之前的分块，在此处重组形成完整的数据流，供客户端接收和处理。

TCP四次挥手

```
955 20.321981    127.0.0.1       127.0.0.1       TCP        44 40451 → 80 [FIN, ACK] Seq=3428 Ack=527704 Win=2161152 Len=0
957 20.321994    127.0.0.1       127.0.0.1       TCP        44 80 → 40451 [ACK] Seq=527704 Ack=3429 Win=2157824 Len=0
959 20.322020    127.0.0.1       127.0.0.1       TCP        44 80 → 40451 [FIN, ACK] Seq=527704 Ack=3429 Win=2157824 Len=0
960 20.322039    127.0.0.1       127.0.0.1       TCP        44 40451 → 80 [ACK] Seq=3429 Ack=527705 Win=2161152 Len=0
```

最后就是当我们关掉浏览器关掉网页时，这时候客户端和服务端的`tcp`要进行四次挥手从而断开连接，具体步骤如下：

1. 客户端向服务器发送一个`FIN`包，请求关闭连接。`FIN`包用于表示客户端已经发送完所有数据，准备断开连接。
2. 服务器接收到FIN包后，会发送一个`ACK`包作为回应，确认收到了客户端的FIN包。
3. 服务器在发送完所有剩余数据后，也会向客户端发送一个`FIN`包，表示服务器也完成了数据的发送，请求关闭连接。
4. 客户端接收到服务器的FIN包后，会发送一个`ACK`包作为最后的回应，确认收到了服务器的FIN包。

三、实验难点

本次实验的主要难点并不是在编写网页或者是搭建服务器上，而是怎么在wireshark中找到对应的记录并分析。

- **该监控哪个网卡**

  我们打开`wireshark`之后可以发现他给我们罗列出了我们本地所有的网卡。有一些是以太网的，有一些是无线网的，有一些也是专门来监控本地回环的。这就要根据我们搭的服务器来进行选择。如果像我一样在本地搭建的服务器，就要对`Adapter for loopback traffic capture`网卡进行监控，而如果是租的服务器，就要选择相应的以太网接口来进行监控。

- **该怎么设置过滤器**

  找到正确的网卡之后，其实还有一个难点，就是就算是正确的网卡，它里面也会有很多条协议的记录。所以我们要设置过滤器来进行过滤。比如这次我们只需要看`http`协议，所以可以在过滤器里面设置`http`这样的过滤器。但为了更为全面的分析，用`tcp`端口号来设置过滤器其实更为好。具体来说就是我们找到我们浏览器进程的端口号，然后以它作为过滤器，就可以查看到浏览器跟服务器交互的记录，并且没有别的杂项。

- **该怎么分析记录**

  这个就要要求我们对TCP可靠性连接的交互过程和`http`协议的两种报文要有清晰的认识。比如`tcp`的三次握手过程和四次挥手过程，`http`的各种请求方法和响应状态码等等。