计算机网络第二次实验报告

邢清画 2211999 物联网工程

一、实验目的

配置Web服务器，分析HTTP交互过程

二、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。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。

三、实验过程

3.1 web服务器设计

3.1.1 服务器选型与开发环境

- 本实验选用 Flask 框架作为 Web 服务器，Flask 是一个轻量级的 Python Web 框架，适合快速开发和测试。
- 开发环境为 Python 3.12，通过命令`pip install Flask` 确保 Flask 和相关依赖库已安装。

3.1.2 项目目录结构

根据 Flask 的项目结构，设计了以下目录结构：

```
web2/
├── app.py            # Flask应用主文件
├── static/           # 存放静态资源的文件夹（如图片、音频文件）
|   ├── logo.png      # 个人LOGO图片
|   ├── ENFJ.png
|   ├── ENFP.png
|   ├── ESFP.png
|   ├── intro.mp3     # 个人介绍音频
|   └── background.png # 页面背景图片
└── templates/
    └── index.html    # HTML模板文件
```

3.1.3 HTML界面设计

`index.html` 为**主页面模板**，包含个人基本信息（如姓名、学号、专业等）、专业技能、视频展示以及其他信息（兴趣爱好、MBTI 类型）等模块。

页面设计包含如下几个主要部分：

- **个人信息**：展示姓名、学号、专业，并添加个人 LOGO 和音频自我介绍。
- **专业技能**：列出熟悉的编程语言和技术栈。
- **成果展示**：链接到一些项目相关视频。
- **其他信息**：展示兴趣爱好、MBTI 类型。

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
    <meta charset="UTF-8">
    <title>实验2页面</title>
</head>
<body>
    <div class="container">
        <h2>基本信息</h2>
        <div class="header">
            <!-- 个人LOGO图片 -->
            <div class="logo">
                <img src="{{ url_for('static', filename='logo.png') }}" alt="个人
LOGO" style="width: 100px; height: auto;">
            </div>
            <!-- 个人信息 -->
            <div class="info">
                <p>姓名：邢清画</p>
                <p>学号：2211999</p>
                <p>专业：物联网工程</p>
                <!-- 音频播放控件 -->
                <audio controls>
                    <source src="{{ url_for('static', filename='intro.mp3') }}"
type="audio/mpeg">
                    您的浏览器不支持音频元素。
                </audio>
            </div>
        </div>

        <hr>

        <!-- 专业技能部分 -->
        <h2>专业技能</h2>
        <ul>
            <li>学习并使用多门编程语言：C，C++，Java，Python，汇编等。</li>
            <li>可以实现基本的数据库处理，前端设计，算法设计。</li>
        </ul>

        <hr>
```

```
        <!-- 个人展示部分 -->
        <h2>相关视频</h2>
        <ul>
            <li>2023 南开大学C++程序设计--doodle jump</li>-->
            <a href="https://www.bilibili.com/video/BV1rT411h7C3/?
spm_id_from=333.337.search-card.all.click" target="_blank" title="2023 南开大学C++程序
设计--doodle jump">Video path:https://www.bilibili.doodle jump</a>

            <li>2023 南开大学虚假新闻检测高校赛</li>
            <a href="https://www.bilibili.com/video/BV1eg4y1o7iw/?
spm_id_from=333.337.search-card.all.click" target="_blank" title="2023 南开大学虚假新闻
检测高校赛">Video path:https://www.bilibili.nlp_fakenewsdetection</a>
    </ul>

        <hr>

<!-- 其他信息部分 -->
<h2>其他信息</h2>
<p><strong>兴趣爱好：</strong>羽毛球、乒乓球、听歌、旅游、素描、电子竞技</p>

<!-- MBTI和星座部分 -->
<p><strong>MBTI：</strong>
    <span style="display: inline-flex; align-items: center; gap: 10px;">
        <span style="display: inline-flex; align-items: center;">
            <img src="{{ url_for('static', filename='ENFJ.png') }}" alt="ENFJ"
title="ENFJ" style="width: 32px; height: auto;">
            <span style="margin-left: 5px;">ENFJ</span>
        </span>
        <span>-></span>
        <span style="display: inline-flex; align-items: center;">
            <img src="{{ url_for('static', filename='ESFP.png') }}" alt="ESFP"
title="ESFP" style="width: 32px; height: auto;">
            <span style="margin-left: 5px;">ESFP</span>
        </span>
        <span>-></span>
        <span style="display: inline-flex; align-items: center;">
            <img src="{{ url_for('static', filename='ENFP.png') }}" alt="ENFP"
title="ENFP" style="width: 32px; height: auto;">
            <span style="margin-left: 5px;">ENFP</span>
        </span>
        <span>-></span>
        <span style="display: inline-flex; align-items: center;">
            <img src="{{ url_for('static', filename='ENFJ.png') }}" alt="ENFJ"
title="ENFJ" style="width: 32px; height: auto;">
            <span style="margin-left: 5px;">ENFJ</span>
        </span>
        </span>
    </span>
</p>
    </div>
</body>
```

```
</html>
```

3.1.4 CSS 样式和布局

- 为了保证页面的美观，使用内嵌 CSS 样式设置页面布局和元素样式。整体页面使用 `flex` 布局使内容居中，容器部分添加了边框和阴影效果。
- 页面背景设置为静态图片，且使用透明度调整，以确保背景不干扰内容的可读性。背景图片使用 `background-attachment: fixed` 使其固定在页面中心。

```
    <style>
        body {
            font-family: Arial, sans-serif;
            display: flex;
            justify-content: center;
            align-items: center;
            margin: 0;
            padding: 20px;
            background: rgba(255, 255, 255, 0.8) url('{{ url_for('static',
filename='background.png') }}') no-repeat center center;
            background-size: cover;
            background-attachment: fixed;
            min-height: 100vh;
        }
        .container {
            border: 3px solid #ccc;
            padding: 20px;
            max-width: 800px;
            width: 100%;
            box-sizing: border-box;
            background-color: rgba(255, 255, 255, 0.7); /* 容器的半透明背景 */
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        }
        .header {
            display: flex;
            align-items: center;
            margin-bottom: 20px;
        }
        .logo {
            margin-right: 20px;
        }
        .info {
            line-height: 1.6;
        }
        h1, h2 {
            font-weight: bold;
            text-align: left;
            margin-top: 0;
        }
        hr {
            margin: 20px 0;
        }
```

```
        ul {
            list-style-type: disc;
            padding-left: 20px;
        }
        a {
            color: blue;
            text-decoration: none;
        }
        a:hover {
            text-decoration: underline;
        }
        /* 响应式设计 */
        @media (max-width: 600px) {
            .container {
                padding: 10px;
            }
            .header {
                flex-direction: column;
                align-items: flex-start;
            }
            .logo {
                margin-right: 0;
                margin-bottom: 10px;
            }
        }
    </style>
```

3.1.5 服务器配置与启动

- `app.py` 文件定义了 Flask 服务器的主要逻辑，通过 `@app.route('/')` 路由指向 `index.html` 页面。
- 服务器通过以下代码启动，并在本地 `80` 端口监听 HTTP 请求：

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80)
```

3.1.6 Web 页面展示与测试

1. 启动服务器：

- 在项目根目录下，使用终端或命令行运行该命令，或直接运行app.py文件来启动 Flask 服务器：

```
sudo python app.py
```

- 启动后，终端会显示服务器运行状态

2. 打开网页：

- 通过浏览器访问**服务器地址**（http://127.0.0.1或http://localhost）或点击终端地址链接
- 检查页面是否正确加载，包括图片、音频、背景图等资源的显示和布局。

如上图所示，页面可以正确加载所有资源，图片、背景可以正常显示，音频文件也可以正常播放。

```
127.0.0.1 - - [30/Oct/2024 15:25:52] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [30/Oct/2024 15:25:54] "GET /static/ENFJ.png HTTP/1.1" 304 -
127.0.0.1 - - [30/Oct/2024 15:25:54] "GET /static/logo.png HTTP/1.1" 304 -
127.0.0.1 - - [30/Oct/2024 15:25:54] "GET /static/ENFP.png HTTP/1.1" 304 -
127.0.0.1 - - [30/Oct/2024 15:25:54] "GET /static/intro.mp3 HTTP/1.1" 206 -
127.0.0.1 - - [30/Oct/2024 15:25:54] "GET /static/ESFP.png HTTP/1.1" 304 -
127.0.0.1 - - [30/Oct/2024 15:25:54] "GET /static/background.png HTTP/1.1" 304 -
```

GET / HTTP/1.1：表示 HTTP 请求的具体内容，包含三个部分：

- 请求方法（GET）：表示请求的操作类型，GET 表示获取资源。
- 请求路径（/ 或 /static/intro.mp3）：请求的资源路径，例如 `/` 表示根页面，`/static/intro.mp3` 表示音频文件。
- HTTP 协议版本（HTTP/1.1）：HTTP 的版本。

状态码（例如 `200`、`304` 和 `206`）：

- `200 OK`：表示**请求成功**，服务器返回了请求的资源。
- `304 Not Modified`：表示请求的**资源未被修改**，客户端可以使用缓存的版本，无需从服务器重新下载。
- `206 Partial Content`：表示**部分内容响应**，通常用于音频或视频文件的分段下载，适用于流式传输。

3. **测试页面响应与布局：**

- 在**不同浏览器**和**窗口大小**下测试页面，确保网页在各种设备上的适应性，所有内容均在设计的边框范围内显示。
- 确认**音频**控件、**图片**加载和**页面背景**在缩放时效果一致。

四、实验结果

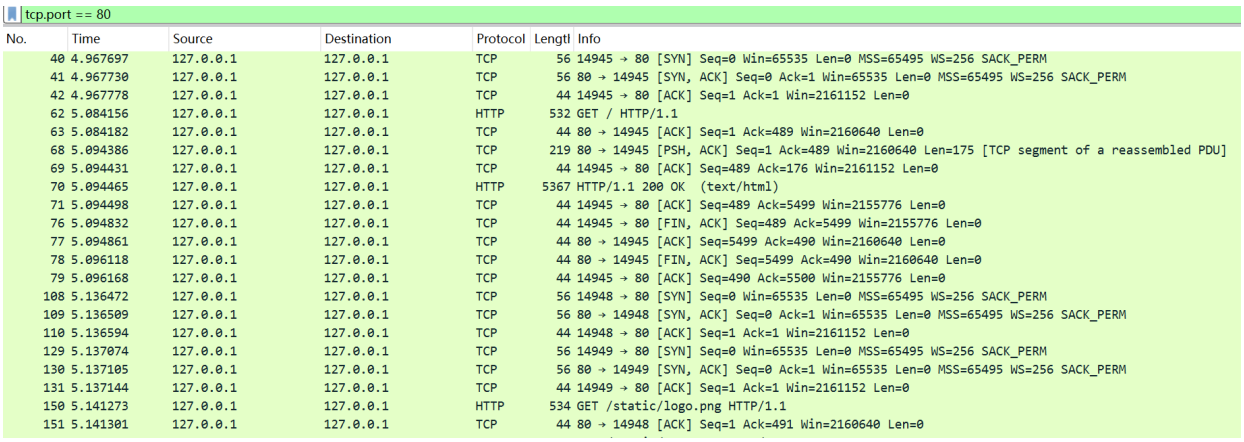
4.1 Tcp 建立连接

打开 Wireshark，选择 “Adapter for loopback traffic capture” 接口。在 Wireshark 捕获的 TCP 报文中会看到多个字段，这些字段为分析 TCP 三次握手以及数据传输过程提供了重要信息。

初始过滤条件：在 Wireshark 中使用以下过滤条件来捕获 HTTP 以及 TCP 的三次握手和四次挥手的过程：

```
tcp.port == 80
```

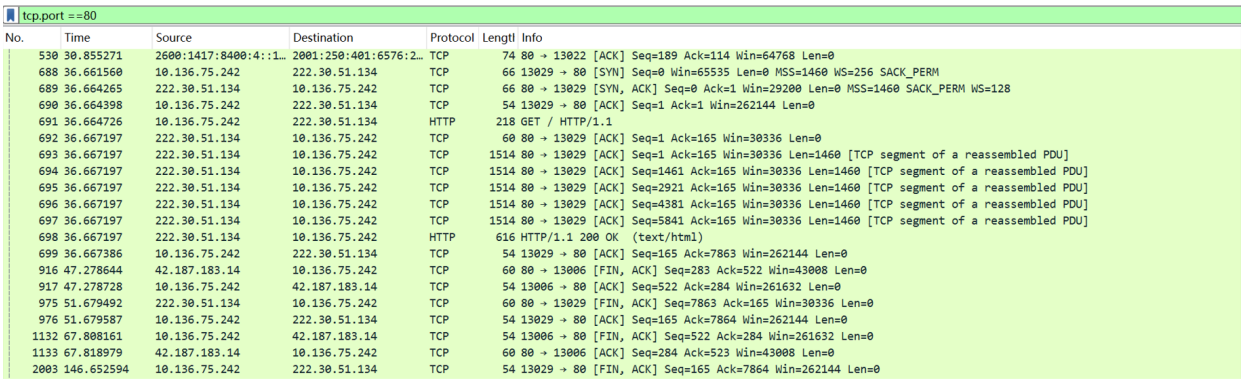
显示所有通过 TCP 80 端口的流量，包括 HTTP 请求及其对应的三次握手和四次挥手过程。



tcp.port == 80

No.	Time	Source	Destination	Protocol	Length	Info
40	4.967697	127.0.0.1	127.0.0.1	TCP	56	14945 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
41	4.967730	127.0.0.1	127.0.0.1	TCP	56	80 → 14945 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
42	4.967778	127.0.0.1	127.0.0.1	TCP	44	14945 → 80 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
62	5.084156	127.0.0.1	127.0.0.1	HTTP	532	GET / HTTP/1.1
63	5.084182	127.0.0.1	127.0.0.1	TCP	44	80 → 14945 [ACK] Seq=1 Ack=489 Win=2160640 Len=0
68	5.094386	127.0.0.1	127.0.0.1	TCP	219	80 → 14945 [PSH, ACK] Seq=1 Ack=489 Win=2160640 Len=175 [TCP segment of a reassembled PDU]
69	5.094431	127.0.0.1	127.0.0.1	TCP	44	14945 → 80 [ACK] Seq=489 Ack=176 Win=2161152 Len=0
70	5.094465	127.0.0.1	127.0.0.1	HTTP	5367	HTTP/1.1 200 OK (text/html)
71	5.094498	127.0.0.1	127.0.0.1	TCP	44	14945 → 80 [ACK] Seq=489 Ack=5499 Win=2155776 Len=0
76	5.094832	127.0.0.1	127.0.0.1	TCP	44	14945 → 80 [FIN, ACK] Seq=489 Ack=5499 Win=2155776 Len=0
77	5.094861	127.0.0.1	127.0.0.1	TCP	44	80 → 14945 [ACK] Seq=5499 Ack=490 Win=2160640 Len=0
78	5.096118	127.0.0.1	127.0.0.1	TCP	44	80 → 14945 [FIN, ACK] Seq=5499 Ack=490 Win=2160640 Len=0
79	5.096168	127.0.0.1	127.0.0.1	TCP	44	14945 → 80 [ACK] Seq=490 Ack=5500 Win=2155776 Len=0
108	5.136472	127.0.0.1	127.0.0.1	TCP	56	14948 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
109	5.136509	127.0.0.1	127.0.0.1	TCP	56	80 → 14948 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
110	5.136594	127.0.0.1	127.0.0.1	TCP	44	14948 → 80 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
129	5.137074	127.0.0.1	127.0.0.1	TCP	56	14949 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
130	5.137105	127.0.0.1	127.0.0.1	TCP	56	80 → 14949 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
131	5.137144	127.0.0.1	127.0.0.1	TCP	44	14949 → 80 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
150	5.141273	127.0.0.1	127.0.0.1	HTTP	534	GET /static/logo.png HTTP/1.1
151	5.141301	127.0.0.1	127.0.0.1	TCP	44	80 → 14948 [ACK] Seq=1 Ack=491 Win=2160640 Len=0

（本地回环接口127.0.0.1）



tcp.port ==80

No.	Time	Source	Destination	Protocol	Lengtl	Info
530	30.855271	2600:1417:8400:4::1…	2001:250:401:6576:2…	TCP	74	80 → 13022 [ACK] Seq=189 Ack=114 Win=64768 Len=0
688	36.661560	10.136.75.242	222.30.51.134	TCP	66	13029 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
689	36.664265	222.30.51.134	10.136.75.242	TCP	66	80 → 13029 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM WS=128
690	36.664398	10.136.75.242	222.30.51.134	TCP	54	13029 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
691	36.664726	10.136.75.242	222.30.51.134	HTTP	218	GET / HTTP/1.1
692	36.667197	222.30.51.134	10.136.75.242	TCP	60	80 → 13029 [ACK] Seq=1 Ack=165 Win=30336 Len=0
693	36.667197	222.30.51.134	10.136.75.242	TCP	1514	80 → 13029 [ACK] Seq=1 Ack=165 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
694	36.667197	222.30.51.134	10.136.75.242	TCP	1514	80 → 13029 [ACK] Seq=1461 Ack=165 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
695	36.667197	222.30.51.134	10.136.75.242	TCP	1514	80 → 13029 [ACK] Seq=2921 Ack=165 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
696	36.667197	222.30.51.134	10.136.75.242	TCP	1514	80 → 13029 [ACK] Seq=4381 Ack=165 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
697	36.667197	222.30.51.134	10.136.75.242	TCP	1514	80 → 13029 [ACK] Seq=5841 Ack=165 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
698	36.667197	222.30.51.134	10.136.75.242	HTTP	616	HTTP/1.1 200 OK (text/html)
699	36.667386	10.136.75.242	222.30.51.134	TCP	54	13029 → 80 [ACK] Seq=165 Ack=7863 Win=262144 Len=0
916	47.278644	42.187.183.14	10.136.75.242	TCP	60	80 → 13006 [FIN, ACK] Seq=283 Ack=522 Win=43008 Len=0
917	47.278728	10.136.75.242	42.187.183.14	TCP	54	13006 → 80 [ACK] Seq=522 Ack=284 Win=261632 Len=0
975	51.679492	222.30.51.134	10.136.75.242	TCP	60	80 → 13029 [FIN, ACK] Seq=7863 Ack=165 Win=30336 Len=0
976	51.679587	10.136.75.242	222.30.51.134	TCP	54	13029 → 80 [ACK] Seq=165 Ack=7864 Win=262144 Len=0
1132	67.808161	10.136.75.242	42.187.183.14	TCP	54	13006 → 80 [FIN, ACK] Seq=522 Ack=284 Win=261632 Len=0
1133	67.818979	42.187.183.14	10.136.75.242	TCP	60	80 → 13006 [ACK] Seq=284 Ack=523 Win=43008 Len=0
2003	146.652594	10.136.75.242	222.30.51.134	TCP	54	13029 → 80 [FIN, ACK] Seq=165 Ack=7864 Win=262144 Len=0

（WLAN接口）

1. **Source（源地址） 和 Destination（目的地址）：**
   - 这两个字段显示数据包的发送方和接收方的 IP 地址。
   - 例如，在WLAN下`10.136.75.242` 表示客户端的 IP 地址，而 `222.30.51.134` 表示服务器的 IP 地址；本地回环均为 `127.0.0.1`。
2. **Protocol（协议）：**
   - 该字段显示数据包使用的协议类型。此处 `TCP` 表示数据包使用了 TCP 协议进行传输，而 `HTTP` 表示 HTTP 数据包。
3. **Info（信息）：**
   - SYN：建立连接时的请求标志，表示请求建立新连接。
   - SYN, ACK：服务器对客户端的 SYN 请求做出响应，同时带有确认标志 ACK，表示接收并同意连接请求。
   - ACK：客户端发送的确认包，表示已收到服务器的 `SYN, ACK` 包，确认连接建立。

（这也是和三次握手紧密相关的字段）

4. **Seq（Sequence Number，序列号）：**
   - 序列号用于标识 TCP 流中数据包的顺序。TCP 是基于序列号的可靠传输协议，双方通过 Seq 值确认数据包的顺序和完整性。
   - `Seq=0` 表示连接建立时的初始序列号，之后每个包的 Seq 值会根据数据量依次增加。
5. **Ack（Acknowledgment Number，确认号）：**
   - 确认号表示接收方期望的下一个序列号，用于确认已接收到的数据包。TCP 使用该值来确保数据包的顺序和完整性。
   - `Ack=1` 表示接收方期望下一个数据包的序列号为 1。
6. **Win（Window Size，窗口大小）：**
   - 窗口大小表示接收方当前可接收的数据量，控制数据流的速度，以避免网络拥塞。
   - `Win=65535` 表示接收方可以接收最大 65535 字节的数据。
7. **Len（Length，长度）：**
   - 该字段表示 TCP 数据段的长度，即数据包中有效数据的字节数。三次握手时通常为 0，因为此时仅用于建立连接。

8. MSS（Maximum Segment Size，最大报文段大小）：

- MSS 是 TCP 连接中每次传输的最大数据量。MSS 值在连接建立期间协商，并影响后续数据包的大小。
- 例如，`MSS=1460` 表示最大报文段大小为 1460 字节。

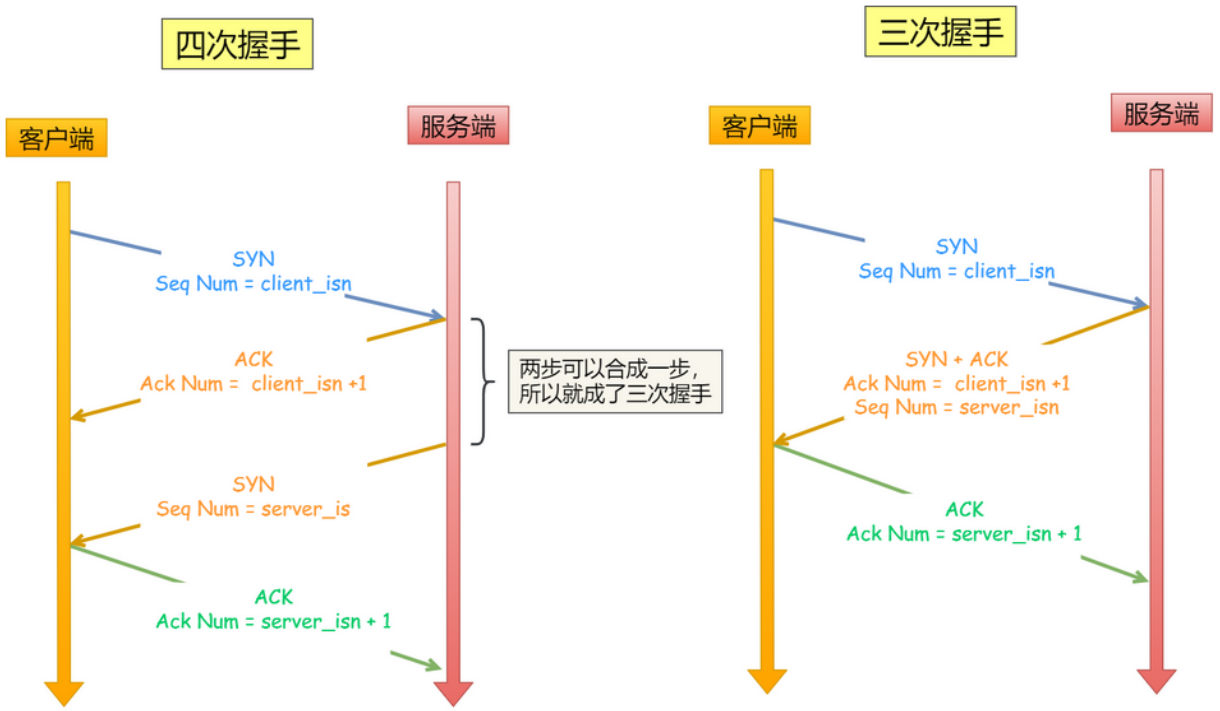
9. WS（Window Scale，窗口缩放）：

- 窗口缩放因子用于扩大窗口大小的范围，在高带宽网络中提高传输效率。`WS=256` 表示窗口大小将被扩大 256 倍。

10. SACK_PERM（Selective Acknowledgment Permitted，选择性确认允许）：

- SACK 是一种选择性确认机制，用于允许接收方选择性确认已接收的数据块，避免因数据包丢失导致的重复传输。`SACK_PERM` 表示该连接支持 SACK 功能，提高了传输效率。

4.2 三次握手



寻找 `SYN`、`SYN-ACK` 和 `ACK` 包。这三个包标识了客户端和服务器之间的连接建立过程。

Source	Destination	Protocol	Lengtl	Info
127.0.0.1	127.0.0.1	TCP	56	14945 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
127.0.0.1	127.0.0.1	TCP	56	80 → 14945 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
127.0.0.1	127.0.0.1	TCP	44	14945 → 80 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
127.0.0.1	127.0.0.1	HTTP	532	GET / HTTP/1.1
127.0.0.1	127.0.0.1	TCP	44	80 → 14945 [ACK] Seq=1 Ack=489 Win=2160640 Len=0
127.0.0.1	127.0.0.1	TCP	219	80 → 14945 [PSH, ACK] Seq=1 Ack=489 Win=2160640 Len=175 [TCP segment of a reassembled PDU]
127.0.0.1	127.0.0.1	TCP	44	14945 → 80 [ACK] Seq=489 Ack=176 Win=2161152 Len=0
127.0.0.1	127.0.0.1	HTTP	5367	HTTP/1.1 200 OK (text/html)
127.0.0.1	127.0.0.1	TCP	44	14945 → 80 [ACK] Seq=489 Ack=5499 Win=2155776 Len=0

4.2.1 第一次握手

```
> Frame 40: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
v Transmission Control Protocol, Src Port: 14945, Dst Port: 80, Seq: 0, Len: 0
    Source Port: 14945
    Destination Port: 80
    [Stream index: 5]
  > [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 1279230484
    [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: 0x6de5 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
```

- 客户端 `127.0.0.1` 端口 `14945` 向服务器 `127.0.0.1`端口 `80` 发送 `SYN` 包，请求建立连接。
- 标志位`SYN`，序列号 `Seq=0`，窗口大小 `Win=65535`，表示初始请求。
- 进入 `SYN_SENT` 状态。

4.2.2 第二次握手

```
Frame 41: 56 bytes on wire (448 bits), 56 bytes captured (448 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: 80, Dst Port: 14945, Seq: 0, Ack: 1, Len: 0
  Source Port: 80
  Destination Port: 14945
  [Stream index: 5]
> [Conversation completeness: Complete, WITH_DATA (31)]
  [TCP Segment Len: 0]
  Sequence Number: 0    (relative sequence number)
  Sequence Number (raw): 2627025271
  [Next Sequence Number: 1    (relative sequence number)]
  Acknowledgment Number: 1    (relative ack number)
  Acknowledgment number (raw): 1279230485
  1000 .... = Header Length: 32 bytes (8)
> Flags: 0x012 (SYN, ACK)
  Window: 65535
  [Calculated window size: 65535]
  Checksum: 0x97c7 [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
```

- 服务器 `127.0.0.1` 回复 `SYN, ACK` 包，确认连接请求。
- 标志位`SYN, ACK`，序列号 `Seq=0`，确认号 `Ack=1`，窗口大小`Win=65535`。
- 此时服务器进入 `SYN_RCVD` 状态。

4.2.3 第三次握手（No. 68）

```
> Frame 42: 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
v Transmission Control Protocol, Src Port: 14945, Dst Port: 80, Seq: 1, Ack: 1, Len: 0
    Source Port: 14945
    Destination Port: 80
    [Stream index: 5]
  > [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 1279230485
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 2627025272
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x010 (ACK)
    Window: 8442
    [Calculated window size: 2161152]
    [Window size scaling factor: 256]
    Checksum: 0xb1c4 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
```

- 客户端发送 `ACK` 包。
- 标志位`ACK`，序列号 `Seq=1`，确认号 `Ack=1`，表示收到服务器的 `SYN+ACK` 包。
- 进入 `ESTABLISHED` 状态，连接建立完成。

经过上述的三次握手，TCP 连接正式建立，双方都置为 ACK flag，交换并确认了对方的初始序列号。

4.3 HTTP协议

4.3.1 Adapter for loopback traffic capture 接口（本地回环）

打开 Wireshark，选择 “Adapter for loopback traffic capture” 接口。过滤出只使用HTTP协议 的数据包：

http

No.	Time	Source	Destination	Protocol	Lengtl	Info
62	5.084156	127.0.0.1	127.0.0.1	HTTP	532	GET / HTTP/1.1
70	5.094465	127.0.0.1	127.0.0.1	HTTP	5367	HTTP/1.1 200 OK (text/html)
83	5.110778	127.0.0.1	127.0.0.1	HTTP	170	CONNECT 61.151.183.16:80 HTTP/1.1
85	5.110971	127.0.0.1	127.0.0.1	HTTP	83	HTTP/1.1 200 Connection established
87	5.111157	127.0.0.1	127.0.0.1	TCP	175	14946 → 7890 [PSH, ACK] Seq=127 Ack=40 Win=2161152 Len=131
150	5.141273	127.0.0.1	127.0.0.1	HTTP	534	GET /static/logo.png HTTP/1.1
156	5.141429	127.0.0.1	127.0.0.1	HTTP	534	GET /static/ENFJ.png HTTP/1.1
186	5.142351	127.0.0.1	127.0.0.1	HTTP	534	GET /static/ESFP.png HTTP/1.1
224	5.144798	127.0.0.1	127.0.0.1	HTTP	632	GET /static/ENFP.png HTTP/1.1
258	5.145428	127.0.0.1	127.0.0.1	HTTP	540	GET /static/background.png HTTP/1.1
316	5.211435	127.0.0.1	127.0.0.1	HTTP	660	GET /static/intro.mp3 HTTP/1.1
325	5.232315	127.0.0.1	127.0.0.1	HTTP	195	GET http://weixin.qq.com/ HTTP/1.1
395	7.013917	127.0.0.1	127.0.0.1	HTTP	316	HTTP/1.1 304 NOT MODIFIED
399	7.014031	127.0.0.1	127.0.0.1	HTTP	6686	HTTP/1.1 200 OK (PNG)
463	7.019946	127.0.0.1	127.0.0.1	HTTP	92	HTTP/1.1 200 OK (PNG)
515	7.032135	127.0.0.1	127.0.0.1	HTTP	734	HTTP/1.1 200 OK (PNG)
539	7.033565	127.0.0.1	127.0.0.1	HTTP	7830	HTTP/1.1 200 OK (JPEG JFIF image)
563	7.035316	127.0.0.1	127.0.0.1	MPEG-1	6186	Audio Layer 3, 192 kb/s, 48 kHz
582	8.834702	127.0.0.1	127.0.0.1	HTTP	200	GET /ECAgent/?op=__check_alive__&token=&guid=guid-sp&callback=e HTTP/1.1
584	8.834914	127.0.0.1	127.0.0.1	HTTP	191	HTTP/1.1 200 OK (text/javascript)
606	10.240869	127.0.0.1	127.0.0.1	HTTP	46	HTTP/1.1 502 Bad Gateway
621	13.852118	127.0.0.1	127.0.0.1	HTTP	200	GET /ECAgent/?op=__check_alive__&token=&guid=guid-sp&callback=e HTTP/1.1
623	13.852427	127.0.0.1	127.0.0.1	HTTP	191	HTTP/1.1 200 OK (text/javascript)
652	19.824504	127.0.0.1	127.0.0.1	HTTP	200	GET /ECAgent/?op=__check_alive__&token=&guid=guid-sp&callback=e HTTP/1.1
654	19.824723	127.0.0.1	127.0.0.1	HTTP	191	HTTP/1.1 200 OK (text/javascript)
672	22.119215	127.0.0.1	127.0.0.1	HTTP	170	CONNECT 61.151.183.16:80 HTTP/1.1
674	22.119326	127.0.0.1	127.0.0.1	HTTP	83	HTTP/1.1 200 Connection established
676	22.119475	127.0.0.1	127.0.0.1	TCP	175	10276 → 7890 [PSH, ACK] Seq=127 Ack=40 Win=2161152 Len=131
682	22.158302	127.0.0.1	127.0.0.1	HTTP	195	GET http://weixin.qq.com/ HTTP/1.1
691	24.838881	127.0.0.1	127.0.0.1	HTTP	200	GET /ECAgent/?op=__check_alive__&token=&guid=guid-sp&callback=e HTTP/1.1
693	24.839073	127.0.0.1	127.0.0.1	HTTP	191	HTTP/1.1 200 OK (text/javascript)
710	27.169256	127.0.0.1	127.0.0.1	HTTP	46	HTTP/1.1 502 Bad Gateway

由于除了访问目标页面，还有其他数据包在产生，为了方便观察，接下来过滤出既使用 TCP 80端口 传输，同时使用 HTTP协议 的数据包：

```
tcp.port == 80 && http
```

tcp.port == 80 && http

No.	Time	Source	Destination	Protocol	Lengtl	Info
62	5.084156	127.0.0.1	127.0.0.1	HTTP	532	GET / HTTP/1.1
70	5.094465	127.0.0.1	127.0.0.1	HTTP	5367	HTTP/1.1 200 OK (text/html)
150	5.141273	127.0.0.1	127.0.0.1	HTTP	534	GET /static/logo.png HTTP/1.1
156	5.141429	127.0.0.1	127.0.0.1	HTTP	534	GET /static/ENFJ.png HTTP/1.1
186	5.142351	127.0.0.1	127.0.0.1	HTTP	534	GET /static/ESFP.png HTTP/1.1
224	5.144798	127.0.0.1	127.0.0.1	HTTP	632	GET /static/ENFP.png HTTP/1.1
258	5.145428	127.0.0.1	127.0.0.1	HTTP	540	GET /static/background.png HTTP/1.1
316	5.211435	127.0.0.1	127.0.0.1	HTTP	660	GET /static/intro.mp3 HTTP/1.1
395	7.013917	127.0.0.1	127.0.0.1	HTTP	316	HTTP/1.1 304 NOT MODIFIED
399	7.014031	127.0.0.1	127.0.0.1	HTTP	6686	HTTP/1.1 200 OK (PNG)
463	7.019946	127.0.0.1	127.0.0.1	HTTP	92	HTTP/1.1 200 OK (PNG)
515	7.032135	127.0.0.1	127.0.0.1	HTTP	734	HTTP/1.1 200 OK (PNG)
539	7.033565	127.0.0.1	127.0.0.1	HTTP	7830	HTTP/1.1 200 OK (JPEG JFIF image)
563	7.035316	127.0.0.1	127.0.0.1	MPEG-1	6186	Audio Layer 3, 192 kb/s, 48 kHz

数据包分析

1. **初始 HTML 页面请求：**

- **数据包 62：** `GET / HTTP/1.1` 请求，客户端请求服务器的根页面（首页）。

Source	Destination	Protocol	Lengtl	Info
127.0.0.1	127.0.0.1	HTTP	532	GET / HTTP/1.1

- **数据包 70：**服务器对 HTML 页面请求的响应，返回 `200 OK` 状态，内容类型为 `text/html`。意味着首页的 HTML 内容已成功返回给客户端。

127.0.0.1	127.0.0.1	HTTP	5367	HTTP/1.1 200 OK (text/html)

2. **静态资源请求：**

- **数据包 150 - 316：**这些 `GET` 请求是客户端在加载 HTML 页面后，为渲染页面而请求的静态资源，包括：

150	5.141273	127.0.0.1	127.0.0.1	HTTP	534 GET /static/logo.png HTTP/1.1
156	5.141429	127.0.0.1	127.0.0.1	HTTP	534 GET /static/ENFJ.png HTTP/1.1
186	5.142351	127.0.0.1	127.0.0.1	HTTP	534 GET /static/ESFP.png HTTP/1.1
224	5.144798	127.0.0.1	127.0.0.1	HTTP	632 GET /static/ENFP.png HTTP/1.1
258	5.145428	127.0.0.1	127.0.0.1	HTTP	540 GET /static/background.png HTTP/1.1
316	5.211435	127.0.0.1	127.0.0.1	HTTP	660 GET /static/intro.mp3 HTTP/1.1

- 这些资源分别对应页面中的图片、音频等文件。每个请求后，服务器返回 `200 OK` 状态，确认资源已成功发送给客户端。

3. **音频文件请求：**

- **数据包 316：**显示了对 `/static/intro.mp3` 音频文件的请求，服务器返回状态码 `200 OK`，并使用 `MPEG-1 Audio Layer 3` 格式，表明音频文件已开始加载。
- **数据包 563：**这里的响应内容标明音频文件的比特率和采样率，显示为 `192 kb/s, 48 kHz`，符合常见的 MP3 音频格式。

563	7.035316	127.0.0.1	127.0.0.1	MPEG-1	6186 Audio Layer 3, 192 kb/s, 48 kHz

4. **304 Not Modified 响应：**

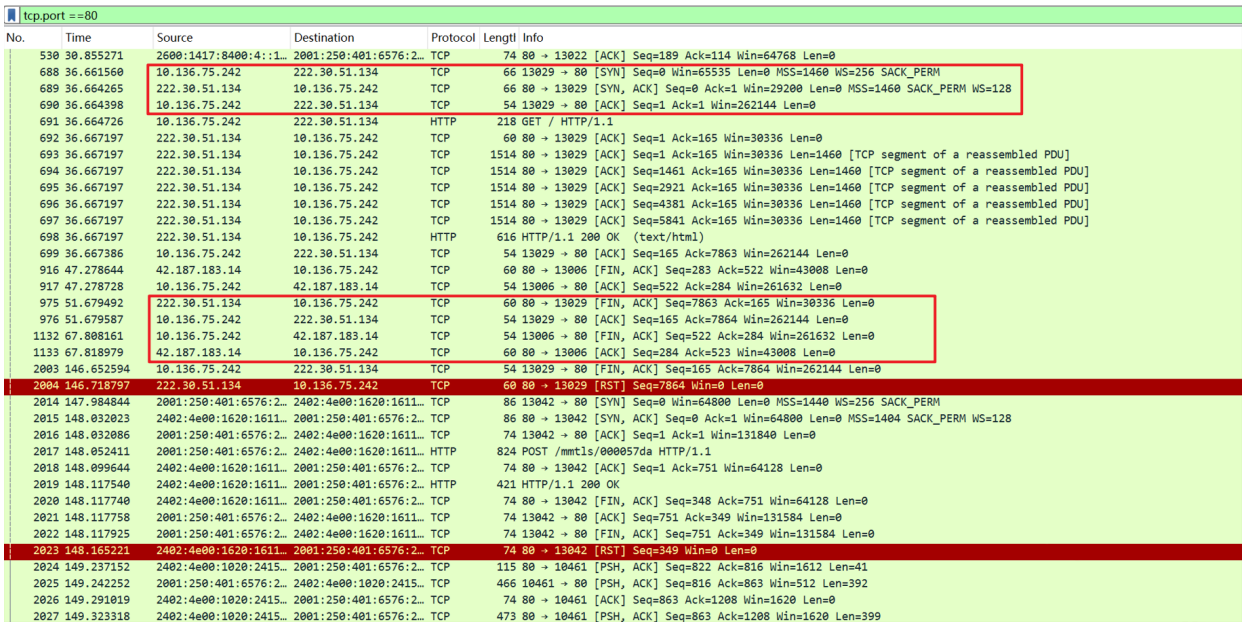
- **数据包 395 及后续：**这些请求是对已经加载过的资源的重复请求，如图片、音频等。
- 服务器返回 `304 Not Modified` 状态，表示这些资源未更改，客户端可以直接从缓存中加载，而不需要重新下载。这种方式提高了页面加载效率，减少了不必要的网络传输。

```
TCP payload (488 bytes)
Hypertext Transfer Protocol
  GET / HTTP/1.1\r\n
  Host: 127.0.0.1\r\n
  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:132.0) Gecko/20100101 Firefox/132.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
  Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2\r\n
  Accept-Encoding: gzip, deflate, br, zstd\r\n
  Connection: keep-alive\r\n
  Upgrade-Insecure-Requests: 1\r\n
  Sec-Fetch-Dest: document\r\n
  Sec-Fetch-Mode: navigate\r\n
  Sec-Fetch-Site: none\r\n
```

前两行的部分的请求方法字段给出了请求类型，URI 给出请求的资源位置（127.0.0.1/）。HTTP 中的请求类型为 GET，最后 HTTP 协议版本给出 HTTP 的版本号为 1.1。后面的部分主要是用于说明请求源、连接类型、以及一些 Cookie 信息等。

4.3.2 WLAN 接口

下图为在WLAN下进行抓包（WLAN下会有更复杂的数据包信息）：



tcp.port ==80

No.	Time	Source	Destination	Protocol	Lengtl	Info
530	30.855271	2600:1417:8400:4::1…	2001:250:401:6576:2…	TCP	74	80 → 13022 [ACK] Seq=189 Ack=114 Win=64768 Len=0
688	36.661560	10.136.75.242	222.30.51.134	TCP	66	13029 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
689	36.664265	222.30.51.134	10.136.75.242	TCP	66	80 → 13029 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM WS=128
690	36.664398	10.136.75.242	222.30.51.134	TCP	54	13029 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
691	36.664726	10.136.75.242	222.30.51.134	HTTP	218	GET / HTTP/1.1
692	36.667197	222.30.51.134	10.136.75.242	TCP	60	80 → 13029 [ACK] Seq=1 Ack=165 Win=30336 Len=0
693	36.667197	222.30.51.134	10.136.75.242	TCP	1514	80 → 13029 [ACK] Seq=1 Ack=165 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
694	36.667197	222.30.51.134	10.136.75.242	TCP	1514	80 → 13029 [ACK] Seq=1461 Ack=165 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
695	36.667197	222.30.51.134	10.136.75.242	TCP	1514	80 → 13029 [ACK] Seq=2921 Ack=165 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
696	36.667197	222.30.51.134	10.136.75.242	TCP	1514	80 → 13029 [ACK] Seq=4381 Ack=165 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
697	36.667197	222.30.51.134	10.136.75.242	TCP	1514	80 → 13029 [ACK] Seq=5841 Ack=165 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
698	36.667197	222.30.51.134	10.136.75.242	HTTP	616	HTTP/1.1 200 OK (text/html)
699	36.667386	10.136.75.242	222.30.51.134	TCP	54	13029 → 80 [ACK] Seq=165 Ack=7863 Win=262144 Len=0
916	47.278644	42.187.183.14	10.136.75.242	TCP	60	80 → 13006 [FIN, ACK] Seq=283 Ack=522 Win=43008 Len=0
917	47.278728	10.136.75.242	42.187.183.14	TCP	54	13006 → 80 [ACK] Seq=522 Ack=284 Win=261632 Len=0
975	51.679492	222.30.51.134	10.136.75.242	TCP	60	80 → 13029 [FIN, ACK] Seq=7863 Ack=165 Win=30336 Len=0
976	51.679587	10.136.75.242	222.30.51.134	TCP	54	13029 → 80 [ACK] Seq=165 Ack=7864 Win=262144 Len=0
1132	67.808161	10.136.75.242	42.187.183.14	TCP	54	13006 → 80 [FIN, ACK] Seq=522 Ack=284 Win=261632 Len=0
1133	67.818979	42.187.183.14	10.136.75.242	TCP	60	80 → 13006 [ACK] Seq=284 Ack=523 Win=43008 Len=0
2003	146.652594	10.136.75.242	222.30.51.134	TCP	54	13029 → 80 [FIN, ACK] Seq=165 Ack=7864 Win=262144 Len=0
2004	146.718797	222.30.51.134	10.136.75.242	TCP	60	80 → 13029 [RST] Seq=7864 Win=0 Len=0
2014	147.984844	2001:250:401:6576:2…	2402:4e00:1620:1611…	TCP	86	13042 → 80 [SYN] Seq=0 Win=64800 Len=0 MSS=1440 WS=256 SACK_PERM
2015	148.032023	2402:4e00:1620:1611…	2001:250:401:6576:2…	TCP	86	80 → 13042 [SYN, ACK] Seq=0 Ack=1 Win=64800 Len=0 MSS=1404 SACK_PERM WS=128
2016	148.032086	2001:250:401:6576:2…	2402:4e00:1620:1611…	TCP	74	13042 → 80 [ACK] Seq=1 Ack=1 Win=131840 Len=0
2017	148.052411	2001:250:401:6576:2…	2402:4e00:1620:1611…	HTTP	824	POST /mmtls/000057da HTTP/1.1
2018	148.099644	2402:4e00:1620:1611…	2001:250:401:6576:2…	TCP	74	80 → 13042 [ACK] Seq=1 Ack=751 Win=64128 Len=0
2019	148.117540	2402:4e00:1620:1611…	2001:250:401:6576:2…	HTTP	421	HTTP/1.1 200 OK
2020	148.117740	2402:4e00:1620:1611…	2001:250:401:6576:2…	TCP	74	80 → 13042 [FIN, ACK] Seq=348 Ack=751 Win=64128 Len=0
2021	148.117758	2001:250:401:6576:2…	2402:4e00:1620:1611…	TCP	74	13042 → 80 [ACK] Seq=751 Ack=349 Win=131584 Len=0
2022	148.117925	2001:250:401:6576:2…	2402:4e00:1620:1611…	TCP	74	13042 → 80 [FIN, ACK] Seq=751 Ack=349 Win=131584 Len=0
2023	148.165221	2402:4e00:1620:1611…	2001:250:401:6576:2…	TCP	74	80 → 13042 [RST] Seq=349 Win=0 Len=0
2024	149.237152	2402:4e00:1020:2415…	2001:250:401:6576:2…	TCP	115	80 → 10461 [PSH, ACK] Seq=822 Ack=816 Win=1612 Len=41
2025	149.242252	2001:250:401:6576:2…	2402:4e00:1020:2415…	TCP	466	10461 → 80 [PSH, ACK] Seq=816 Ack=863 Win=512 Len=392
2026	149.291019	2402:4e00:1020:2415…	2001:250:401:6576:2…	TCP	74	80 → 10461 [ACK] Seq=863 Ack=1208 Win=1620 Len=0
2027	149.323318	2402:4e00:1020:2415…	2001:250:401:6576:2…	TCP	473	80 → 10461 [PSH, ACK] Seq=863 Ack=1208 Win=1620 Len=399

1. POST 请求（用于上传用户会话或加密数据，涉及多部分数据的传输）：

- 多个数据包显示了 `POST` 请求，发送到 URL 中包含 `encrypt` 和 `tk` 参数的路径。
- 内容类型为 `application/multipart-formdata`，通常用于传输文件或多部分数据。
- 与身份验证、会话数据或加密的会话令牌相关，可能是应用程序上传或发送加密数据包的操作。

2. GET 请求（静态资源加载或连接测试，以确保与服务器的连接稳定性）：

- `GET` 请求主要用来访问服务器上的静态资源或确认连接的状态，例如 `/connecttest.txt` 文件的访问。
- 数据包中显示的 `/connecttest.txt` 请求返回 `200 OK`，内容类型为 `text/plain`，用于连接测试。
- `/mmtls/00006731` 和 `/mmtls/0000674f` 这样的路径可能是获取特定的静态文件资源。

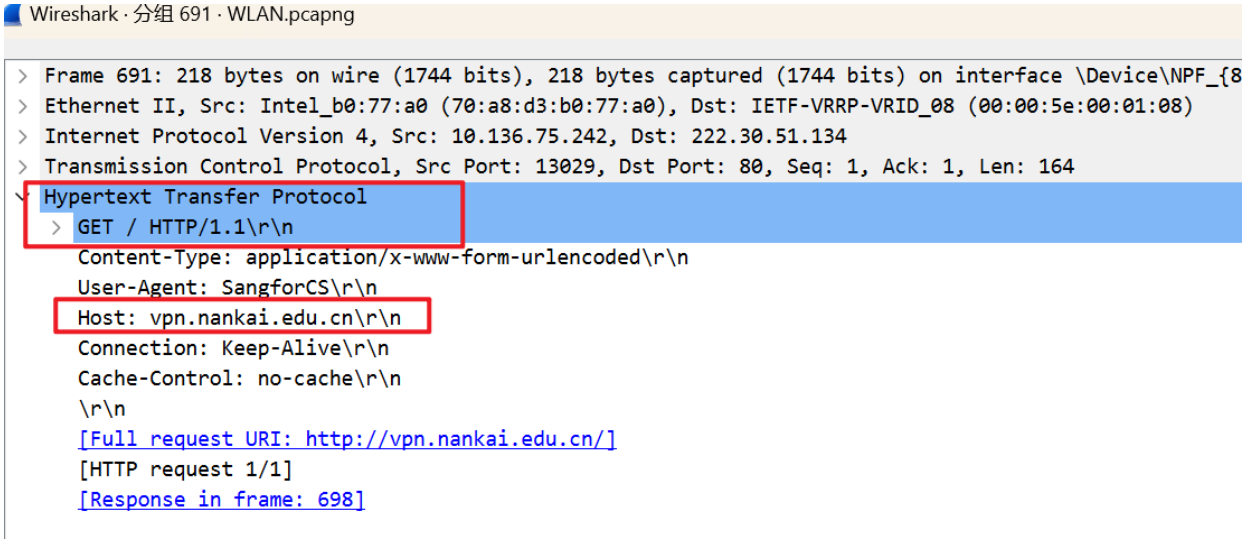
3. 200 OK 响应：

- `200 OK` 表示请求成功，服务器正确返回了客户端所请求的资源。
- 这些 `200 OK` 响应中的 `application/multipart-formdata` 和 `text/plain` 指明了不同的内容类型。

4. OCSP 请求与响应（检查服务器证书的有效性）：

- OCSP 是一种检查证书状态的协议，浏览器或应用程序可能会检查服务器的证书是否有效，从而发起 OCSP 请求并接收响应。

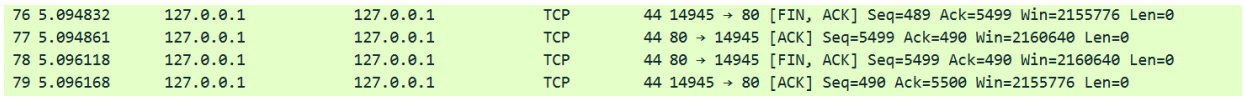
客户端请求 HTML 页面（例如加载首页 `/` 或者 `/index.html`），能够在抓包信息中看到一个 `GET` 请求，通常带有 `text/html` 的 `Content-Type`，表示获取 HTML 文档的请求。



连接了学校的 VPN 网络后，所有网络请求（包括本地的 Flask 请求）都会通过 VPN 出口进行处理。这会导致 `Host` 字段显示 VPN 的地址。

4.4 TCP关闭连接（本地回环）

下图包括了四次挥手的数据包，数据包与实验报告中三次握手来源于同一次的抓包处理（14945<-->80）



寻找 `FIN` 和 `ACK` 包的组合。四次挥手包含了客户端和服务器各自发送的 `FIN` 和对应的 `ACK`，用于关闭连接。

4.4.1 第一次挥手（第一行）

主动关闭方发送一个 FIN 并进入 FIN_WAIT1 状态：

- [FIN, ACK]：源 IP `127.0.0.1` 端口 `80` 向目的 IP `127.0.0.1` 端口 `14945` 发送 `FIN, ACK` 包，表示主动关闭连接的一方请求断开连接。
- Seq=283，Ack=522：这是当前包的序列号和确认号。

4.4.2 第二次挥手（第二行）

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：

- [ACK]：目的 IP `127.0.0.1` 端口 `14945` 回复 `ACK` 包，确认收到来自服务器的 `FIN`。
- Seq=522，Ack=284：确认了对方的 `FIN` 包，表示已经接收并同意关闭连接的一半。

4.4.3 第三次挥手（第三行）

被动关闭方发送一个 FIN 并进入 LAST_ACK 状态

- [FIN, ACK]：客户端 `127.0.0.1` 向服务器发送 `FIN, ACK` 包，表示准备关闭连接的另一半，进入关闭状态。
- Seq=522，Ack=284：客户端发起了自己的 `FIN` 请求。

4.4.4 第四次挥手（第四行）

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，经过一段时间后关闭连接；被动关闭方收到主动关闭方的 ACK 后，关闭连接；

- [ACK]：服务器 `127.0.0.1` 端口 `80` 发送 `ACK` 包，确认收到客户端的 `FIN` 请求，完成连接的关闭。
- Seq=284，Ack=523：确认号表示已确认接收到对方的 `FIN`。

五、实验总结

在本次实验中，我成功搭建了本地 Flask 服务器，并通过 Wireshark 抓包分析了 TCP 三次握手和四次挥手的过程，验证了 HTTP 请求的发送和接收。我观察到不同类型的 HTTP 请求（如 `GET` 和 `POST`）的流量，以及页面加载过程中静态资源的缓存机制（304 Not Modified 响应）。

遇到的问题：

1. **无法区分不同连接的四次挥手**：最初抓包中混入了其他连接的流量，导致识别特定页面的四次挥手困难。通过端口和 IP 地址过滤后，成功锁定特定会话。
2. **Host 字段异常**：由于 VPN 或代理干扰，Host 字段显示为外部网络地址，通过断开 VPN 并选择回环接口解决。
3. **静态资源未加载的疑问**：由于缓存机制，部分资源未重新加载，调整过滤条件并清除缓存后，成功捕获所有请求。

本次实验增强了对 TCP 连接管理和 HTTP 协议的理解，为网络协议分析提供了宝贵的实践经验。