南开大学
Nankai University

南　　开　　大　　学

计　算　机　学　院

计算机网络实验报告

实验 3：配置 Web 服务器，捕获 HTTP 报文并分析

黄尚扬

年级：2023 级

专业：计算机科学与技术

指导教师：徐敬东 张建忠

2026 年 1 月 2 日

摘要

关键字：计算机网络 HTTP 协议 Web 服务器 Wireshark 抓包

本报告是计算机网络课程中的第三次实验。

在这一部分，笔者将围绕 Web 服务器的搭建与 HTTP 交互过程展开实验与分析。通过自行构建简单的 Web 页面，并利用浏览器访问该页面，结合 Wireshark 对通信过程进行抓包与解析，笔者将对 HTTP 报文在各协议层次上的封装方式以及完整的交互流程进行详细说明，从而加深对计算机网络协议栈及实际通信过程的理解。

目录

一、 实验目的

(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（包含专业、学号、姓名）和 6 幅图像。

(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。

(3) 分析两个报文的封装层次，包括数据链路层、互连层、传输层、应用层。

(4) 对整个 HTTP 交互过程进行详细说明，使用 Wireshark 过滤器使其只显示 HTTP 协议。

(5) 提交 HTML 文档、Wireshark 捕获文件和实验报告

注：页面不要太复杂，包含所要求的基本信息即可。使用 HTTP，不使用 HTTPS。

二、 项目结构

实验的 git 链接为：Hsy23333/NKU_ComputerNetworks

本次实验存放在 l3 文件夹中，包含一个 html 文档和 6 张 jpg 图片。搭建好服务器环境后按照路径访问即可。

三、 实验原理

(一) TCP 协议

TCP 协议概述 传输控制协议（Transmission Control Protocol，TCP）是工作在传输层的一种面向连接的、可靠的通信协议，主要用于为上层应用提供稳定、有序且无差错的数据传输服务。与无连接的传输协议相比，TCP 通过引入连接管理、差错控制、流量控制以及拥塞控制等机制，有效保证了数据在复杂网络环境下的可靠传输。

TCP 的连接建立与释放 TCP 采用三次握手（Three-Way Handshake）机制来建立连接。客户端首先向服务器发送 SYN 报文，请求建立连接；服务器收到请求后返回 SYN+ACK 报文进行确认；客户端再发送 ACK 报文作为最终确认，至此双方完成连接建立并进入数据传输阶段。该过程能够有效避免历史连接请求对当前连接造成干扰。

在通信结束时，TCP 通过四次挥手（Four-Way Handshake）机制释放连接。通信双方分别独立地关闭发送方向，从而确保剩余数据能够被完整接收，避免数据丢失。

TCP 的可靠传输机制 TCP 通过多种机制实现可靠传输。首先，TCP 为每个字节的数据分配序列号，并通过确认号（ACK）机制保证数据按序到达；其次，发送方在超时未收到确认时会触发重传机制，从而应对网络中的丢包情况；此外，TCP 还引入校验和字段，用于检测数据在传输过程中是否发生比特错误。

TCP 的流量控制 为了防止发送方发送速率过快而导致接收方缓冲区溢出，TCP 采用基于滑动窗口的流量控制机制。接收方通过通告窗口大小（Receive Window, rwnd）告知发送方其当前可接收的数据量，发送方据此动态调整发送速率，从而实现端到端的流量匹配。

TCP 的拥塞控制 TCP 通过拥塞控制机制来避免网络中出现严重拥塞。典型的拥塞控制算法包括慢启动、拥塞避免、快速重传和快速恢复。发送方根据网络反馈动态调整拥塞窗口（Congestion Window, cwnd），在保证网络稳定性的前提下尽可能提高链路利用率。

TCP 在实验中的应用 在本实验中，TCP 协议为客户端与服务器之间的通信提供了可靠的传输基础。基于 TCP 的套接字通信能够确保应用层数据完整、有序地到达对端，为后续的应用层协议分析与实验验证提供了稳定的运行环境。

(二) HTTP 协议

HTTP 协议概述 超文本传输协议（HyperText Transfer Protocol，HTTP）是一种工作在应用层的无状态协议，主要用于在 Web 客户端与服务器之间传输超文本及相关资源。HTTP 基于请求—响应（Request–Response）模型，客户端通过发送请求报文向服务器请求资源，服务器在接收到请求后返回相应的响应报文。HTTP 通常运行在 TCP 协议之上，依赖 TCP 提供的可靠传输服务。[1]

HTTP 的工作模式 HTTP 采用典型的客户端—服务器通信模式。通信过程中，浏览器作为客户端主动向 Web 服务器发起请求，请求内容包括请求方法、请求资源路径以及相关的首部信息；服务器在解析请求后，根据请求内容返回相应的状态码、响应首部以及实体内容。该模式使得 Web 系统结构清晰，易于扩展和维护。

HTTP 请求与响应报文结构 HTTP 报文由起始行、首部行以及可选的消息体三部分组成。请求报文的起始行为请求行，包含请求方法、请求 URI 以及协议版本；响应报文的起始行为状态行，包含协议版本、状态码及其文字描述。首部字段用于传递附加控制信息，如主机地址、内容类型和连接方式等，而消息体则用于承载实际传输的数据内容。

HTTP 的主要请求方法 HTTP 定义了多种请求方法，其中最常用的是 GET 和 POST 方法。GET 方法主要用于请求服务器资源，其请求参数通常附加在 URL 中；POST 方法则用于向服务器提交数据，请求参数包含在消息体中。在本实验中，浏览器访问 Web 页面时主要使用 GET 方法向服务器请求 HTML 文档及相关图像资源。

HTTP 的无状态特性 HTTP 是一种无状态协议，即服务器不会主动保存客户端的会话状态。每一次请求都被视为相互独立的事务，这种设计简化了服务器实现并提高了系统的可扩展性。然而，在实际应用中，为了维持用户会话状态，通常会结合 Cookie 或 Session 等机制进行状态管理。

HTTP 连接管理 在 HTTP/1.0 中，默认采用短连接方式，即每完成一次请求—响应过程后便断开 TCP 连接；而在 HTTP/1.1 中，引入了持久连接机制，允许在同一 TCP 连接上连续传输多个 HTTP 报文，从而减少连接建立与释放带来的额外开销，提高通信效率。

HTTP 在实验中的应用 在本实验中，客户端浏览器通过 HTTP 协议向本地 Web 服务器请求 HTML 页面及图像资源。借助 Wireshark 抓包工具，可以观察到 HTTP 报文在 TCP 连接上的传输过程，并进一步分析请求与响应报文的具体内容及其在协议栈中的封装关系，从而加深对 HTTP 协议工作机制的理解。

(三) 交互环境

实验环境与服务器搭建 本实验在本地主机环境下进行，采用 Python 语言搭建简易 Web 服务器，并通过 `localhost:8000` 端口对外提供 HTTP 服务。其中，`localhost` 表示本机回环地址，

用于实现客户端与服务器在同一主机上的通信；端口号 8000 作为常用的非特权端口，避免与系统中已有服务产生冲突。

服务器端基于 Python 内置的 HTTP 服务模块实现，能够对浏览器发送的 HTTP 请求进行解析，并向客户端返回相应的 HTML 文档及静态资源。该方式无需额外安装复杂的服务器软件，搭建过程简洁，便于实验环境的快速部署与调试，适合作为教学与实验验证用途。

客户端使用常见 Web 浏览器访问 `http://localhost:8000`，浏览器通过 HTTP 协议向本地服务器发起请求，请求内容包括 HTML 页面以及页面中引用的图像资源。在该过程中，HTTP 报文通过 TCP 连接在客户端与服务器之间传输，为后续的协议分析提供了完整且可控的实验场景。

由于客户端与服务器均运行在同一主机上，实验过程中可以有效排除外部网络环境对通信过程的影响，从而更加清晰地观察 HTTP 报文的产生、传输与封装过程，便于使用 Wireshark 对通信数据进行捕获与分析。

Wireshark 在本实验中的作用 Wireshark 是一种常用的网络协议分析工具，能够对网络接口上的数据包进行实时捕获与解析。在本实验中，Wireshark 主要用于捕获浏览器与本地 Web 服务器之间的通信数据，从而对 HTTP 协议及其底层传输过程进行直观分析。

通过在本地主机上启用 Wireshark 抓包，并结合合适的显示过滤规则，可以准确获取客户端访问 `localhost:8080` 时所产生的 HTTP 报文。Wireshark 能够对捕获的数据包进行分层解析，清晰展示数据在数据链路层、网络层、传输层以及应用层中的封装结构，为分析协议栈各层的功能提供了直接依据。

在实验过程中，Wireshark 不仅用于观察 HTTP 请求与响应报文的具体内容，还可用于分析 TCP 连接的建立与释放过程，例如三次握手与四次挥手等关键通信阶段。通过对报文时间顺序和字段信息的观察，可以更直观地理解 TCP 与 HTTP 协议之间的协同工作机制。

此外，Wireshark 支持对捕获结果进行保存和离线分析，便于实验数据的整理与复现。通过对抓包结果的分析，笔者能够将抽象的协议原理与实际网络通信过程相结合，从而加深对计算机网络相关协议运行机制的理解。

四、 实验步骤

(一) 网页编写

首先我们需要编写一个 html 网页。依照实验要求，其内容如下:

main.html

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
    <meta charset="UTF-8">
    <title>计算机网络lab3 页面展示</title>
</head>

<body>
    <h1>计算机网络lab3 页面展示</h1>

    <p><strong>姓名: </strong>黄尚扬</p>
    <p><strong>学号: </strong>2313911</p>
```

```
    <p><strong>专业：</strong>计算机科学与技术</p>

    <hr>

    <h2>个人相关图片展示</h2>

    <img src="img1.jpg" width="200">
    <img src="img2.jpg" width="200">
    <img src="img3.jpg" width="200">
    <br><br>
    <img src="img4.jpg" width="200">
    <img src="img5.jpg" width="200">
    <img src="img6.jpg" width="200">

</body>
</html>
```

该页面采用标准的 HTML5 文档结构进行编写，用于在浏览器中展示实验相关的基本信息与图像内容。文档开头通过 `<!DOCTYPE html>` 声明当前文档类型为 HTML5，以确保浏览器按照统一的标准解析页面内容。`<html>` 标签作为整个 HTML 文档的根元素，其中 `lang="zh-CN"` 属性用于声明页面主要语言为简体中文，有助于浏览器及辅助工具正确进行语言识别与编码处理。

页面的头部区域由 `<head>` 标签定义，其中 `<meta charset="UTF-8">` 用于指定文档字符编码为 UTF-8，从而保证中文字符能够被正确显示；`<title>` 标签定义了页面标题，该标题将显示在浏览器标签页上，用于标识当前 Web 页面。页面主体内容位于 `<body>` 标签内。通过 `<h1>` 标签设置页面的一级标题，用于突出页面主题；随后使用多个 `<p>` 段落标签配合 `<strong>` 标签，对姓名、学号和专业等关键信息进行展示，其中 `<strong>` 标签用于强调文本内容，使其在页面中更加醒目。`<hr>` 标签用于在页面中插入一条水平分隔线，用以区分基本信息区域与后续内容展示区域，从而提升页面结构的清晰度。

在图像展示部分，页面通过 `<h2>` 标签设置二级标题，对图片区域进行说明。随后使用多个 `<img>` 标签加载本地图片资源，其中 `src` 属性指定图像文件路径，`width` 属性用于统一控制图片显示宽度。通过插入 `<br>` 换行标签，将图片分为上下两行排列。

可以知道，其包含了简单文本信息（包含专业、学号、姓名）和 6 幅图像。**虽然结构简单但完全符合作业要求**。由于这部分不是重点，笔者不再做无用的雕花工作。

下面则是该 html 被浏览器访问时的样子，可以看出显示正常，我们能够进入下一部分：

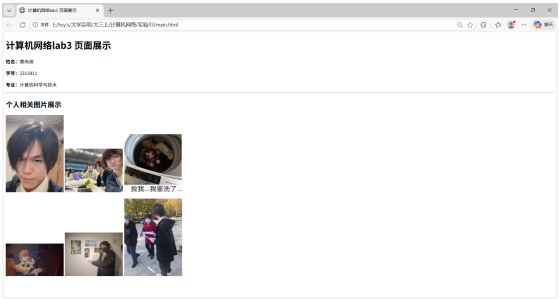

图 1: 网页展示

(二) Web 服务器环境建立

依照前文的解释，我们使用下列简单的命令，在 localhost:8000 上建立一个本地服务器：

powershell

```
python -m http.server 8000
```

正常执行时，会出现 Serving HTTP on :: port 8000 (http://[::]:8000/) ... 字样，说明服务器正常启动。值得一提的是，笔者在实验根目录下启动服务器，于是需要按照路径寻址进行访问。

在服务器启动后，浏览器可通过访问 `http://localhost:8000` 与本地 Web 服务器建立连接。由于服务器是在实验根目录下启动的，Python 内置的 HTTP 服务会将该目录作为站点根目录，并自动生成对应的目录索引页面。浏览器首先接收到的即为该目录下文件结构的展示，如图 2 所示。

当用户在目录页面中点击具体的 HTML 文件时，浏览器会向服务器发送针对该文件的 HTTP GET 请求，服务器根据请求路径返回相应的页面内容；若页面中包含对图像等静态资源的引用，浏览器将继续发起多个 GET 请求以获取相关资源。通过这种方式，可以完整地观察一次页面访问过程中 HTTP 请求与响应的产生与传输过程，为后续的协议抓包与分析提供实验基础。

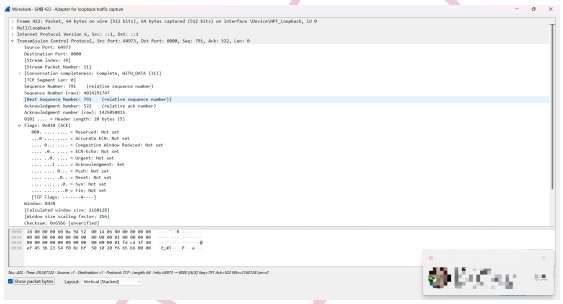

图 2: localhost:8000 上的目录

(三) Wireshark 捕获预备

在完成 Web 服务器的启动后，为捕获浏览器与本地服务器之间的通信数据，需要启动 Wireshark 并对相应网络接口进行监听。由于本实验中客户端与服务器均运行在同一主机上，通信过程通过本地回环接口完成，因此在 Wireshark 中选择回环接口进行数据包捕获。

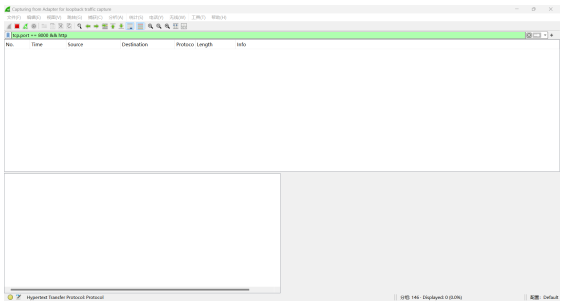

图 3: 初始状态

开始抓包后，保持 Wireshark 处于监听状态，并在浏览器中访问对应的 HTML 页面。此时，浏览器会向本地服务器发起 HTTP 请求，请求页面文件及其所包含的图像等静态资源，相关的

TCP 与 HTTP 报文将被 Wireshark 实时捕获并记录。

在页面成功加载并显示完成后，可停止 Wireshark 抓包。通过对捕获结果进行筛选与分析，可以清晰观察到一次完整页面访问过程中 TCP 连接建立、HTTP 请求与响应以及连接释放等通信过程，为后续的协议层次分析与交互过程说明提供数据依据。

特别地，我们有以下两个过滤器指令：

```
1 http
2 tcp.port == 8000 && ipv6.addr == ::1
```

`http` 是 Wireshark 提供的应用层显示过滤器，用于筛选并显示所有被识别为 HTTP 协议的数据包。在启用该过滤器后，捕获结果中仅保留 HTTP 请求与响应相关的报文，其余如 TCP 握手报文、确认报文等将被隐藏。通过该过滤器，可以更加直观地观察浏览器与 Web 服务器之间的 HTTP 交互过程，包括请求方法、请求路径、响应状态码以及传输的资源类型等信息，便于对 HTTP 协议本身进行分析。

`tcp.port == 8000 && ipv6.addr == ::1` 是一个组合显示过滤器，用于限定捕获结果只显示与本实验相关的 TCP 通信数据。其中，`tcp.port == 8000` 表示仅筛选目标端口或源端口为 8000 的 TCP 报文，对应本地 Web 服务器监听的端口；`ipv6.addr == ::1` 用于限定通信地址为 IPv6 回环地址，即本地主机。二者结合使用，可以有效排除无关网络流量，仅保留浏览器与本地服务器之间的 TCP 通信数据，为分析 TCP 连接建立、数据传输及连接释放过程提供清晰的抓包结果。

由此，我们就能进入正式实验。

(四) 正式交互实验

保持 wireshark 监听，在 localhost:8000 下，我们由 l3 目录进入 main.html。此时网页正常加载，监听窗口中随之出现了许多记录。可以认为实验进展顺利。

关于记录的分析等部分，笔者会在下面详细解释。

五、 交互过程的捕获分析

在本模块，笔者会按照网页交互的逻辑顺序进行梳理分析。**作业要求 (2) 和 (4) 中的内容在本模块都会涉及。**

(一) TCP 三次握手

笔者键入上个模块中提到的第二行过滤器，得到了以下结果。我们可以定位到三次握手建立连接的部分：

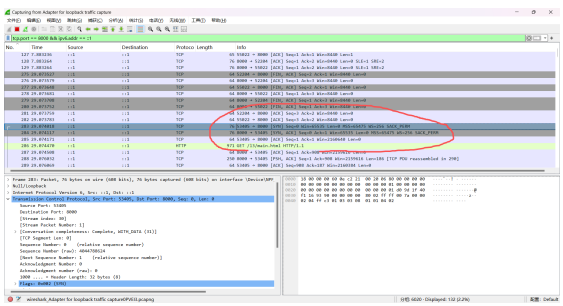

图 4: 三次握手的捕获

我们接下来详细分析其内容。

1. 第一次握手

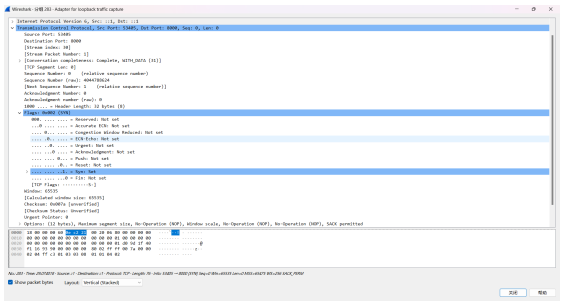

图 5: 第一次握手

第一次握手由客户端发起，用于向服务器请求建立 TCP 连接。该报文的源端口为 53405，目的端口为 8000，分别对应客户端临时端口与服务器 HTTP 服务端口。客户端随机生成初始序列号 4044788624，该值为绝对序列号，在 Wireshark 中对应的相对序列号为 0。此时 ACK 标志位未置位，而 SYN 标志位置位，表明该报文为连接建立请求。发送该报文后，客户端进入 SYN-SENT 状态，等待服务器的响应。

2. 第二次握手

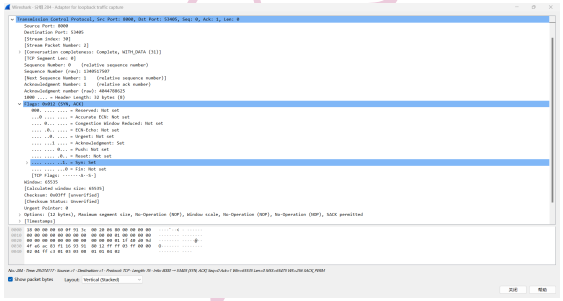

图 6: 第二次握手

第二次握手由服务器发送给客户端，用于确认客户端的连接请求并同步服务器自身的初始状态。该报文的源端口为 8000，目的端口为 53405。服务器随机生成的初始序列号为 1340517507（绝对序列号），其相对序列号为 0。报文中同时设置了 SYN 与 ACK 标志位，其中确认号（ACK 字段）的绝对值为客户端第一次握手序列号加 1，即

$$4044788624 + 1 = 4044788625$$

其相对确认号为 1。这表明服务器已成功接收到客户端的连接请求，并同意建立连接。此时，服务器进入 SYN-RCVD 状态。

3. 第三次握手

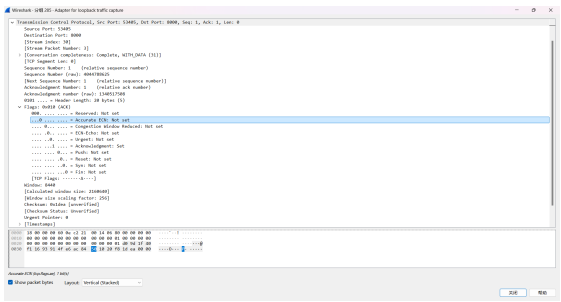

图 7: 第三次握手

第三次握手再次由客户端发送至服务器，用于对服务器的确认报文进行回应。该报文的源端口与目的端口仍分别为 53405 和 8000。客户端在该报文中设置 ACK 标志位，其序列号在相对意义上为 1，即在第一次握手基础上加 1；确认号（ACK 字段）的绝对值为服务器第二次握手序列号加 1，即

$$1340517507 + 1 = 1340517508$$

其相对确认号为 1。该报文发送完成后，客户端进入 ESTABLISHED 状态；当服务器接收到该报文后，也将进入 ESTABLISHED 状态，至此 TCP 连接成功建立。

通过上述三次握手过程可以看出，TCP 协议通过序列号与确认号机制，实现了通信双方初始状态的同步与连接可靠性的保障，为后续 HTTP 数据的可靠传输奠定了基础。

(二) HTTP 加载交互

将过滤器设置为 http，得到如下结果，可以获知全部都为 HTTP 协议：

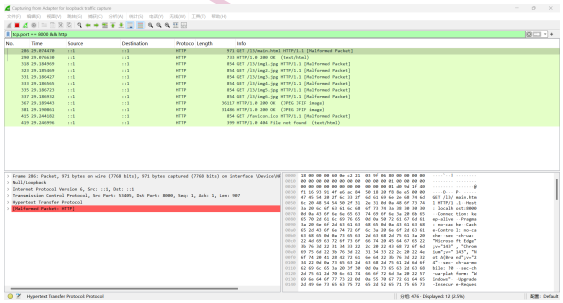

图 8: http 过滤捕获结果

通信协议默认为 HTTP1.1。

在本次 HTML 页面访问实验中，浏览器与 Web 服务器之间的交互主要通过 HTTP GET 请求完成。当用户在浏览器中访问指定页面地址时，浏览器首先向服务器发送针对 HTML 文档的 GET 请求，请求页面主体内容。服务器在接收到请求后，返回包含 HTML 源码的响应报文，浏览器对该内容进行解析与渲染。

在解析 HTML 文档的过程中，浏览器会根据页面中引用的外部资源继续向服务器发送多个 GET 请求，请求对应的静态资源。服务器分别返回这些资源的数据内容，浏览器在接收完成后将其加载并显示在页面中。由此可见，一次完整的页面访问过程通常包含一次针对 HTML 文件的 GET 请求以及多次针对页面资源的 GET 请求。

通过 Wireshark 抓包可以观察到，这里依次出现了 6 张 jpg 图片的 GET 请求，说明 http 的交互不论在内容完整性上还是有序性上都正常进行。

至于最后一个 GET 及其返回的 404，笔者在查阅信息后得知，此为获取页面图标的请求；而由于笔者没有进行对应设计（当然，作业内也没有要求），所以找不到资源也是非常正常的。

下面是关于请求和响应具体的分析。

图 9: http 请求内容

上面是获取 main.html 的请求。

HTTP 请求报文由请求行、请求头以及可选的请求体组成，其中请求头用于向服务器说明客户端的基本信息及其对响应内容的要求。本实验中请求头各字段含义如下。

请求头（Request Header）

- **Host**：表示请求的目标服务器地址，用于指明客户端希望访问的主机。在本实验中，该字段对应用户本机的 IP 地址，即本地 Web 服务器的地址；
- **User-Agent**：用于标识客户端的软件环境，例如浏览器类型、版本以及操作系统信息。本实验中该字段显示客户端使用的是 Edge 浏览器，运行于 Windows 平台；
- **Accept**：指定客户端能够接收的响应内容类型，如 `text/html`、`application/xhtml+xml` 和 `application/xml`，用于指导服务器返回合适格式的资源；
- **Accept-Language**：用于声明客户端期望接收的自然语言类型，例如 `zh-CN`，有助于服务器进行语言内容的选择；
- **Accept-Encoding**：表示客户端能够接受的内容编码方式，例如 `gzip`，服务器可据此对响应内容进行压缩以提高传输效率；
- **Connection**：用于控制连接的管理方式。`keep-alive` 表示采用持久连接，在同一 TCP 连接上可以连续传输多个 HTTP 请求与响应，从而减少连接建立与释放的开销。

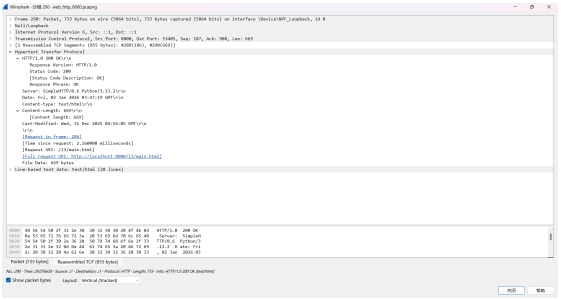

图 10: html 响应内容

我们在上图取了其中一个响应为例，可以窥见其响应格式。

HTTP 响应报文主要由响应行和响应头两部分组成，其各字段含义如下。

响应行（Response Line） 响应行用于说明服务器对客户端请求的处理结果，主要包括以下内容：

- **Response Version**：表示服务器采用的 HTTP 协议版本，本实验中为 `HTTP/1.1`；
- **Status Code**：状态码，用于反映请求的处理结果，常见的包括 `200`（请求成功）、`3xx`（重定向）、`4xx`（客户端错误）以及 `5xx`（服务器错误）；
- **Response Phrase**：状态码对应的文字描述，例如 `OK`，用于对响应结果进行说明。

响应头（Response Header） 响应头用于向客户端传递与响应相关的附加信息，主要字段包括：

- **Date**：表示服务器生成响应的时间，例如 `Thu, 31 Oct 2024 10:27:56 GMT`；
- **Server**：用于标识服务器所使用的软件及其版本信息，本实验中显示为 Apache 服务器及 PHP 运行环境；
- **Content-Type**：表示响应实体的内容类型，例如 `text/html`，用于告知浏览器如何解析返回的数据。

(三) TCP 四次挥手

在 HTTP 数据传输完成后，客户端与服务器通过 TCP 四次挥手机制释放已建立的连接。四次挥手过程保证了通信双方能够在数据传输结束后，分别、可靠地关闭各自的发送方向，从而避免数据丢失。

笔者在捕获信息靠后的位置筛选出了如下几条：

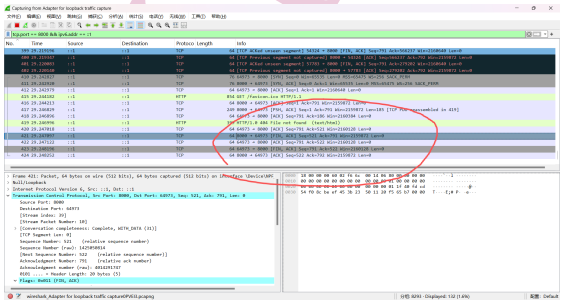

图 11: 四次挥手

与三次握手类似地，我们接下来详细进行分析。

1. 第一次挥手

第一次挥手由客户端发起，用于请求关闭连接。客户端向服务器发送带有 FIN 标志位的 TCP 报文，表示客户端已无数据发送。该报文的 ACK 标志位置位，用于确认此前接收到的数据。发送该报文后，客户端进入 `FIN-WAIT-1` 状态。

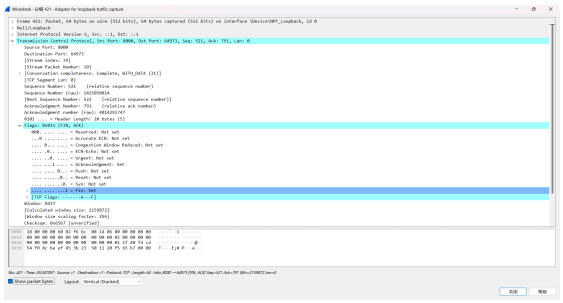

图 12: 第一次挥手

图中可见 8000 与 64973 端口交互报文，而 FIN 被置为 1，和预期相符。

2. 第二次挥手

第二次挥手由服务器发送给客户端，用于确认客户端的连接关闭请求。服务器在接收到 FIN 报文后，返回一个仅设置 ACK 标志位的确认报文，确认号为客户端 FIN 报文序列号加 1。此时，服务器进入 CLOSE-WAIT 状态，而客户端在收到该确认报文后进入 FIN-WAIT-2 状态，等待服务器发送关闭请求。

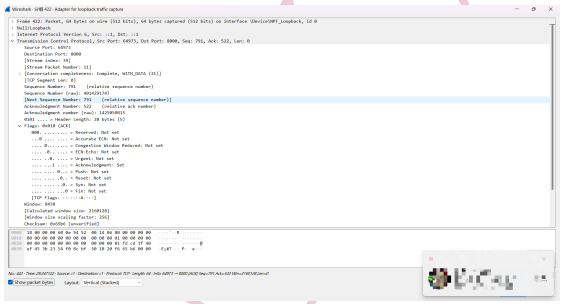

图 13: 第二次挥手

图片中可见 FIN 和 ACK 均被置位，即服务端通知客户端将断开连接，和预期相符。

3. 第三次挥手

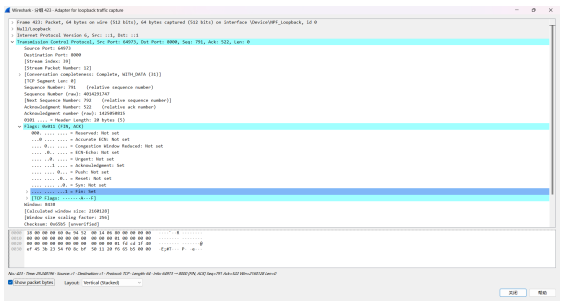

图 14: 第三次挥手

第三次挥手由服务器发起。当服务器完成剩余数据的发送后，会向客户端发送带有 FIN 标志位的 TCP 报文，表示服务器也准备关闭连接。该报文发送后，服务器进入 LAST-ACK 状态，等待客户端的最终确认。如上图所示。

4. 第四次挥手

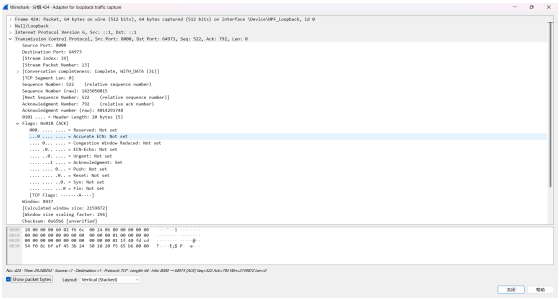

图 15: 第四次挥手

第四次挥手由客户端完成。客户端在接收到服务器的 FIN 报文后，向服务器发送 ACK 确认报文，确认号为服务器 FIN 报文序列号加 1。随后，客户端进入 TIME-WAIT 状态，在等待足够的时间以确保服务器收到确认报文后，客户端才最终进入 CLOSED 状态。服务器在接收到该确认报文后立即进入 CLOSED 状态，至此 TCP 连接被完全释放。

通过对四次挥手过程的分析可以看出，TCP 协议通过分阶段关闭连接的方式，实现了连接释放过程中的可靠性与有序性，确保通信双方的数据传输能够安全结束。至此，本部分实验任务完成。

六、 报文层次分析

典型的 HTTP 通信报文在传输过程中依次经历数据链路层、网络层、传输层以及应用层的封装与解封装过程。

数据链路层 数据链路层负责在相邻节点之间传输帧。在本实验中，由于客户端与服务器均运行在同一主机上，通信通过回环接口完成，因此在抓包结果中可以观察到报文使用回环链路进行传输。该层主要完成帧的封装以及必要的链路层标识，为上层数据提供可靠的本地传输环境。

网络层（互连层） 网络层负责逻辑寻址与分组转发。本实验中所捕获的报文使用 IPv6 协议进行传输，其源地址与目的地址均为回环地址 ::1，表明通信双方位于同一主机。网络层将传输层报文封装为 IP 数据报，并提供端到端的逻辑寻址能力。

传输层 传输层采用 TCP 协议，为应用层提供面向连接的可靠数据传输服务。该层通过端口号区分不同的应用进程，其中服务器端监听端口为 8000，客户端使用临时端口与之建立连接。TCP 通过序列号、确认号以及重传机制，保证 HTTP 报文在传输过程中的可靠性与有序性。

应用层 应用层采用 HTTP 协议，用于实现浏览器与 Web 服务器之间的资源请求与响应。HTTP 报文作为应用层数据，被封装在 TCP 报文段中进行传输。在本实验中，客户端通过 HTTP GET 请求获取 HTML 页面及其相关资源，服务器返回相应的 HTTP 响应，从而完成页面内容的加载过程。

通过上述分层分析可以看出，HTTP 报文在实际传输过程中依赖于下层各协议的协同工作，各层各司其职，共同完成从应用数据到物理传输的封装与传递过程。

实验内容分析 结合 http 请求/响应报文的抓包结果可以看出，本次 HTML 页面加载过程严格遵循 HTTP/1.1 协议的请求—响应模型。在启用 `http` 显示过滤器后，捕获结果中仅保留 HTTP 协议相关报文，表明浏览器与服务器之间的应用层通信过程清晰、完整，未混入其他无关协议数据。

在页面访问过程中，浏览器首先向服务器发送针对 `main.html` 的 HTTP GET 请求，服务器返回对应的 HTML 文档作为响应。浏览器在接收到 HTML 内容后，对页面进行解析，并根据文档中引用的外部资源路径，继续向服务器发起多次 GET 请求以获取所需的图像文件。抓包结果中依次出现的 6 次 JPG 图像请求与响应，表明页面中引用的静态资源均被正确解析并成功加载。

此外，在所有页面资源加载完成后，浏览器还自动向服务器发起了一次针对页面图标的 GET 请求。由于实验页面中未设置对应的图标文件，服务器返回了状态码为 404 的响应。该现象属于浏览器的默认行为，并不影响页面主体内容的正常显示，也不属于实验实现错误。

从请求与响应报文的字段内容可以看出，请求头中包含了完整的客户端环境信息与内容协商参数，服务器则通过响应行与响应头明确告知请求处理结果及返回数据的类型。整体交互过程中，各 HTTP 报文均在同一 TCP 持久连接上顺序传输，未出现丢包或乱序现象，说明本次 HTTP 加载过程在内容完整性与交互有序性方面均符合协议设计预期。

七、 总结

本次实验围绕 Web 服务器的搭建与 HTTP 协议交互过程展开，笔者在本地主机环境下利用 Python 内置的 HTTP 服务模块成功构建了简易 Web 服务器，并通过浏览器访问自编写的 HTML 页面，实现了页面文本信息与多张静态图像资源的正常加载。在此基础上，使用 Wireshark 对客户端与服务器之间的通信过程进行了抓包与分析，完整观察并验证了 HTTP 报文在 TCP 连接上的实际传输过程。

通过对抓包结果的分析，笔者清晰地识别并理解了 TCP 三次握手建立连接、HTTP 请求—响应交互以及 TCP 四次挥手释放连接的全过程，进一步掌握了 HTTP 协议在真实网络环境中的运行机制。同时，通过对请求报文与响应报文字段的逐项解析，加深了对 HTTP 报文结构、首部字段含义以及持久连接机制的理解。

此外，本实验还从协议分层角度出发，对 HTTP 报文在数据链路层、网络层、传输层和应用层中的封装关系进行了系统分析，使抽象的网络协议模型与实际抓包结果形成了直观对应，有效提升了对计算机网络协议栈整体协作方式的认识。

总体而言，本次实验顺利完成了所有既定目标，不仅强化了对 HTTP 与 TCP 协议工作原理的理解，也提升了使用 Wireshark 进行网络协议分析的实践能力，为后续更复杂网络应用与协议分析实验奠定了良好基础。

参考文献

[1] 中国科学院大学计算机科学与技术学院. Http/ . *https://baike.baidu.com/item/http/*243074, 2025.