实验 3：配置 Web 服务器 & HTTP 报文分析
计算机网络第三次实验报告

姓名：郭佳成　学号：2311990　专业：密码科学与技术

2025 年 12 月 26 日

目录

1 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（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。

2 实验原理

2.1 HTTP 协议详解

HTTP（HyperText Transfer Protocol，超文本传输协议）是应用层协议，是万维网的数据通信基础。它设计用于分布式、协作式和超媒体信息系统。

2.1.1 HTTP 版本演进

在本次实验中，我们主要关注 HTTP/1.0 和 HTTP/1.1 的区别：

- **HTTP/1.0**：默认使用短连接（Short-lived Connections）。每请求一个资源（如 HTML 页面或图片），都要新建一个 TCP 连接。传输完成后立即断开。优点是实现简单，缺点是连接建立开销大，延迟高。
- **HTTP/1.1**：引入了持久连接（Persistent Connections，即 Keep-Alive）。允许在同一个 TCP 连接中发送多个请求和接收多个响应，显著减少了 TCP 握手和挥手的开销。

2.2 TCP 传输控制协议

TCP（Transmission Control Protocol）提供面向连接的、可靠的字节流服务。

2.2.1 有限状态机 (FSM)

TCP 的连接管理是一个复杂的状态机，这个在我们上一次的实验 (Lab2) 中很好的体现过了，主要包含以下两个过程：

- **建立连接：**双方从 `CLOSED` 状态开始。
  - 主动打开方（Client）发送 SYN，进入 `SYN_SENT`。
  - 被动打开方（Server）收到 SYN 回复 SYN+ACK，进入 `SYN_RCVD`。
  - 双方最终进入 `ESTABLISHED` 状态，开始传输数据。
- **断开连接：**
  - 主动关闭方发送 FIN，进入 `FIN_WAIT_1`。
  - 被动关闭方回复 ACK，进入 `CLOSE_WAIT`；随后发送 FIN，进入 `LAST_ACK`。
  - 主动关闭方收到 FIN 回复 ACK，进入 `TIME_WAIT`，等待 2MSL 后关闭。

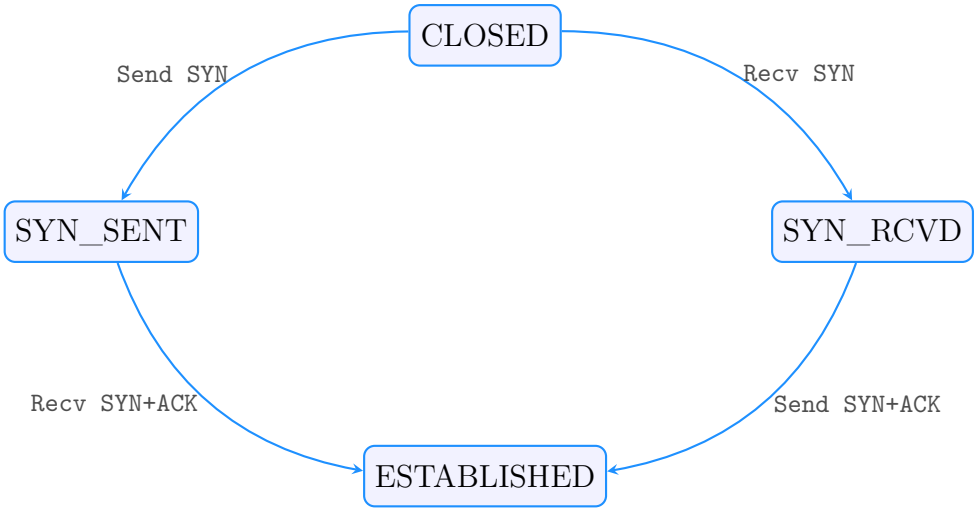

图 1: TCP 建立连接状态流转示意图

2.2.2 端口复用

Web 服务默认监听 TCP 80 端口。在本实验中，由于我选用的是自己搭建博客的服务器为了避开可能存在的 HTTPS（443）强制跳转或 80 端口占用，我使用非标准端口（1234 本次实验）进行通信。客户端则使用随机的高位端口（Ephemeral Port）作为源端口。

3 实验环境

3.1 网络拓扑

本次实验采用 Client-Server 架构，通过公网进行通信也就是我的主机与我的阿里云服务器。

- **Web 服务器 (Server)：**
  - 平台：阿里云 ECS (Linux Ubuntu)
  - IP 地址：60.205.14.222
  - 软件栈：Python 3.6.8 内置 `http.server` 模块
  - 监听端口：TCP 1234
- **客户端 (Client)：**
  - 平台：Windows 11 本地主机
  - IP 地址：10.130.85.114 (局域网 IP)
  - 浏览器：Microsoft Edge (Chromium 内核)
  - 抓包工具：Wireshark 4.0.6

4 实验过程

4.1 网页设计与实现

根据实验要求，编写 `index.html`。页面使用 HTML5 标准结构，通过 CSS 进行简单排版，确保包含个人信息及 6 张图片，并且 html 的设计并不是此次实验的重点，于是编写的是一个很简单的页面也方便我们抓包。

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
    <meta charset="UTF-8">
    <title>计 算 机 网 络 实 验 三 - 郭 佳 成</title>
    <style>
        body {
            font-family: 'Microsoft YaHei', sans-serif;
            text-align: center;
            background-color: #f0f2f5;
            margin: 0;
            padding: 20px;
        }
```

```
        .container {
            max_width: 800px;
            margin: 0 auto;
            background-color: white;
            padding: 40px;
            border-radius: 10px;
            box-shadow: 0 4px 6px rgba(0,0,0,0.1);
        }
        h1 { color: #333; }
        .info {
            margin: 20px 0;
            padding: 20px;
            background-color: #e8f5e9;
            border-radius: 5px;
            border-left: 5px solid #4caf50;
            text-align: left;
        }
        .info p { margin: 10px 0; font-size: 1.1em; }
        .gallery {
            display: grid;
            grid-template-columns: repeat(auto-fit, minmax(200px, 1fr
                ));
            gap: 20px;
            margin-top: 30px;
        }
        .gallery img {
            width: 100%;
            height: 150px;
            object-fit: cover;
            border-radius: 5px;
            box-shadow: 0 2px 4px rgba(0,0,0,0.1);
            transition: transform 0.3s ease;
        }
        .gallery img:hover {
            transform: scale(1.05);
        }
    </style>
</head>
<body>
    <div class="container">
```

```
53          <h1>计 算 机 网 络 实 验 三： Web 服 务 器 测 试</h1>
54
55          <div class="info">
56              <p><strong>姓 名： </strong>郭 佳 成</p>
57              <p><strong>学 号： </strong>2311990</p>
58              <p><strong>专 业： </strong>密 码 科 学 与 技 术</p>
59          </div>
60
61          <div class="gallery">
62              <img src="images/image_1.png" alt="实 验 图 片">
63              <img src="images/image_2.png" alt="实 验 图 片">
64              <img src="images/image_3.png" alt="实 验 图 片">
65              <img src="images/image_4.png" alt="实 验 图 片">
66              <img src="images/image_5.png" alt="实 验 图 片">
67              <img src="images/image_6.jpg" alt="实 验 图 片">
68          </div>
69      </div>
70  </body>
71  </html>
```

Listing 1: index.html

4.2 服务端部署与启动

在服务器端，我们采用 Python 的标准库快速搭建 HTTP 服务。该服务默认以当前目录为根目录，能够处理 GET 请求并返回静态文件。

```
1  # 在 www 目 录 下 执 行
2  python3 -m http.server 1234
```

如图所示，我通过 vscode 的 ssh 连接上我的服务器然后运行上面的命令，可以看到服务已成功启动并监听在 0.0.0.0 的 1234 端口。

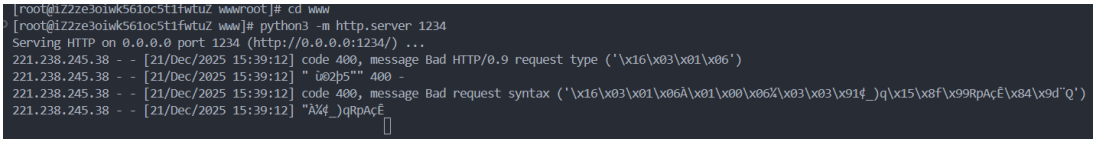

图 2: Python Web 服务器启动界面

4.3 客户端访问与抓包

打开 Wireshark，选择正在使用的网卡（WLAN），输入过滤器 `tcp.port == 1234` 以捕获所有相关的 TCP 流量（包含握手和挥手）。为了专注于应用层，我们下面将使用 `http` 过滤器进行筛选。随后在浏览器地址栏输入 http://60.205.14.222:1234。可以看到页面成功加载，图片显示正常。

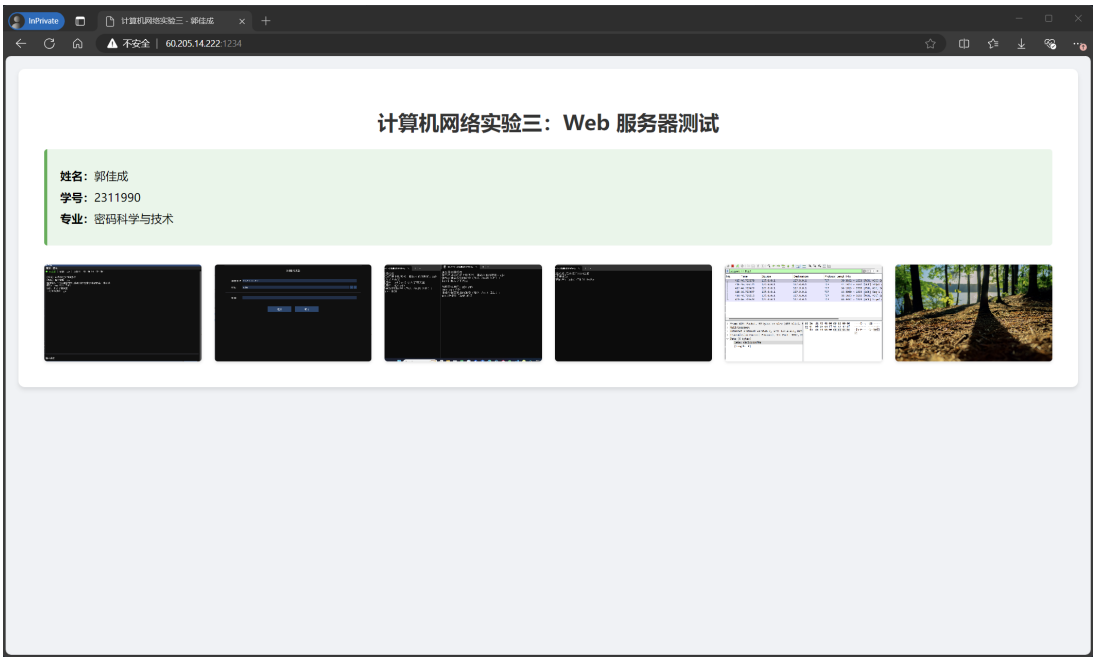

图 3: 客户端成功访问网页

然后我们打开 Wireshrak, 设置好端口号就可以开始抓包了，这下面图片的是一个大概，先三次握手后就开始 GET 请求，分别是文本的 html 和网页上面的 6 张图片。

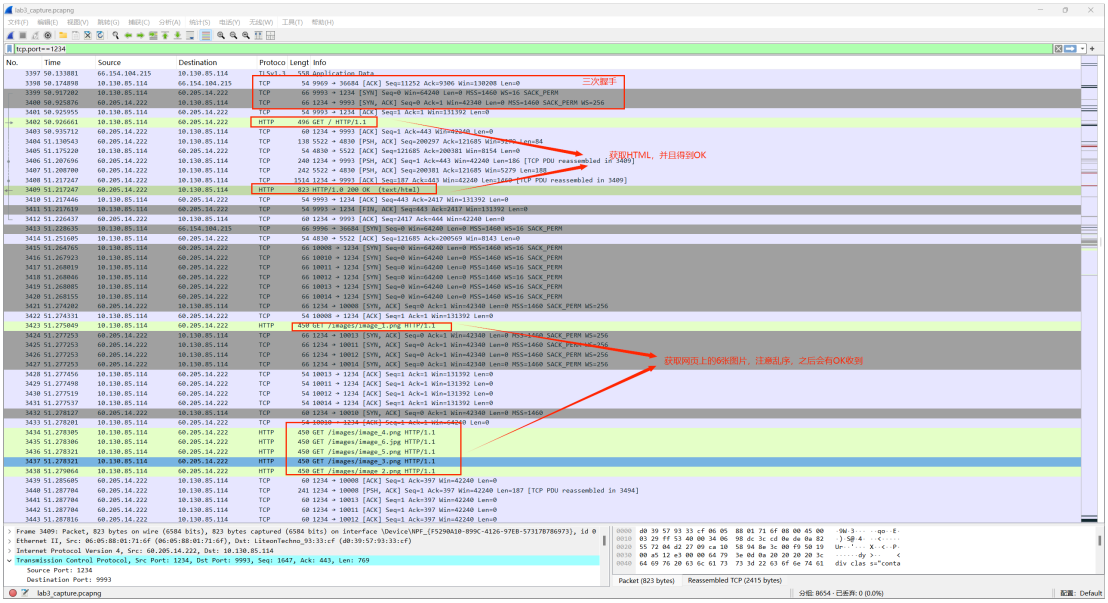

图 4: Wireshark 抓包概览

5 Wireshark 抓包详细分析

5.1 协议交互概览

为了满足实验要求，我们在 Wireshark 中应用 `http` 过滤器，仅显示 HTTP 协议层面的交互。可以看到清晰的”GET” 请求和”200 OK” 响应。

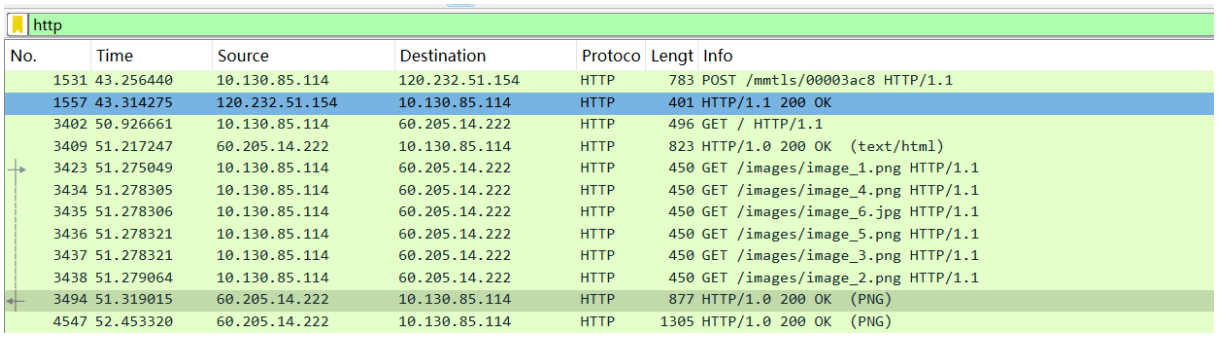

图 5: Wireshark HTTP 协议交互概览

5.2 TCP 三次握手 (Three-way Handshake)

TCP 是可靠的传输层协议，在发送数据前必须建立连接。本次实验中，客户端端口为 9993，服务器端口为 1234。

5.2.1 第一次握手 (SYN)

客户端发送 SYN 报文，请求建立连接。

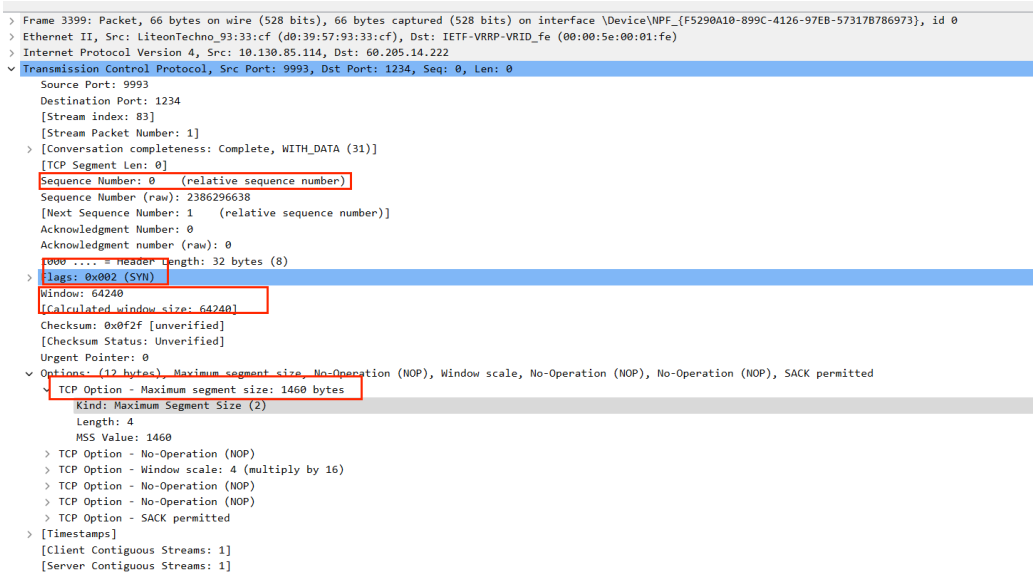

图 6: 第一次握手：客户端发送 SYN

解析：

- **Flags：** SYN 置 1，表示这是一个同步序列号的控制报文。
- **Seq：** 0 (Relative)。Wireshark 为了显示方便，将初始序列号 (ISN) 显示为相对值 0。实际的绝对序列号是一个随机生成的 32 位整数，目的是为了防止网络中滞留的旧报文段干扰新的连接。
- **Window Size：** 64240。客户端通告自己的接收窗口大小，告诉服务器“我现在的接收缓冲区有 64KB 空闲”。
- **MSS (Maximum Segment Size)：** 1460。这是以太网 MTU (1500) 减去 IP 头 (20) 和 TCP 头 (20) 后的最大载荷长度，目的是避免 IP 层分片。

5.2.2 第二次握手 (SYN+ACK)

服务器收到 SYN 后，回复 SYN+ACK。

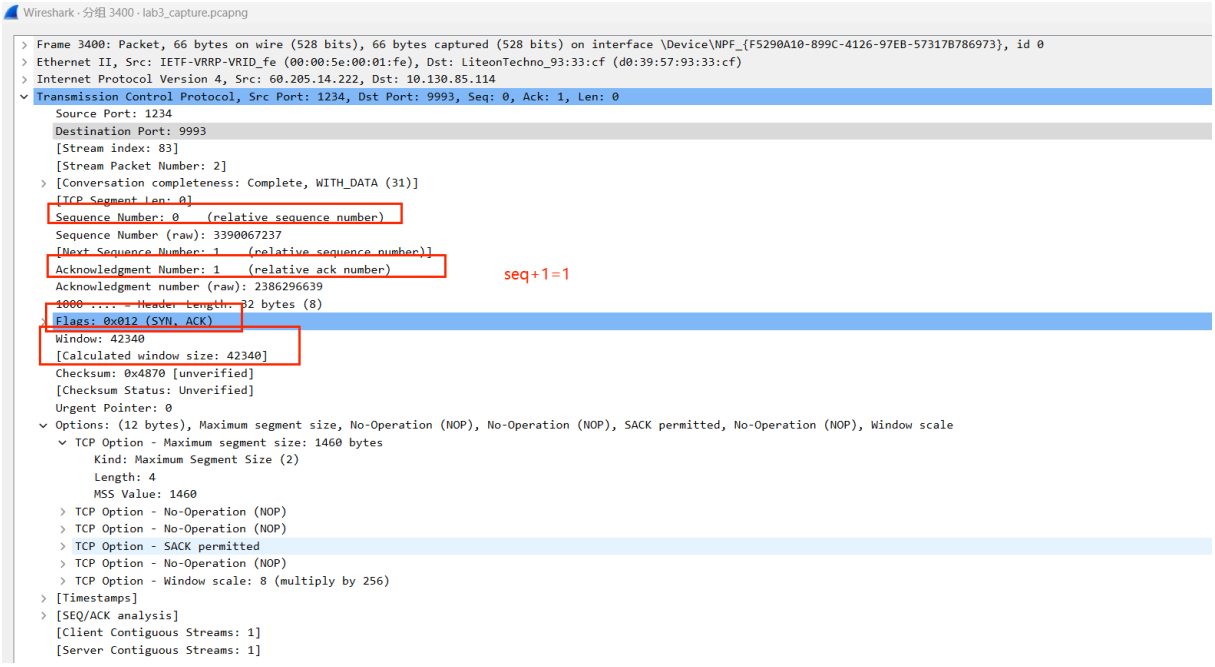

图 7: 第二次握手：服务器回复 SYN+ACK

解析：

- **Flags：** SYN，ACK 同时置 1。ACK 用于确认客户端的 SYN，SYN 用于同步服务器自己的序列号。
- **Seq：** 0 (Relative)。服务器生成的 ISN。
- **Ack：** 1。确认号为客户端 Seq + 1，表示服务器期待收到的下一个字节序号是 1。
- **Window Size：** 42340。服务器的接收窗口略小于客户端。

5.2.3 第三次握手 (ACK)

客户端收到响应后，发送 ACK 确认。

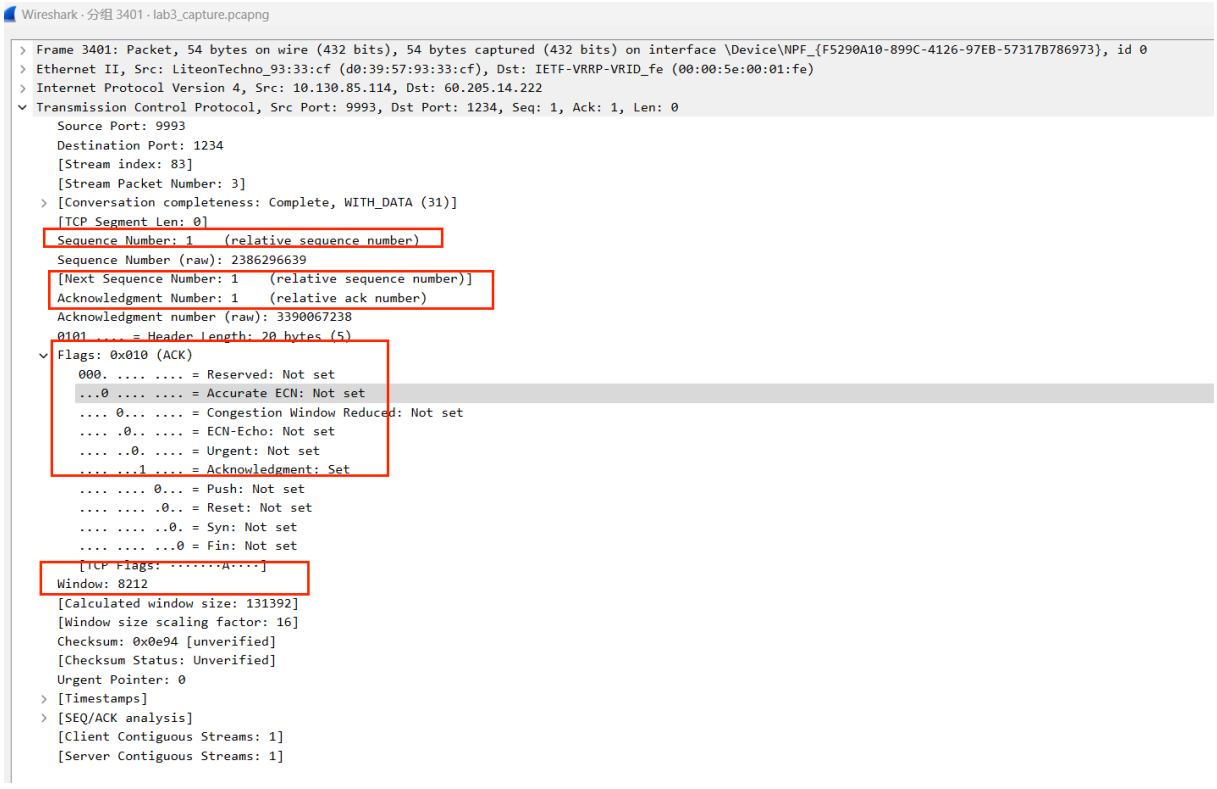

图 8: 第三次握手：客户端发送 ACK

解析：

- **Flags：** ACK 置 1。
- **Seq：** 1。
- **Ack：** 1。确认服务器的 SYN。
- **状态迁移：** 发送完此包后，客户端进入 ESTABLISHED 状态；服务器收到此包后，也进入 ESTABLISHED 状态。连接正式建立，双方可以开始传输 HTTP 报文。

5.3 HTTP 协议交互分析

5.3.1 HTTP 请求报文 (GET)

握手完成后，客户端立即发送 HTTP GET 请求（Frame 3402）。

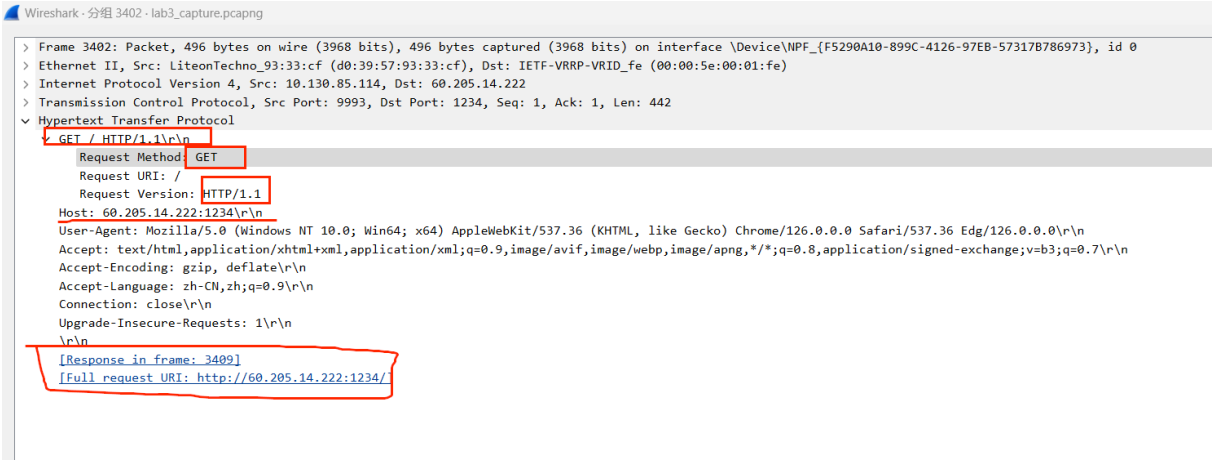

图 9: HTTP GET 请求报文详情

关键头部字段：

- **Request Method：** GET。这是最常用的 HTTP 方法，用于请求指定的资源。
- **Request URI：** /。表示请求根目录下的默认文档（index.html）。
- **Host：** 60.205.14.222:1234。HTTP/1.1 协议规定必须包含 Host 头，用于虚拟主机技术（即在同一 IP 上托管多个域名）。
- **User-Agent：** Mozilla/5.0 ... Chrome/126.0 ...。这串长字符串告诉服务器，客户端是运行在 Windows 10 上的 Chrome 内核浏览器，服务器可据此返回适配 PC 端页面的内容。

5.3.2 HTTP 响应报文 (200 OK)

服务器处理请求，返回 HTML 内容（Frame 3409）。

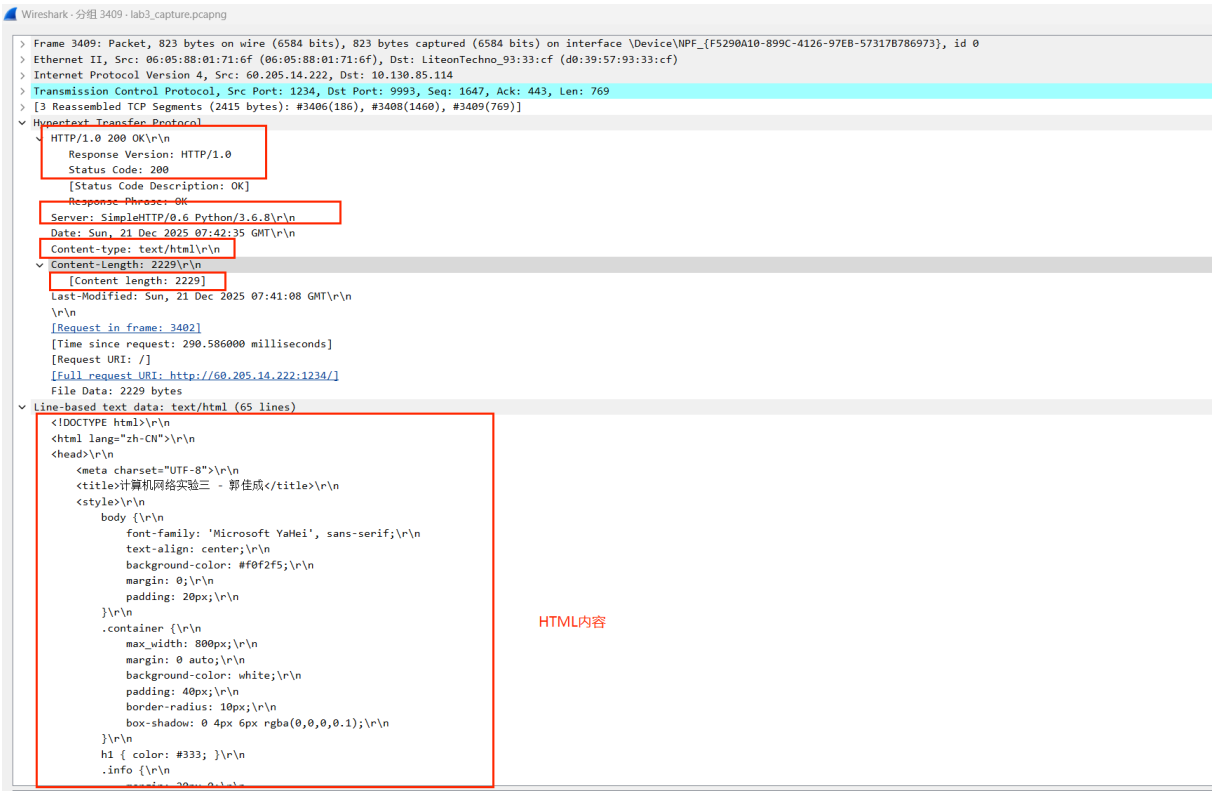

图 10: HTTP 200 OK 响应报文详情

关键头部字段：

- **Status Code：** `200 OK`。这是最理想的响应码，表示请求已成功被服务器理解并处理。
- **Server：** `SimpleHTTP/0.6 Python/3.6.8`。服务器自报家门，显示它是由 Python 3.6.8 的 SimpleHTTP 模块搭建的。
- **Content-Type：** `text/html`。MIME 类型，明确告知浏览器响应体是 HTML 文本，浏览器应启动 HTML 解析器进行渲染，而不是当作纯文本显示或下载。
- **Content-Length：** 2229。响应体的长度。TCP 是流式传输，应用层需要通过此字段（或 Chunked 编码）来确定消息在哪里结束。

5.4 报文封装层次深度剖析

为了深入理解计算机网络体系结构，我们选取 Frame 3402 (GET 请求) 进行自底向上的逐层剥离分析。

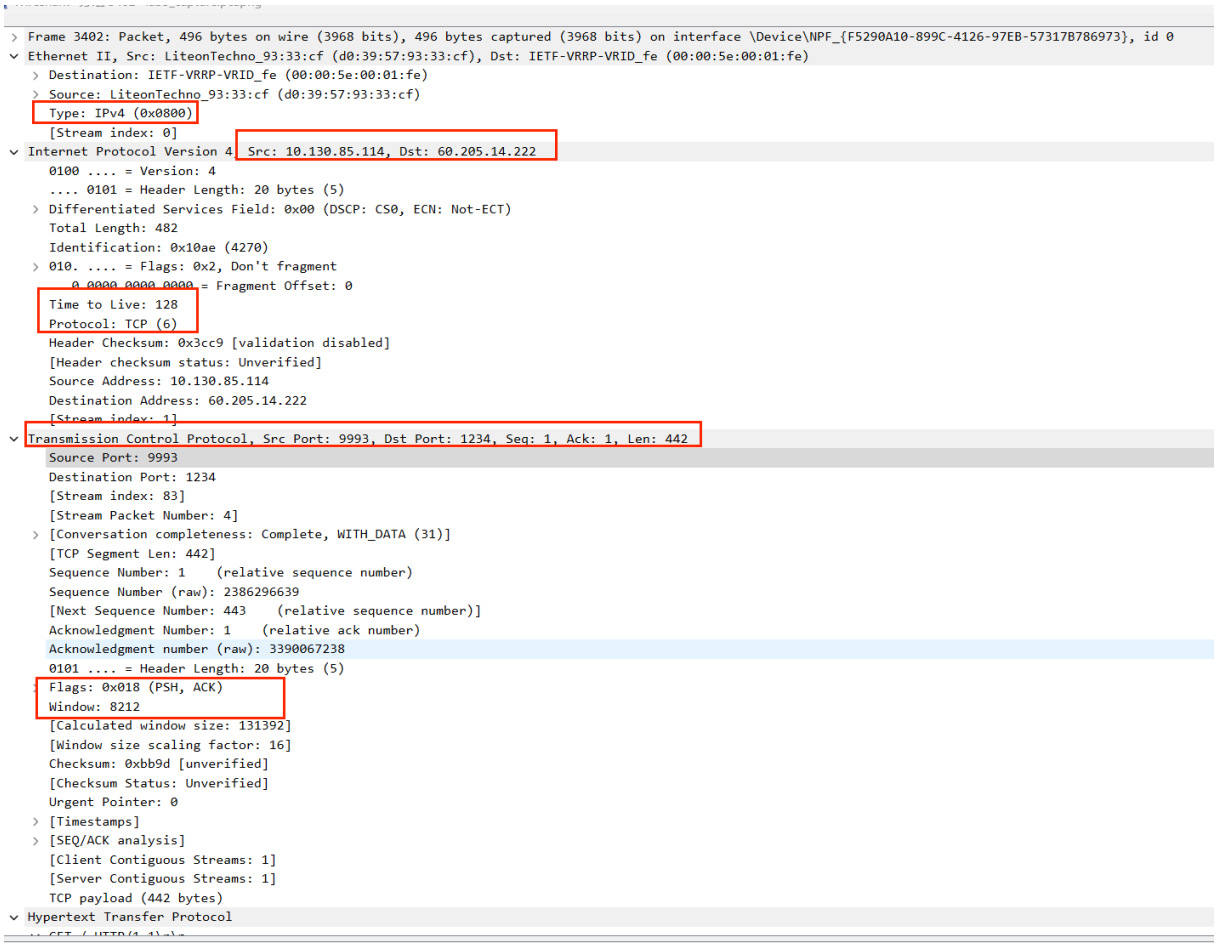

图 11: HTTP 请求报文的四层封装结构

1. **数据链路层 (Ethernet II)**：负责相邻节点间的帧传输。
   - **Src MAC**：`d0:39:57:93:33:cf`。这是本机无线网卡的物理地址，全球唯一。
   - **Dst MAC**：`00:00:5e:00:01:fe`。这是我所在局域网网关（路由器）的 MAC 地址。因为目标 IP 在公网，本机无法直接送达，只能先发给网关进行路由。
   - **Type**：`0x0800`。类型字段，0800 表示上层封装的是 IPv4 协议。
2. **网络层 (IPv4)**：负责主机到主机的逻辑寻址和路由。
   - **Src IP**：`10.130.85.114`。本机的私有 IP 地址。
   - **Dst IP**：`60.205.14.222`。服务器的公网 IP 地址。
   - **Protocol**：`6`。协议号 6 代表传输层使用 TCP 协议（UDP 是 17，ICMP 是 1）。
   - **TTL (Time To Live)**：`128`。生存时间，数据包每经过一个路由器减 1，减到 0 则被丢弃，防止数据包在网络环路中无限循环。

3. **传输层 (TCP)**：负责进程到进程的可靠数据传输。
   - **Src Port**：9993。操作系统随机分配的临时端口，用于区分本机上不同的浏览器标签页或进程。
   - **Dst Port**：1234。服务器监听的 HTTP 服务端口。
   - **Flags**：PSH，ACK。ACK 确认之前的握手，PSH (Push) 提示接收端 TCP 栈应立即将数据推送给应用层，不要在缓冲区等待凑齐更多数据。
4. **应用层 (HTTP)**：负责具体的业务逻辑。
   - 承载了 ASCII 编码的 HTTP 请求行和头部字段，直接面向用户需求（“给我 index.html”）。

5.5 连接释放与“三次挥手”现象

在本次实验中，我们观察到一个特殊的现象：Python 服务器在发送响应的同时设置了 FIN 标志位（No. 3409），这意味着它主动关闭了连接, 这个地方就碰巧和我上一个实验自己实现的四次挥手差不多都是发完 fin 就主动关闭了链接。

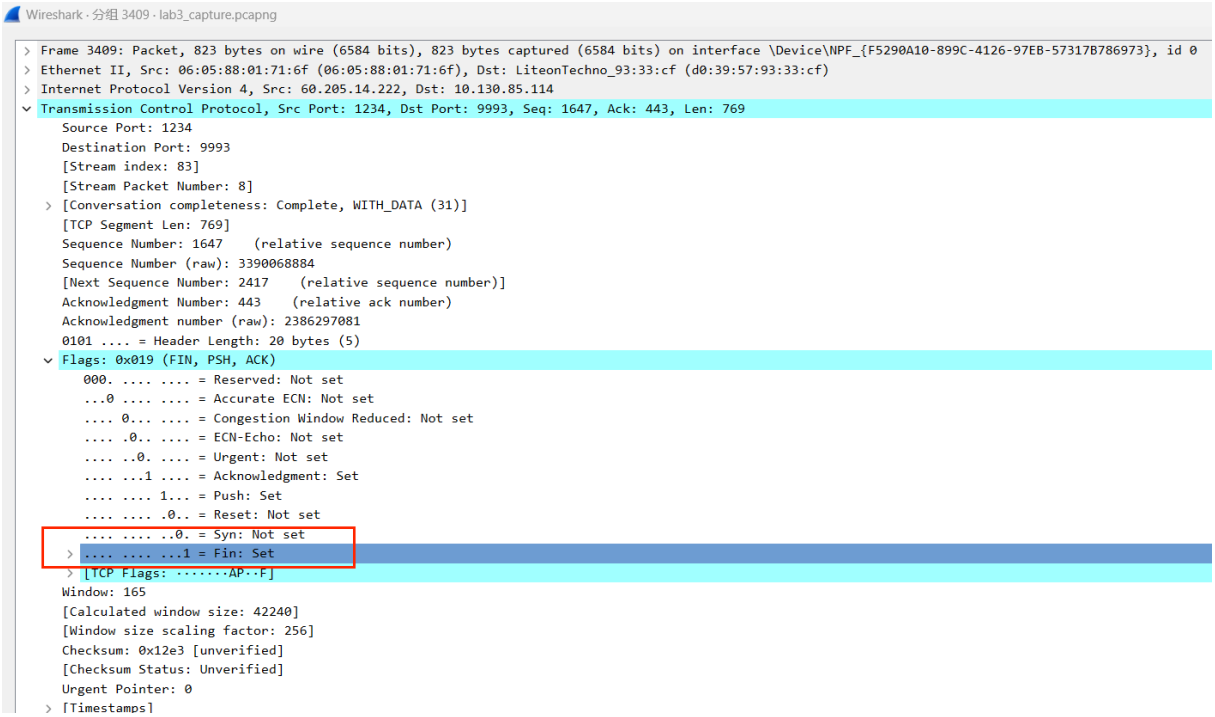

图 12: 服务器发送响应并携带 FIN 标志

标准的四次挥手在这里被优化为了三次交互（捎带确认机制）：

1. **第一次挥手 (Server FIN)**: 服务器在发送完最后一个字节的数据（HTML 内容）后，将 TCP 报文的 FIN 位置 1。这相当于服务器说：“我的数据发完了，我要关闭我的发送通道。”此时服务器进入 FIN_WAIT_1 状态。

2. **第二次/三次挥手 (Client ACK + FIN)**: 客户端收到服务器的 FIN 后，协议栈自动回复 ACK。同时，由于浏览器解析完 HTML 发现不需要复用此连接（或者服务器已经明确关闭），客户端也会立即发送 FIN。在抓包中（Frame 3411），我们看到客户端发送了 `ACK, FIN`。这实际上是将标准流程中的第二次（ACK）和第三次（Client FIN）挥手合并在了一个报文中，提高了效率。

3. **第四次挥手 (Server ACK)**: 服务器收到客户端的 FIN 后，回复最后的 ACK。至此，双向连接彻底关闭。

6 性能与流量统计分析

为了更直观地评估本次 HTTP 交互的性能，我们利用 Python 脚本对抓包文件（*lab3_capture.pcapng*）进行了自动化的统计分析。结果如下表所示：

统计指标	数值	说明
总持续时间	5.6455 秒	从第一个 SYN 包到最后一个 FIN 包的时长
总数据包数	2159 个	包含所有握手、数据传输、挥手包
总流量	2,304,128 字节	约 2.2 MB，主要是 6 张图片的数据
平均吞吐量	398.57 KB/s	受到云服务器上行带宽限制
连接建立次数 (SYN)	14 次	远多于理论上的 7 次（1 HTML + 6 Img）
连接关闭次数 (FIN)	15 次	对应了频繁的连接建立与断开

表 1: 本次 HTTP 交互的流量统计数据

```
(new) D:\大三上\计算机网络\实验\Lab3>python analyze_pcap.py
Analyzing D:\大三上\计算机网络\实验\Lab3\lab3_capture.pcapng...

==================================================
TCP STREAM STATISTICS (TCP 流统计)
==================================================
Total Duration (持续时间): 5.6455 seconds
Total Packets  (总包数):   2159
Total Bytes    (总流量):   2304128 bytes
Throughput     (吞吐量):   398.57 KB/s
SYN Packets:   14 (Connection Requests)
FIN Packets:   15 (Connection Closures)
```

图 13: 运行截图

深入分析：

- **连接数异常分析：**网页包含 1 个 HTML 和 6 张图片，理论上只需 7 次请求。但监测到 14 次 SYN，这可能是因为浏览器并发尝试连接（Pre-connection）、或者请求了 favicon.ico 及其它资源失败重试导致的。
- **短连接特性验证：**每次请求都经历了完整的 SYN → 数据传输 → FIN 生命周期。这证明了 Python `http.server` 默认使用的是非持久连接（Short-lived Connection）。与现代 Nginx/Apache 的 Keep-Alive 模式相比，这种方式会造成大量的 TCP 握手开销，增加页面加载延迟。

7 遇到的问题与思考

7.1 关于"Connection: close" 的思考

在实验中，我最初困惑于为什么每个图片都要重新握手。查阅资料后发现，Python 的 `SimpleHTTP` 模块默认是一次请求对应一个 TCP 连接。

- **问题：**每个资源都要经历 1.5 RTT 的握手延迟，导致加载速度变慢。
- **对比：**现代生产环境（如 Nginx）通常默认开启 Keep-Alive，允许在同一个 TCP 连接中传输多个文件，效率更高。

7.2 端口与防火墙问题

最初配置服务器时，外网无法访问。经排查是云服务器安全组拦截了 1234 端口。放行该端口后，问题解决。这也提醒我在网络编程中要时刻关注网络连通性和防火墙策略。

7.3 浏览器缓存干扰

在初次抓包时，发现只抓到了 304 Not Modified 响应，没有 200 OK 和图片数据。这是因为浏览器缓存了之前的请求。解决方法是开启浏览器的“隐私模式”或在开发者工具中禁用缓存，从而抓到了完整的 HTTP 交互过程。

8 总结

本次实验通过搭建真实的 Web 服务器和抓包分析，让我对 HTTP 协议和 TCP 协议栈有了“肉眼可见”的认识。

1. **验证了分层模型：**从物理地址到 IP 地址，再到端口号和应用层内容，每一层头部信息的剥离都清晰地展示了 OSI 模型的封装思想。

2. **理解了连接状态机：**通过观察 Flags 的变化，我对 TCP 的 $ESTABLISHED$, SYN_SENT, FIN_WAIT_1 等状态流转有了直观的理解。

3. **数据支撑结论：**通过脚本统计分析，量化了短连接模式下的连接开销，用数据验证了“持久连接（Keep-Alive）”的重要性。