Web服务器搭建实验报告

一、实验目的

通过本实验，掌握Web服务器的搭建与配置，了解HTTP协议的基本交互过程，以及使用Wireshark捕获网络数据包的基本操作。

二、实验环境

- **保证客户端和服务器端在同一局域网中**
- **服务器端操作系统**：Vmware:Linux Ubuntu 22.04
- **客户端操作系统**：Windows11
- **服务器端IP地址**：192.168.86.128
- **客户端IP地址**：192.168.86.1
- **Web服务器搭建软件**：Apache2
- **抓包软件**：Wireshark 4.0.8（服务器端抓包）
- **客户端浏览器**：Microsoft Edge

三、实验内容

1. Web服务器搭建

选择在Linux系统上搭建Apache Web服务器，具体步骤如下：

1. 更新软件包列表：

```
sudo apt update
```

2. 安装Apache服务器：

```
sudo apt install apache2
```

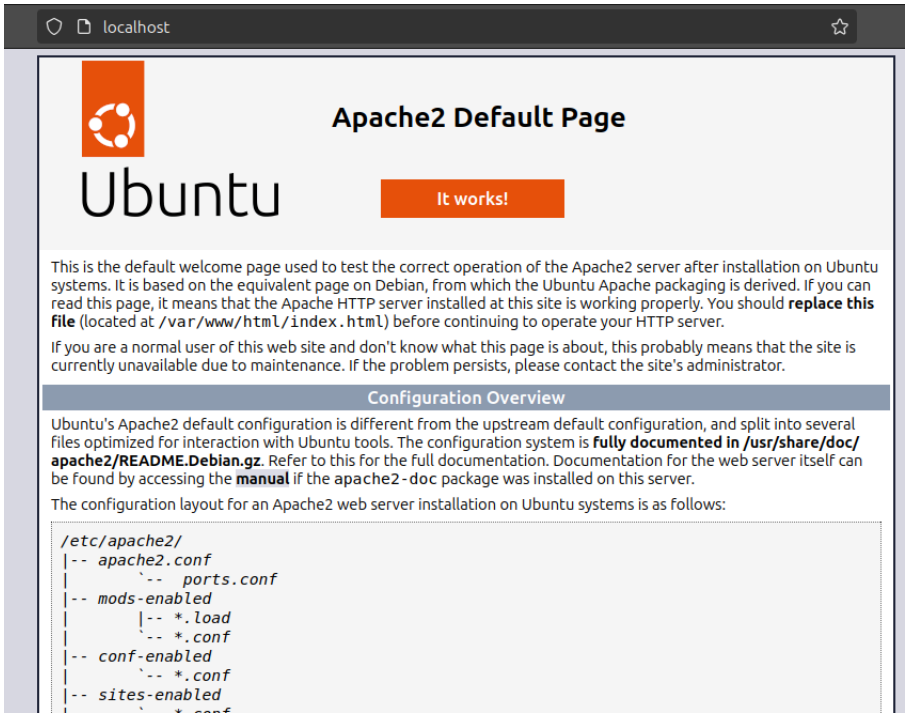
3. 启动Apache服务：

```
sudo systemctl start apache2
```

4. 设置开机自启：

```
sudo systemctl enable apache2
```

5. 验证服务器是否正常运行，在服务器（虚拟机）浏览器中访问 `http://localhost` ，显示Apache默认页面。



2. 简单Web页面制作

因为课程重心不在html文档的美观上（那应该是数据可视化的内容），这里只创建一个包含个人信息的简单HTML页面，文件内容如下：

```
<!DOCTYPE html>
<html lang="zh">
<head>
    <meta charset="UTF-8">
    <title>我的网页</title>
</head>
<body>
    <h1>极简版个人网页</h1>
    <p>专业：计算机科学</p>
    <p>学号：2213739</p>
    <p>姓名：杨涵</p>
    <img src="logo.png" alt="我的LOGO">
    <audio controls>
        <source src="intro.mp3" type="audio/mpeg">
        缺省显示：您的浏览器不支持音频元素。
    </audio>
</body>
</html>
```

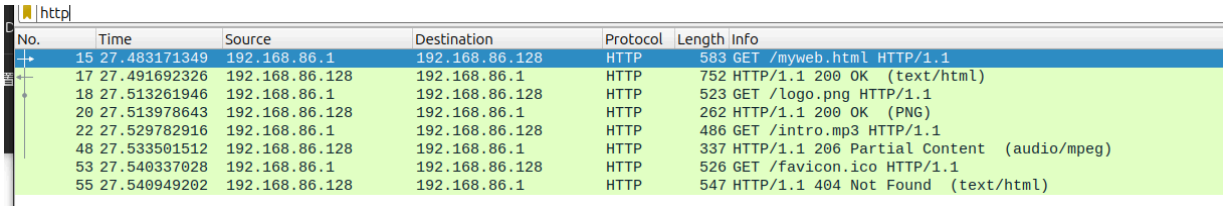
将该文件命名为 `myweb.html` ，并将它和两个资源文件一起放置在Apache的根目录 `/var/www/html/` 下。

3. 客户端访问Web页面

在客户端（物理机）浏览器中输入 `http://192.168.86.128/myweb.html` ，查看制作的个人主页，发现可以正常显示。

4. Wireshark捕获HTTP数据包

1. 服务器端打开Wireshark（root权限下），选择网络接口ens33(服务器端与外界网络连接接口)进行捕获。
2. 设置过滤器为 `http` ，以只显示HTTP协议的相关数据包。
3. 在客户端浏览器中重新访问 `http://localhost/myweb.html` （ctrl f5刷新缓存），观察Wireshark中捕获到的HTTP请求与响应报文。

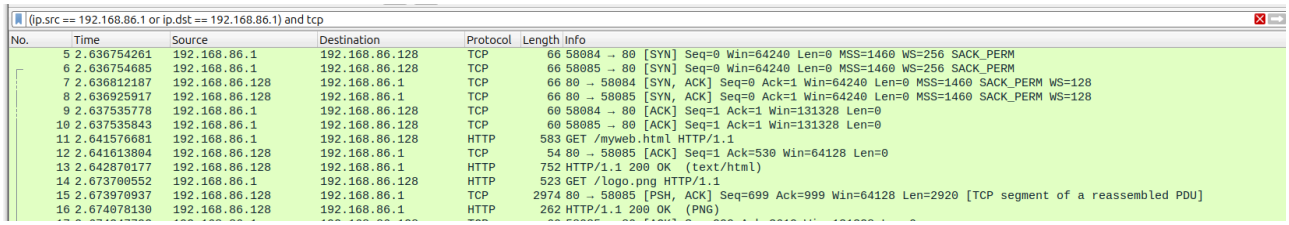


- 结果如图所示，一共有4对http报文，分别对应着html文档、logo.png、intro.mp3、ico（网页图标，即使文档中没有也会请求）的请求与响应。
- 多次实验后发现，多次请求和回复始终秉持一定的顺序，即发送get_a请求后必须收到response_a后才能继续发送get_b。事实证明pipeline在http1.1中虽然被支持，但实际因为准确性问题并没有被广泛采用。
- 课上学的不同http协议版本的区别主要在于对传输层的调用次数不同，如1.0非持续连接每次请求资源都要去新发起TCP连接、接收完立刻断开。
- 协议的版本影响传输效率，但这个区别并不影响http本身的数据包数量，如本实验中html文档资源+图片+音频+浏览器图标共4个文件，理论上无论哪个版本的html协议都会有8个http数据包（请求/响应）。

5. Wireshark捕获TCP数据包

由于HTTP协议是基于TCP协议实现的，这里再给出TCP协议相关数据包的捕获显示：

```
（ip.src == 192.168.86.128 or ip.dst == 192.168.86.128) and tcp
```



TCP协议相关数据包如图所示，其中包含了HTTP数据包。具体交互过程见下面的分析。

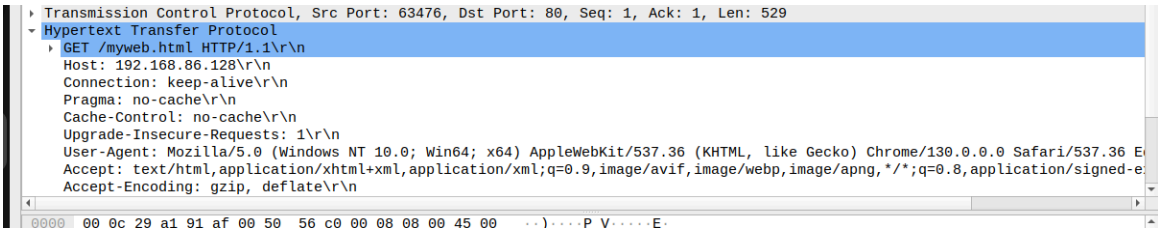
四、HTTP&TCP协议的交互过程说明

在浏览器与Web服务器之间的交互过程中，HTTP（超文本传输协议）负责数据的请求和响应。以下是HTTP交互过程的详细说明：

1. **客户端请求**：
   - 用户在浏览器中输入URL（例如 `http://localhost/myweb.html` ），浏览器构建一个HTTP GET请求，询问服务器提供所请求的资源。
   - 请求报文的基本结构如下：

     ```
     GET /myweb.html HTTP/1.1
     Host: 192.168.86.128（server adr）
     ...
     ```

     抓包结果如下图所示，可以验证：

2. **服务器响应**：
   - 服务器接收到请求后，会处理该请求，查找所请求的资源（例如 `myweb.html` ），并返回HTTP响应。
   - 响应报文的基本结构如下：

     ```
     HTTP/1.1 200 OK
     附加信息，如服务器端地址、更新时间等
     ```

     抓包结果如下图所示，可以验证：

```
Transmission Control Protocol, Src Port: 80, Dst Port: 63476, Seq: 1, Ack: 53
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
  Date: Fri, 25 Oct 2024 12:31:01 GMT\r\n
  Server: Apache/2.4.57 (Ubuntu)\r\n
  Last-Modified: Thu, 24 Oct 2024 13:36:19 GMT\r\n
  ETag: "1ce-625391660868a-gzip"\r\n
  Accept-Ranges: bytes\r\n
  Vary: Accept-Encoding\r\n
```

3. **状态码**：
   - 服务器在响应中包含状态码，用于指示请求的处理结果。例如：
     - `200 OK`：请求成功，服务器返回所请求的资源。
     - `404 Not Found`：请求的资源不存在。
     - `500 Internal Server Error`：服务器遇到错误，无法完成请求。

   如图所示，由于我们没有在文件夹中放入ico文件，请求ico后会返回404 not found：

```
Hypertext Transfer Protocol
  HTTP/1.1 404 Not Found\r\n
  Date: Fri, 25 Oct 2024 12:31:01 GMT\r\n
  Server: Apache/2.4.57 (Ubuntu)\r\n
  Content-Length: 276\r\n
  Keep-Alive: timeout=5, max=100\r\n
  Connection: Keep-Alive\r\n
  Content-Type: text/html; charset=iso-8859-1\r\n
  \r\n
```

4. **资源加载**：
   - 如果响应成功，浏览器解析HTML内容，并根据其中的资源链接（如图片、音频等）发起额外的HTTP请求。
   - 每个附件都将遵循类似的请求和响应过程。例如，当浏览器解析到 `<img src="logo.png">` 时，会发送：

```
GET /logo.png HTTP/1.1
Host: 192.168.86.168（server adr）
...
```

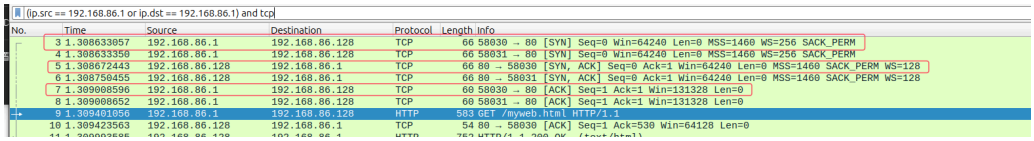
   抓包结果如下图所示，可以验证：

```
Hypertext Transfer Protocol
  GET /logo.png HTTP/1.1\r\n
  Host: 192.168.86.128\r\n
  Connection: keep-alive\r\n
  Pragma: no-cache\r\n
  Cache-Control: no-cache\r\n
```

5. **完成加载**：
   - 所有资源都成功加载后，浏览器将呈现完整的网页，用户可以与之交互。
6. **低层支持**：
   - http作为顶层应用层的一种协议，需要各个低层的协议支持。比较重要的是TCP的建立连接和发送机制、断开过程。
   - 虽然本次实验只要求观察http的数据包，观察TCP包仍然可以验证http1.1的“持久连接”特性——建立连接后多个请求和接收都在同一个TCP连接上进行，而没有每次都断开。

- 以下是TCP协议的具体交互过程：
  - 在HTTP发送第一个数据包后，传输层尝试在src和dest间建立连接，这是一个“3次握手”的过程。
    - 客户端发送一个 SYN 包以请求建立连接。
    - 服务器回复 SYN-ACK 包以确认并响应连接请求。
    - 客户端发送 ACK 包确认连接建立。

      以上过程对应着下面这3个数据包。

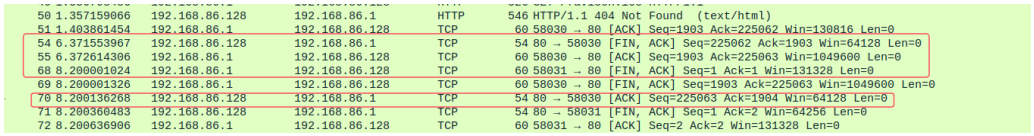


      如图所示，3个圈起来的数据包分别是[SYN],[SYN,ACK]和[ACK]，逻辑正确。

      之所以有很多个数据包（两组），是因为发送和确认的过程是异步的！
  - 建立连接后，每次A给B发一个HTTP数据包，B在收到后会反过来给A发一个ACK确认：

```
 7 0.004770593   192.168.86.1          192.168.86.128        HTTP       583 GET /myweb.html HTTP/1.1
 8 0.004853718   192.168.86.128        192.168.86.1          TCP         54 80 → 65344 [ACK] Seq=1 Ack=530 Win=64128 Len=0
 9 0.007938983   192.168.86.128        192.168.86.1          HTTP       752 HTTP/1.1 200 OK  (text/html)
10 0.054084883   192.168.86.1          192.168.86.128        TCP         60 65344 → 80 [ACK] Seq=530 Ack=699 Win=130560 Len=0
```

    这个过程中，还可能涉及到PSH等，我们不深究具体过程！
  - 所有数据发送完毕后，TCP选择断开连接，这是一个“4次挥手”的过程，TCP的4次挥手断开连接过程可以详细描述为：
    - 一方（如服务器）发送一个 FIN 包以请求断开连接。
    - 另一方（如客户端）回复 ACK 确认接收到断开请求。
    - 服务器发送 FIN 包以请求断开连接。
    - 客户端回复 ACK 确认断开连接。



    如图所示，4个圈起来的数据包分别是
    - **FIN:server->client**,
    - **ACK:client->server**,
    - **FIN:client->server**,
    - **ACK:client->server**。

      逻辑正确。之所以看起来有很多个数据包，同样是因为发送和确认的过程是异步的，而且抓包时间并不是到达时间（甚至可能有遗漏）！上面这个结果已经是多次实验之后取到的一个结果相对明晰的了。

通过这一过程，HTTP协议确保了客户端与服务器之间的高效数据传输和交互。

五、实验总结

通过本次实验，掌握了Web服务器的搭建与配置，熟悉了简单HTML页面的制作以及HTTP协议的交互过程。同时，了解了Wireshark的基本使用方法，为今后网络编程与调试打下了良好的基础。