南開大學

计算机网络实验报告

Lab2

学院：网络空间安全学院

专业：信息安全、法学

学号：2113203

姓名：付政烨

班级：信安法班

摘要

在本次实验中，我们主要探讨了如何搭建一个简单的 Web 服务器并通过浏览器访问其提供的页面，同时也对浏览器与服务器之间的交互过程进行了深入的捕获和分析。首先，我们选择 Flask 来搭建 Web 服务器。在服务器上，创建了一个简洁的 Web 页面。页面的设计相对简单，但它成功地满足了所有的要求。随后，使用了浏览器来访问这个页面。为了能够深入了解浏览器与 Web 服务器之间的交互过程，我们使用了 Wireshark 来捕获整个通信过程。值得注意的是，为了确保捕获的数据包清晰可读，通过 Wireshark 的捕获结果，我们可以清楚地看到浏览器发起的请求和服务器的响应，包括请求和响应的头部信息、所使用的 HTTP 方法以及返回的内容等。这不仅帮助我们更好地理解 HTTP 的工作机制。本次实验不仅提供了一个实际的 Web 服务器搭建的经验，还通过 Wireshark 的使用，加深了我们对 Web 通信机制的理解。

关键字：Web 服务器搭建；　Wireshark 捕获；　HTTP 通信分析

目录

一、 实验要求 **1**

二、 Web 服务器搭建 **1**

（一） 安装 Flask 1
（二） 创建基本的 Flask 应用 1
（三） 运行服务器 2
（四） 在浏览器中访问 2
（五） 部署到公网 3

三、 编写 Web 页面（HTML） **3**

四、 Wireshark 捕获交互过程 **5**

（一） 基本信息分析 5
（二） 请求报文分析 5
（三） 响应报文分析 7
（四） TCP 三次握手 8
（五） TCP 四次挥手 10

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。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。

二、 Web 服务器搭建

(一) 安装 Flask

使用 pip（Python 的包管理工具）安装 Flask：

```
pip install Flask
```

(二) 创建基本的 Flask 应用

创建一个简单的 Flask Web 应用程序 (app.py)，用于显示 templates 文件夹中的 index.html 文件内容。当用户访问应用程序的根 URL 时，它将显示该文件的内容。

```
from flask import Flask,render_template

app=Flask(__name__)

# 创建了网址 /show/info 和函数 index 的对应关系，以后用户在浏览器上访问 /show/info 网站自动执行 index
@app.route('/')
def index():
    # Flask会默认区templates文件夹中寻找文件，读取内容，给用户返回
    return render_template("index.html")

if __name__=='__main__':
    app.run(host='0.0.0.0', port=8080)
```

导入必要的库

- 从 Flask 库中导入 Flask 和 render_template。

创建 Flask Web 应用实例

- app = Flask(__name__) 这行代码创建了一个 Flask Web 应用的实例，其中 __name__ 是当前模块的名字。

定义路由和视图函数

- 使用 @app.route('/') 装饰器定义了一个路由，它指向网站的根目录 /。

- 当用户访问这个 URL（例如 http://localhost:8080/）时，它将执行下面的 index() 函数。这个函数返回的内容是 index.html 的模板内容。render_template 函数用于从 templates 文件夹中加载 index.html 并返回其内容。注意，需要在应用程序的同一目录下有一个名为 templates 的文件夹，并且这个文件夹中应该包含一个名为 index.html 的文件。

运行应用

- 在 if ___name___ == '___main___': 下面的代码是确保该应用只在直接运行此脚本时启动，而不是在从其他脚本导入时启动。
- app.run(host='0.0.0.0', port=8080) 使应用开始运行，并监听所有 IP 地址（0.0.0.0）上的 8080 端口。这意味着任何能够访问此服务器的计算机都可以访问该应用。

(三) 运行服务器

在命令行中，导航到 app.py 所在的目录，然后运行：

```
python app.py
```

此时，应该会看到一个消息，告诉我们服务器正在运行，并监听 127.0.0.1:8080。

图 1

(四) 在浏览器中访问

在 web 浏览器中，访问 http://127.0.0.1:8080/，看到的消息。

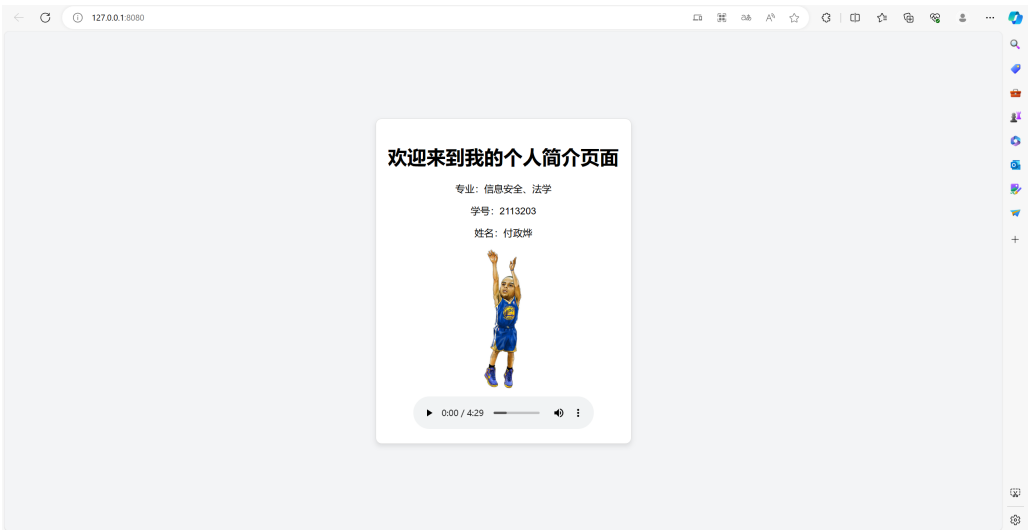


图 2

（五） 部署到公网

上述步骤只是在本地计算机上启动了服务器。如果想让其他人也能访问，需要将其部署到一个公共服务器上。有许多方法可以做到这一点，例如使用 Gunicorn、uWSGI 作为 WSGI 服务器，然后使用 Nginx 或 Apache 作为反向代理。此外，还有许多云平台，如 Heroku、Google Cloud 和 AWS，都支持 Flask 应用的部署。（由于本次实验没有要求，这里不再进行详细展示）

三、 编写 Web 页面（HTML）

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>简介</title>
    <style>
        /* 全局样式 */
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh; /* 视窗高度 */
            background-color: #f3f4f6; /* 背景颜色 */
        }

        .container {
            text-align: center;
            border: 1px solid #ddd;
            padding: 20px;
            border-radius: 10px;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1); /* 箱子阴影 */
            background-color: #ffffff; /* 容器背景颜色 */
        }

        /* 图片居中 */
        .container img {
            display: block;
            margin: 10px auto;
        }

        /* 响应式设计：当屏幕宽度小于600px时，容器宽度为90% */
        @media (max-width: 600px) {
            .container {
```

```
                width: 90%;
            }
        }
    </style>
</head>

<body>
    <div class="container">
        <h1>欢迎来到我的个人简介页面</h1>
        <p>专业：信息安全、法学</p>
        <p>学号：2113203</p>
        <p>姓名：付政烨</p>
        <img src="{{ url_for('static', filename='logo.png') }}" alt="我的LOGO
            " width="100">
        <audio controls>
            <source src="{{ url_for('static', filename='intro.mp3') }}" type=
                "audio/mpeg">
            您的浏览器不支持音频播放。
        </audio>
    </div>
</body>

</html>
```

以上代码提供了页面的基本结构、样式和内容，以及与 Flask 框架的集成方式，其中包括：

文档类型和元数据

- <!DOCTYPE html>：告诉浏览器这是一个 HTML5 文档。
- <html lang=”en”>：标记 HTML 开始，并设置语言为英文。
- <meta name=”viewport” content=”width=device-width, initial-scale=1.0”>：使页面在移动设备上响应式显示。

样式

- body ...：为整个页面设置背景色，并使内容居中显示。
- .container ...：定义主要内容的容器样式，如边框、背景色和阴影。
- @media (max-width: 600px) ...：响应式设计，当屏幕宽度小于 600px 时，调整容器宽度。

页面内容

- <h1> 欢迎来到我的个人简介页面 </h1>：显示主标题。
- <img src=” url_for(’static’, filename=’logo.png’) ” ...>：使用 Flask 的 url_for 函数展示图片。
- <audio controls>...</audio>：提供一个音频播放器，音频文件来自 Flask 的 static 文件夹。

四、 Wireshark 捕获交互过程

(一) 基本信息分析

在使用 Wireshark 软件时，应选择适当的网络端口以进行数据包捕获。鉴于本次实验中开启的 Web 服务器地址为“127.0.0.1”（即本地回环地址），因此在 Wireshark 选择端口时，应选择“Adapter for loopback traffic capture”，即捕获本地回环（loopback）流量的适配器。在计算机网络中，回环地址（通常为 127.0.0.1）是一个特殊的 IP 地址，用于表示当前计算机。当应用程序在同一台计算机上与自身进行通信时，会使用这个地址。

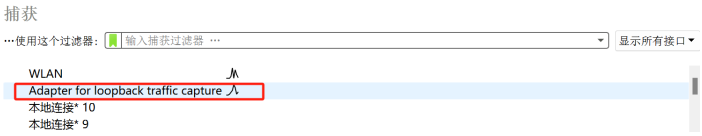


图 3

在启动 Wireshark 进行数据捕获后，利用浏览器访问了上文搭建的网页。随后，可以通过分析捕获到的数据内容，进一步研究其之间的交互过程。经由过滤器进行筛选，主要对协议类型为 HTTP 的数据包进行分析。

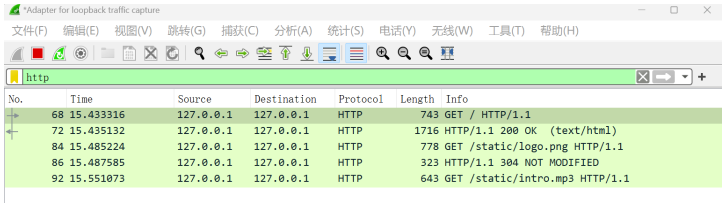


图 4

- No.68、No.84、No.92 数据包：HTTP 请求报文
- No.72、No.86 数据包：HTTP 响应报文

(二) 请求报文分析

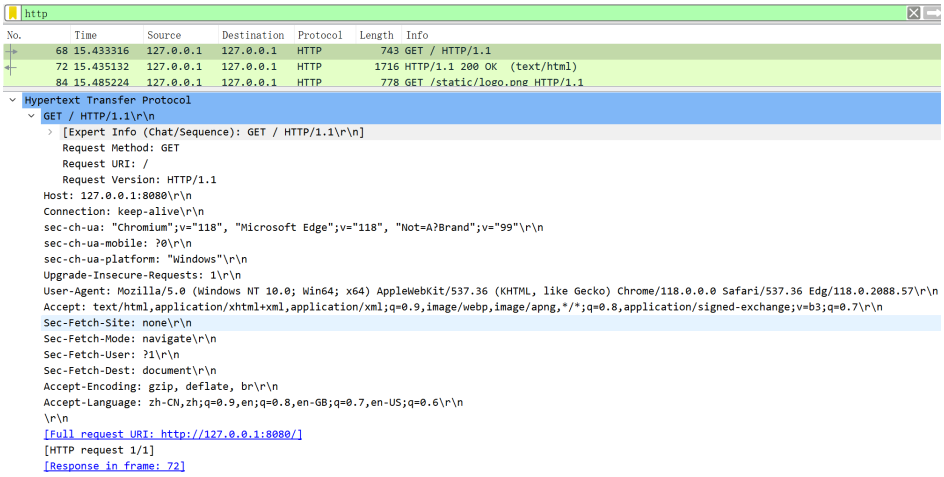


图 5

请求行:

- GET - 表示这是一个 HTTP GET 请求，通常用于请求网页或资源。
- URI: / - 表示请求的资源是服务器的根路径。
- 版本: HTTP/1.1 - 表示此请求使用的 HTTP 版本是 1.1。

请求头:

- Host: 127.0.0.1:8080 - 说明请求的服务器地址是 127.0.0.1，端口是 8080。
- Connection: keep-alive - 保持 TCP 连接活跃，不立即关闭。
- sec-ch-ua: 提供了关于浏览器的信息。例如，此请求来自 Chromium 或 Microsoft Edge 等。
- sec-ch-ua-mobile: ? 0 - 表示此请求不是来自移动设备。
- sec-ch-ua-platform: "windows" - 请求是从 Windows 操作系统发送的。
- Upgrade-Insecure-Requests: 1 - 提示服务器，客户端希望进行不安全到安全的请求升级。
- User-Agent: 提供了更详细的浏览器和操作系统信息。它说请求是从一个 Windows 10 系统上的 Chrome 或 Edge 浏览器发送的。
- Accept: 描述了客户端可以接受的 MIME 类型。例如，它可以接受 HTML、XML、WebP 图像等。
- Sec-Fetch-site: 指定了请求与网站的关系，此处为 none。
- Sec-Fetch-Mode: 指定请求的模式，此处为 navigate。
- Sec-Fetch-User: ?1 - 是否因为用户操作（例如点击链接）而进行的请求。
- Sec-Fetch-Dest: document - 请求的目标是文档。
- Accept-Encoding: 描述了客户端可以接受的内容编码，例如 gzip、deflate 和 br。
- Accept-Language: 描述了客户端可以接受的语言。此处为中文、英文等。

其他信息:

- Full request URI: http://127.0.0.1:8080/ - 提供了完整的请求 URI。
- [HTTP request 1/1]- 这意味着这是捕获的第一个 HTTP 请求。
- [Response in frame: 72]- 表示响应数据在第 72 帧中。

(三) 响应报文分析

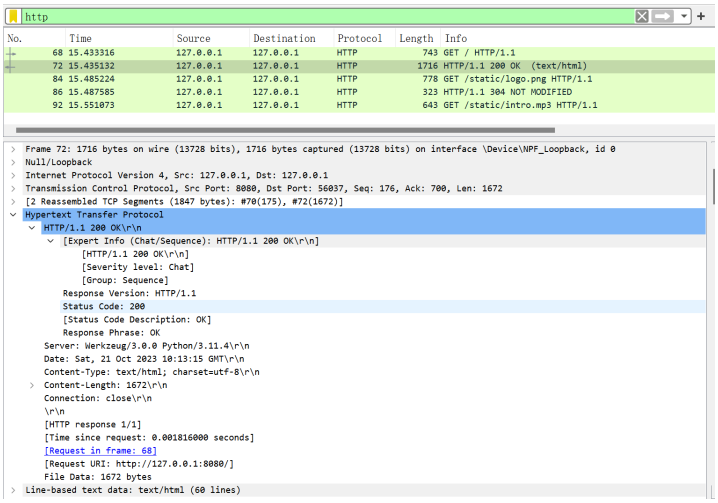


图 6

状态行:

- 版本: HTTP/1.1 - 表示此响应使用的 HTTP 版本是 1.1。
- 状态码: 200 - 这是一个标准的 HTTP 状态码，表示请求成功且服务器已成功处理了请求。
- 响应短语: OK - 这是状态码 200 对应的标准短语。

响应头:

- server: Werkzeug/3.0.0 Python/3.11.4 - 这表明服务器软件是 Werkzeug 版本 3.0.0，它运行在 Python 3.11.4 上。Werkzeug 是一个 WSGI 工具包，用于 Python Web 应用程序开发。
- Date: sat, 21, 0ct 2023 10:13:15 GMT - 这是服务器返回响应的日期和时间。
- Content-Type: text/html; charset=utf-8 - 这表示响应的内容类型是 HTML,并使用 UTF-8 字符集。
- Content-Length: 1672 - 表明响应的正文长度是 1672 字节。
- Connection: close - 这意味着服务器完成此响应后将关闭连接。

其他信息:

- [HTTP response 1/1]- 意味着这是捕获的第一个 HTTP 响应。
- [Time since request: 0.01816000 seconds]- 从请求发送到收到此响应，只花了约 0.01816 秒。
- [Request in frame: 68]- 表示与此响应相关的请求数据在第 68 帧中。
- [Request URI: http://127.0.0.1:8080/]- 提供了与此响应相关的请求 URI。
- File Data: 1672 bytes - 这可能是表示响应正文的数据长度，与 Content-Length 头部相匹配。

(四) TCP 三次握手

TCP 三次握手介绍

TCP 是一种面向连接的、可靠的、基于字节流的传输层通信协议。在 TCP 建立连接时，为了确保连接的可靠性和同步双方的初始序列号，采用了一个被称为“三次握手”的过程。以下是三次握手的详细步骤:

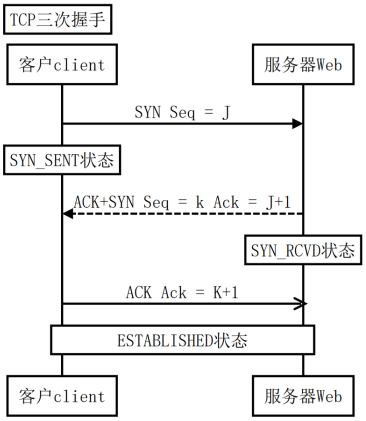


图 7

1. SYN (同步):

- 客户端发送一个 TCP 数据包到服务器。在这个数据包的头部中，SYN 标志位被设置为 1，同时客户端会随机选择一个初始的序列号 J。
- 数据包: Client -> Server [SYN, Seq=x]

2. SYN + ACK (同步 + 确认):

- 服务器收到 SYN 数据包后，为了确认客户端的 SYN，会向客户端发送一个 SYN+ACK 数据包。这个数据包中，SYN 标志位和 ACK 标志位都被设置为 1。服务器也会选择一个自己的初始序列号 y 并且设置 ACK 的值为 $x + 1$ 来确认客户端的序列号。
- 数据包: Server -> Client [SYN, Seq=K, ACK, Ack=J+1]

3. ACK (确认):

- 客户端收到服务器的 SYN+ACK 数据包后，会发送一个 ACK 数据包给服务器，确认服务器的 SYN。这个数据包的 ACK 值会被设置为 $y + 1$。
- 数据包: Client -> Server [ACK, Ack=K+1]

在三次握手完成后，TCP 连接就被成功建立，之后客户端和服务器就可以开始双向数据传输。三次握手不仅确保了连接的可靠性，而且避免了过时的连接请求突然出现在网络中导致的潜在问题。这个三次握手的过程确保了两件事:

1. 双方都知道彼此有能力发送和接收数据（即都是活跃的）。
2. 双方都同步了彼此的初始序列号，这对于 TCP 的可靠数据传输机制来说非常重要。

TCP 三次握手分析

从以下三个数据包中可以看到：这是一个在本机（127.0.0.1）之间的 TCP 连接（从端口 56037 到端口 8080），连接成功建立后，客户端和服务器都同步了各自的初始序列号，并且都知道了彼此的接收窗口大小、窗口大小因子、最大分段大小等参数。这些信息对于后续的数据传输和流量控制非常重要。以下是详细的分析：

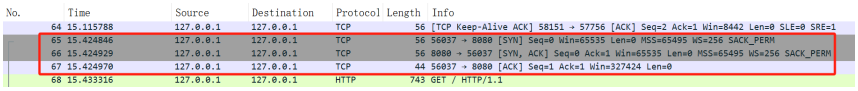


图 8

第 65 个数据包:

- 源地址和端口: 127.0.0.1:56037
- 目的地址和端口: 127.0.0.1:8080
- 标志位: [SYN]- 表示这是一个连接请求。
- 序列号 (Seq): 0 - 这是客户端为此次连接随机选择的初始序列号。
- 窗口大小 (Win): 65535 - 客户端通知服务器它的接收窗口大小。
- 窗口大小因子 (WS): 256 - 这是窗口大小的扩大因子。
- SACK_PERM: 表示客户端支持选择性确认。

第 66 个数据包:

- 源地址和端口: 127.0.0.1:8080
- 目的地址和端口: 127.0.0.1:56037
- 标志位: [SYN, ACK]- 表示这是对客户端 SYN 的响应，同时也是服务器的 SYN 请求。
- 序列号 (Seq): 0 - 这是服务器为此次连接选择的初始序列号。
- 确认号 (Ack): 1 - 确认客户端的初始序列号 +1。
- 窗口大小 (Win): 65535 - 服务器通知客户端它的接收窗口大小。
- 最大分段大小 (MSS): 65495 - 服务器通知客户端它能接受的最大 TCP 分段大小。
- 窗口大小因子 (WS): 256 - 这是窗口大小的扩大因子。
- SACK_PERM: 表示服务器也支持选择性确认。

第 67 个数据包:

- 源地址和端口: 127.0.0.1:56037
- 目的地址和端口: 127.0.0.1:8080
- 标志位: [ACK]- 表示这是对服务器 SYN 的确认。
- 序列号 (Seq): 1 - 客户端的初始序列号 +1。

- 确认号 (Ack): 1 - 确认服务器的初始序列号 +1。
- 窗口大小 (Win): 327424 - 客户端通知服务器它的新接收窗口大小（经过窗口大小因子调整后的大小）。

（五） TCP 四次挥手

TCP 四次挥手介绍

TCP 是一个面向连接的协议，这意味着在数据传输之前需要建立一个连接，而在数据传输完成后需要终止这个连接。为了正常地终止一个 TCP 连接，TCP 使用了一个被称为“四次挥手”的过程。以下是四次挥手的详细步骤：

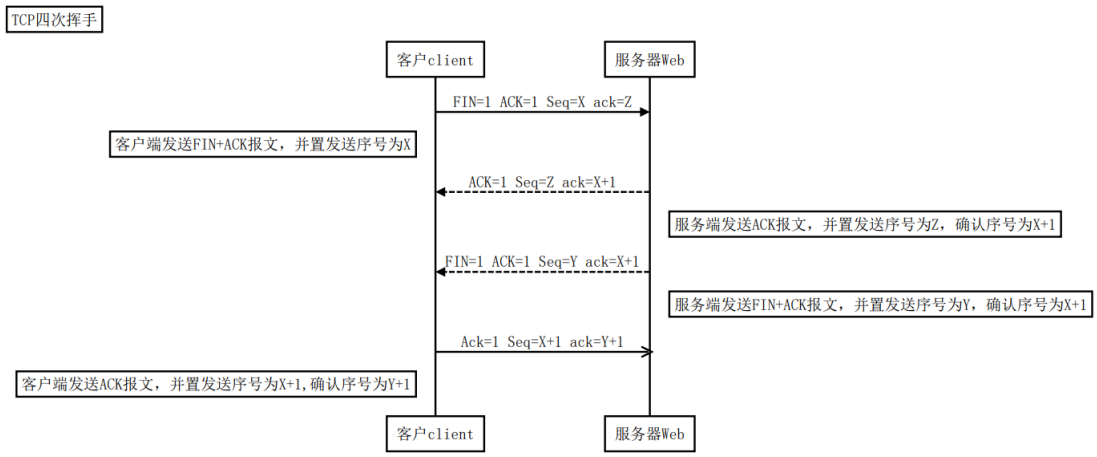


图 9

FIN (结束):

- 主动关闭方发送一个 TCP 数据包到对方。在这个数据包的头部中，FIN 标志位被设置为 1，表示主动方想要关闭连接。数据包: 主动方 $\rightarrow$ 被动方 $[\text{FIN}, \text{Seq} = X]$

ACK (确认):

- 被动关闭方收到 FIN 数据包后，为了确认已收到主动方的终止请求，它会发送一个 ACK 数据包回去。数据包: 被动方 $\rightarrow$ 主动方 $[\text{ACK}, \text{Seq} = Z, \text{Ack} = X + 1]$

FIN (结束):

- 被动关闭方在某个时刻后，也发送一个 FIN 数据包给主动方，表示它也准备好关闭这个连接了。数据包: 被动方 $\rightarrow$ 主动方 $[\text{FIN}, \text{Seq} = Y]$

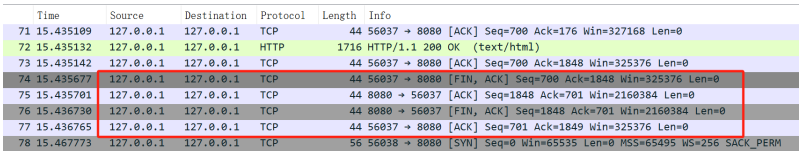
ACK (确认):

- 主动关闭方收到被动方的 FIN 数据包后，发送一个 ACK 数据包来确认。数据包: 主动方 $\rightarrow$ 被动方 $[\text{ACK}, \text{Seq} = X + 1, \text{Ack} = Y + 1]$

TCP 四次挥手介绍

从以下四个数据包中，可以看到：主动方首先发送了一个 FIN, ACK 数据包，表示它已完成数据发送并希望关闭连接；被动方收到这个数据包后，发送了一个 ACK 数据包来确认；被动方

随后也发送了一个 FIN, ACK 数据包，表示它也已完成数据发送并希望关闭连接；主动方收到这个数据包后，发送了一个 ACK 数据包来确认。



	Time	Source	Destination	Protocol	Length	Info
71	15.435109	127.0.0.1	127.0.0.1	TCP	44	56037 → 8080 [ACK] Seq=700 Ack=176 Win=327168 Len=0
72	15.435132	127.0.0.1	127.0.0.1	HTTP	1716	HTTP/1.1 200 OK (text/html)
73	15.435142	127.0.0.1	127.0.0.1	TCP	44	56037 → 8080 [ACK] Seq=700 Ack=1848 Win=325376 Len=0
74	15.435677	127.0.0.1	127.0.0.1	TCP	44	56037 → 8080 [FIN, ACK] Seq=700 Ack=1848 Win=325376 Len=0
75	15.435701	127.0.0.1	127.0.0.1	TCP	44	8080 → 56037 [ACK] Seq=1848 Ack=701 Win=2160384 Len=0
76	15.436730	127.0.0.1	127.0.0.1	TCP	44	8080 → 56037 [FIN, ACK] Seq=1848 Ack=701 Win=2160384 Len=0
77	15.436765	127.0.0.1	127.0.0.1	TCP	44	56037 → 8080 [ACK] Seq=701 Ack=1849 Win=325376 Len=0
78	15.467773	127.0.0.1	127.0.0.1	TCP	56	56038 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM

图 10

第 74 个数据包:

- 源地址: 127.0.0.1
- 标志位: [FIN, ACK] - 主动方希望结束连接，并确认之前接收到的数据。
- 序列号 (Seq): 700 - 这是主动方发送的最后一个字节的序列号。
- 确认号 (Ack): 1848 - 这是被动方发送的下一个预期字节的序列号。
- 窗口大小 (Win): 325376 - 这是主动方的接收窗口大小。

第 75 个数据包:

- 源地址: 127.0.0.1
- 标志位: [ACK] - 被动方确认已收到主动方的 FIN 请求。
- 序列号 (Seq): 1848 - 被动方发送的最后一个字节的序列号。
- 确认号 (Ack): 701 - 确认主动方的序列号 +1。
- 窗口大小 (Win): 2160384 - 这是被动方的接收窗口大小。

第 76 个数据包:

- 源地址: 127.0.0.1
- 标志位: [FIN, ACK] - 被动方也希望结束连接，并确认之前接收到的数据。
- 序列号 (Seq): 1848 - 被动方发送的最后一个字节的序列号。
- 确认号 (Ack): 701 - 确认主动方的序列号。
- 窗口大小 (Win): 2160384 - 这是被动方的接收窗口大小。

第 77 个数据包:

- 源地址: 127.0.0.1
- 标志位: [ACK] - 主动方确认已收到被动方的 FIN 请求。
- 序列号 (Seq): 701 - 确认被动方的序列号。
- 确认号 (Ack): 1849 - 确认被动方的序列号 +1。
- 窗口大小 (Win): 325376 - 这是主动方的接收窗口大小。

参考文献