南開大學
Nankai University

计算机学院

计算机网络实验报告

实验 2: 配置 Web 服务器，编写简单页面，分析交互过程

姓名：谢畅

学号：2113665

专业：计算机科学与技术

2023 年 11 月 3 日

目录

1 web 服务器搭建

1.1 概述

本次实验采取 django 框架实现 web 服务器的配置。Django 是一个基于 Python 的 Web 框架，它提供了一系列工具和库，用于快速开发高质量的 Web 应用程序。我们使用 Django 搭建 Web 服务器，编写 Python 代码，通过 Django 的路由、视图、模板等功能来实现 Web 应用程序的功能。

具体配置如下所示：

- web 框架: django 4.1
- 系统: windows11
- 浏览器与 web 服务器交互方式: 本地发出，本地接受，ip 均为本机地址。

1.2 搭建过程

1.2.1 相关命令作用

```
pip install Django
django-admin startproject myproject
python manage.py startapp myapp
python manage.py runserver
```

上面的命令大致是如下作用:

- 创建 Django 项目：在命令行中使用 django-admin 命令创建一个 Django 项目
- 创建 Django 应用程序：在命令行中使用 python manage.py 命令创建一个 Django 应用程序
- 启动 Django 服务器：在命令行中使用 python manage.py 命令启动 Django 服务器

1.2.2 搭建结果

在命令行输入 *python manage.py runserver*，成功启动服务器

```
(summer_learning) D:\大三上\计算机网络\作业及笔记\projects\lab2\xcBlog>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
November 01, 2023 - 21:57:19
Django version 4.1, using settings 'xcBlog.settings'
Starting development server at http://10.136.78.109:8080/
Quit the server with CTRL-BREAK.
```

图 1.1: 服务器启动

2 页面编写

在上述 django 框架下编写合适的处理函数，呈现 html 即可。由于这里我们只简单编写页面，直接在 views 里面编写如下代码即可 def display(request): return render(request, 'myweb.html')

```
def display(request):
    return render(request, 'myweb.html')
```

2.1 html 展示

在上述 django 框架下编写合适的处理函数，呈现 html 即可，这里仅展示 html 的编写。其中 {%load static %} 是 Django 模板语言中的一个标签，用于加载静态文件，包括图像、音频等文件，存放于 static 目录中。

```
{html}
<!doctype html>
{%load static %}
<html>
<head>
        <meta charset="utf-8">
        <title>my web</title>
        <link rel="shortcut icon" href="{% static 'bitbug_favicon.ico' %}"
        type="image/x-icon">
</head>
<body>
        <h1 style = "text-align: center" class="h1"> 个人主页 </h1>
        <p style="font-size: 20px; background-color:cadetblue;text-align: center">
```

```
14          开始 </p>
15          <div class="container">
16              <div class="info">
17              <p style="font-size: 20px"> 谢畅 </p>
18              <p style="font-size: 20px">2113665</p>
19              <p style="font-size: 20px"> 计算机科学与技术专业 </p>
20              </div>
21              <div class="logo img">
22              <p style="font-size: 20px" id="logo"> 本人 logo 图片如下: </p>
23              <img src="{% static 'logo1.jpg' %}" width="250" height="250" />
24              </div>
25              <br>
26              <p style="font-size: 20px" id="music"> 下面是自我介绍的音频 </p>
27              <audio controls="controls">
28              <source src="{% static 'introduction.m4a' %}" type="audio/mp3" />
29              </audio>
30          </div>
31          <div>
32              <p style="font-size: 20px; background-color:cadetblue;text-align: center">
33              终止 </p>
34          </div>
35  </body>
36  </html>
```

2.2 搭建结果

如下，显示个人主页的相关信息，有网页的图标、图片、音频、文字。

图 2.2: 页面展示效果

3 交互过程分析

在这里，我们主要针对三次握手、四次挥手、http 请求与响应进行分析。

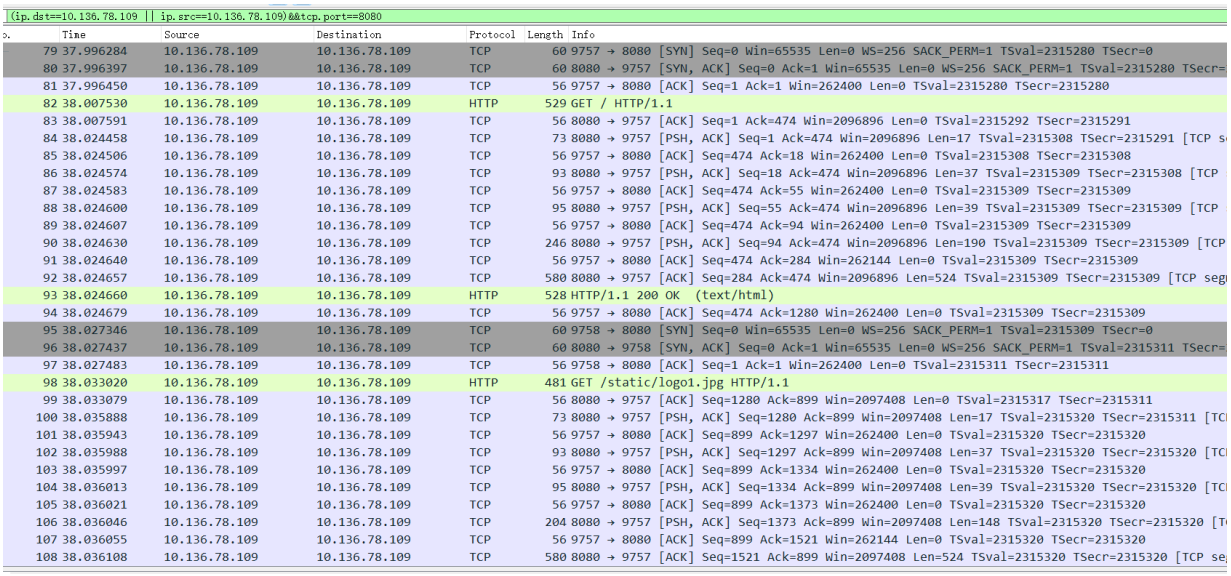
3.1 数据抓取

3.1.1 前期准备

- 采用 wireshark 抓包软件
- 采取 $Adapter\ \ for\ \ loopback\ \ traffic\ \ capture$ 接口进行抓取 (本地发出，本地接受)
- 查找本机 ipv4 地址，当前为 10.136.78.109
- 按照 ip 地址与网页端口号，设置过滤器

  (ip.dst==10.136.78.109 || ip.src==10.136.78.109)&&tcp.port==8080

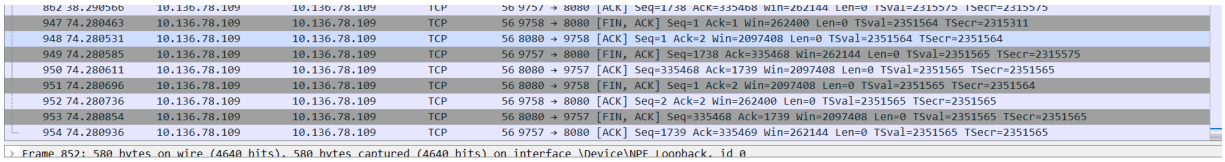
3.1.2 抓取的数据包



(ip.dst==10.136.78.109 || ip.src==10.136.78.109)&&tcp.port==8080

No.	Time	Source	Destination	Protocol	Length	Info
79	37.996284	10.136.78.109	10.136.78.109	TCP	60	9757 → 8080 [SYN] Seq=0 Win=65535 Len=0 WS=256 SACK_PERM=1 TSval=2315280 TSecr=0
80	37.996397	10.136.78.109	10.136.78.109	TCP	60	8080 → 9757 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 WS=256 SACK_PERM=1 TSval=2315280 TSecr=
81	37.996450	10.136.78.109	10.136.78.109	TCP	56	9757 → 8080 [ACK] Seq=1 Ack=1 Win=262400 Len=0 TSval=2315280 TSecr=2315280
82	38.007530	10.136.78.109	10.136.78.109	HTTP	529	GET / HTTP/1.1
83	38.007591	10.136.78.109	10.136.78.109	TCP	56	8080 → 9757 [ACK] Seq=1 Ack=474 Win=2096896 Len=0 TSval=2315292 TSecr=2315291
84	38.024458	10.136.78.109	10.136.78.109	TCP	73	8080 → 9757 [PSH, ACK] Seq=1 Ack=474 Win=2096896 Len=17 TSval=2315308 TSecr=2315291 [TCP s
85	38.024506	10.136.78.109	10.136.78.109	TCP	56	9757 → 8080 [ACK] Seq=474 Ack=18 Win=262400 Len=0 TSval=2315308 TSecr=2315308
86	38.024574	10.136.78.109	10.136.78.109	TCP	93	8080 → 9757 [PSH, ACK] Seq=18 Ack=474 Win=2096896 Len=37 TSval=2315309 TSecr=2315308 [TCP
87	38.024583	10.136.78.109	10.136.78.109	TCP	56	9757 → 8080 [ACK] Seq=474 Ack=55 Win=262400 Len=0 TSval=2315309 TSecr=2315309
88	38.024600	10.136.78.109	10.136.78.109	TCP	95	8080 → 9757 [PSH, ACK] Seq=55 Ack=474 Win=2096896 Len=39 TSval=2315309 TSecr=2315309 [TCP
89	38.024607	10.136.78.109	10.136.78.109	TCP	56	9757 → 8080 [ACK] Seq=474 Ack=94 Win=262400 Len=0 TSval=2315309 TSecr=2315309
90	38.024630	10.136.78.109	10.136.78.109	TCP	246	8080 → 9757 [PSH, ACK] Seq=94 Ack=474 Win=2096896 Len=190 TSval=2315309 TSecr=2315309 [TCP
91	38.024640	10.136.78.109	10.136.78.109	TCP	56	9757 → 8080 [ACK] Seq=474 Ack=284 Win=262144 Len=0 TSval=2315309 TSecr=2315309
92	38.024657	10.136.78.109	10.136.78.109	TCP	580	8080 → 9757 [ACK] Seq=284 Ack=474 Win=2096896 Len=524 TSval=2315309 TSecr=2315309 [TCP seg
93	38.024660	10.136.78.109	10.136.78.109	HTTP	528	HTTP/1.1 200 OK (text/html)
94	38.024679	10.136.78.109	10.136.78.109	TCP	56	9757 → 8080 [ACK] Seq=474 Ack=1280 Win=262400 Len=0 TSval=2315309 TSecr=2315309
95	38.027346	10.136.78.109	10.136.78.109	TCP	60	9758 → 8080 [SYN] Seq=0 Win=65535 Len=0 WS=256 SACK_PERM=1 TSval=2315309 TSecr=0
96	38.027437	10.136.78.109	10.136.78.109	TCP	60	8080 → 9758 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 WS=256 SACK_PERM=1 TSval=2315311 TSecr=
97	38.027483	10.136.78.109	10.136.78.109	TCP	56	9758 → 8080 [ACK] Seq=1 Ack=1 Win=262400 Len=0 TSval=2315311 TSecr=2315311
98	38.033020	10.136.78.109	10.136.78.109	HTTP	481	GET /static/logo1.jpg HTTP/1.1
99	38.033079	10.136.78.109	10.136.78.109	TCP	56	8080 → 9757 [ACK] Seq=1280 Ack=899 Win=2097408 Len=0 TSval=2315317 TSecr=2315311
100	38.035888	10.136.78.109	10.136.78.109	TCP	73	8080 → 9757 [PSH, ACK] Seq=1280 Ack=899 Win=2097408 Len=17 TSval=2315320 TSecr=2315311 [TC
101	38.035943	10.136.78.109	10.136.78.109	TCP	56	9757 → 8080 [ACK] Seq=899 Ack=1297 Win=262400 Len=0 TSval=2315320 TSecr=2315320
102	38.035988	10.136.78.109	10.136.78.109	TCP	93	8080 → 9757 [PSH, ACK] Seq=1297 Ack=899 Win=2097408 Len=37 TSval=2315320 TSecr=2315320 [TC
103	38.035997	10.136.78.109	10.136.78.109	TCP	56	9757 → 8080 [ACK] Seq=899 Ack=1334 Win=262400 Len=0 TSval=2315320 TSecr=2315320
104	38.036013	10.136.78.109	10.136.78.109	TCP	95	8080 → 9757 [PSH, ACK] Seq=1334 Ack=899 Win=2097408 Len=39 TSval=2315320 TSecr=2315320 [TC
105	38.036021	10.136.78.109	10.136.78.109	TCP	56	9757 → 8080 [ACK] Seq=899 Ack=1373 Win=262400 Len=0 TSval=2315320 TSecr=2315320
106	38.036046	10.136.78.109	10.136.78.109	TCP	204	8080 → 9757 [PSH, ACK] Seq=1373 Ack=899 Win=2097408 Len=148 TSval=2315320 TSecr=2315320 [T
107	38.036055	10.136.78.109	10.136.78.109	TCP	56	9757 → 8080 [ACK] Seq=899 Ack=1521 Win=262144 Len=0 TSval=2315320 TSecr=2315320
108	38.036108	10.136.78.109	10.136.78.109	TCP	580	8080 → 9757 [ACK] Seq=1521 Ack=899 Win=2097408 Len=524 TSval=2315320 TSecr=2315320 [TCP se

图 3.3: 数据包 1



947	74.280463	10.136.78.109	10.136.78.109	TCP	56	9758 → 8080 [FIN, ACK] Seq=1 Ack=1 Win=262400 Len=0 TSval=2351564 TSecr=2315311
948	74.280531	10.136.78.109	10.136.78.109	TCP	56	8080 → 9758 [ACK] Seq=1 Ack=2 Win=2097408 Len=0 TSval=2351564 TSecr=2351564
949	74.280585	10.136.78.109	10.136.78.109	TCP	56	9757 → 8080 [FIN, ACK] Seq=1738 Ack=335468 Win=262144 Len=0 TSval=2351565 TSecr=2315575
950	74.280611	10.136.78.109	10.136.78.109	TCP	56	8080 → 9757 [ACK] Seq=335468 Ack=1739 Win=2097408 Len=0 TSval=2351565 TSecr=2351565
951	74.280696	10.136.78.109	10.136.78.109	TCP	56	8080 → 9758 [FIN, ACK] Seq=1 Ack=2 Win=2097408 Len=0 TSval=2351565 TSecr=2351564
952	74.280736	10.136.78.109	10.136.78.109	TCP	56	9758 → 8080 [ACK] Seq=2 Ack=2 Win=262400 Len=0 TSval=2351565 TSecr=2351565
953	74.280854	10.136.78.109	10.136.78.109	TCP	56	8080 → 9757 [FIN, ACK] Seq=335468 Ack=1739 Win=2097408 Len=0 TSval=2351565 TSecr=2351565
954	74.280936	10.136.78.109	10.136.78.109	TCP	56	9757 → 8080 [ACK] Seq=1739 Ack=335469 Win=262144 Len=0 TSval=2351565 TSecr=2351565

Frame 852: 580 bytes on wire (4640 bits), 580 bytes captured (4640 bits) on interface \Device\NPF_Loopback, id 0

图 3.4: 数据包 2

可以看到上面有双端口与服务器进行连接，分别是 9757 端口与 9758 端口。它们分别完成三次握手的连接，http 请求与响应，四次挥手的过程，随即断开。

这里因为我们浏览器与 web 服务器的交互采用 HTTP/1.1。HTTP/1.1 可以使用双端口进行连接，这种方式被称为“管线化连接”（Pipelined Connection）。在管线化连接中，客户端可以在一个 TCP 连接上同时发送多个请求，而不需要等待每个请求的响应。服务器可以按照请求的顺序依次处理请求，并将响应返回给客户端。客户端可以通过不同的端口发送请求，以避免头阻塞的问题。

因此，在上面的数据抓取中有两个端口与服务器 (端口号为 8080) 进行了连接。

3.2 TCP 三次握手分析

为了防止已失效的连接请求报文突然又传送到了服务端，TCP 使用三次握手，确保 web 服务器端与浏览器客户端可以正常通信。

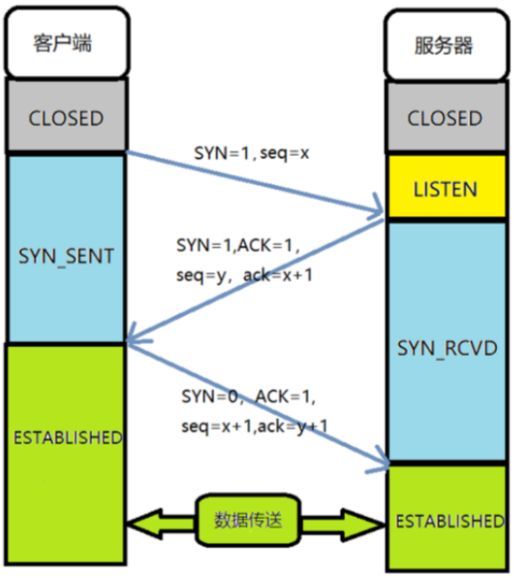
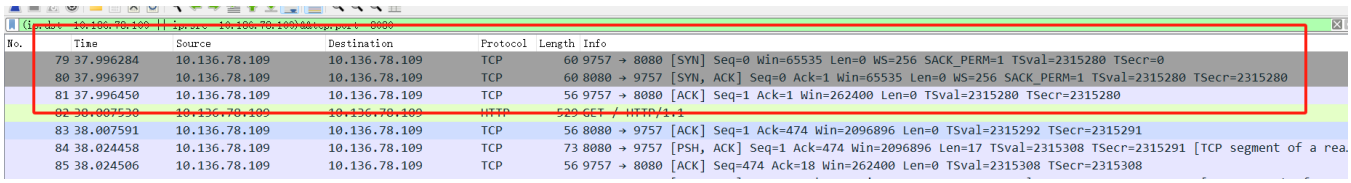


图 3.5: 三次握手原理图

下面我们以 9757 客户端端口与服务器 8080 端口进行示例，分析



No.	Time	Source	Destination	Protocol	Length	Info
79	37.996284	10.136.78.109	10.136.78.109	TCP	60	9757 → 8080 [SYN] Seq=0 Win=65535 Len=0 WS=256 SACK_PERM=1 TSval=2315280 TSecr=0
80	37.996397	10.136.78.109	10.136.78.109	TCP	60	8080 → 9757 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 WS=256 SACK_PERM=1 TSval=2315280 TSecr=2315280
81	37.996450	10.136.78.109	10.136.78.109	TCP	56	9757 → 8080 [ACK] Seq=1 Ack=1 Win=262400 Len=0 TSval=2315280 TSecr=2315280
82	38.007530	10.136.78.109	10.136.78.109	HTTP	529	GET / HTTP/1.1
83	38.007591	10.136.78.109	10.136.78.109	TCP	56	8080 → 9757 [ACK] Seq=1 Ack=474 Win=2096896 Len=0 TSval=2315292 TSecr=2315291
84	38.024458	10.136.78.109	10.136.78.109	TCP	73	8080 → 9757 [PSH, ACK] Seq=1 Ack=474 Win=2096896 Len=17 TSval=2315308 TSecr=2315291 [TCP segment of a rea
85	38.024506	10.136.78.109	10.136.78.109	TCP	56	9757 → 8080 [ACK] Seq=474 Ack=18 Win=262400 Len=0 TSval=2315308 TSecr=2315308

图 3.6: 实验分析: 三次握手

1. **第一次握手**: 客户端向服务器发送一个 SYN 报文段，报文段的首部中的标志位 SYN 置为 1，指明自己的初始化序号 seq=0，此时客户端处于 SYN_SENT 状态。

   如图3.7, 下面主要针对该分组的头部明细 (TCP 头部) 进行分析。

   - 可以看到源端口为 9757, 目的端口为 8080
   - Seq=0，客户端选择自己的初始序号 seq=0，数据包相对序列号从 0 开始，此时还没有发送数据。
   - acknowledge number=0。当接收方收到一个 TCP 数据包时，会将 ACK number 设置为发送方下一个期望接收的数据包的序列号。
   - ACK 未设置。ACK 是指 TCP 数据包中的确认标志位，用于确认接收到的数据包。
   - SYN 已设置，标志位，表示请求连接。
   - WIN=65535，告诉发送方自己的接收窗口大小为 65535

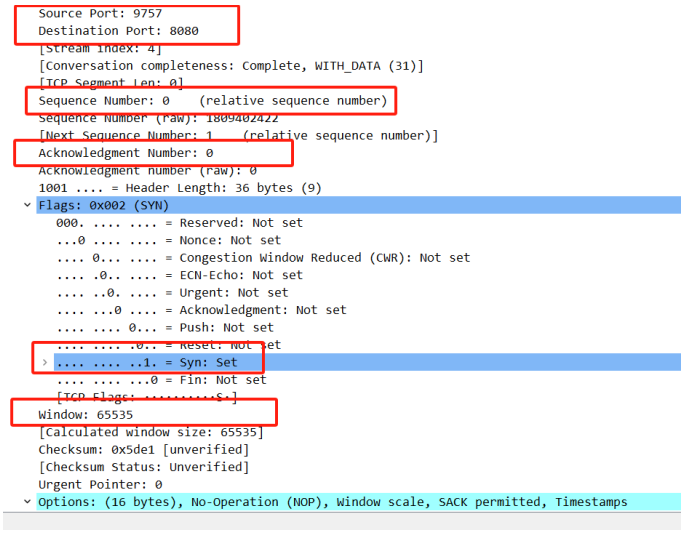


图 3.7: 第一次握手

服务器收到第一段报文后得出结论：客户端发送功能正常，服务器接收功能正常。

2. **第二次握手**：服务器收到 SYN 的报文段后，会以自己的 SYN+ACK 报文进行应答。该应答报文的首部有三个重要信息：首先 SYN、ACK 被置为 1；其次，确认号字段 ack=x+1；最后服务器选择自己的初始序号 seq=y。

   该报文段表明：“我收到了你发起建立连接的请求，该请求报文的初始序号是 x（确认号 ack=x+1 就表明了我收到了初始序号 seq=x 的报文），我同意建立该连接，我的初始序号是 y。”此时服务器处于 SYN_RCVD 状态。

   由于第一次握手客户端 seq=0，即 x=0，因此这里的 ack=1；这里服务器选择的 seq=0，即 y=0。

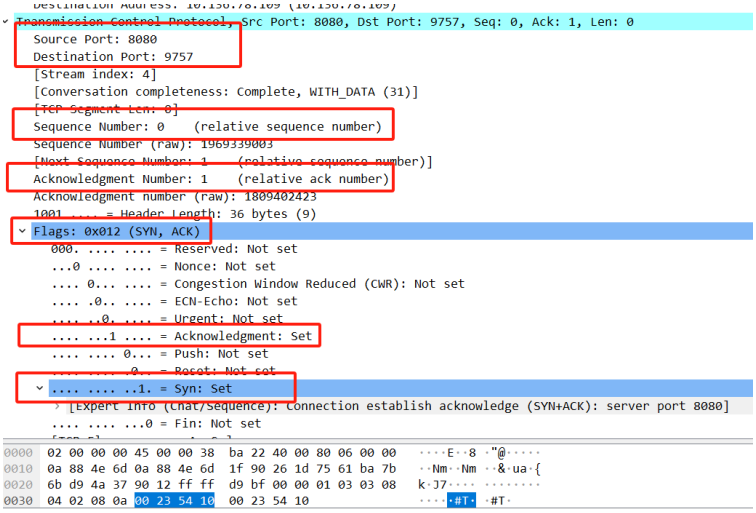


图 3.8: 第二次握手

- 可以看到源端口为 8080(服务器端)，目的端口为 9757(客户端)

- Seq=0，服务器选择自己的初始序号 seq=0，数据包相对序列号从 0 开始，此时还没有发送数据。
- acknowledge number=1，这里第一次握手客户端 seq=0，ack=0+1=1，表示下一个期望接收的数据包的序列号。
- ACK 已设置。确认接收到数据包。
- SYN 已设置，标志位，表示请求连接。

3. **第三次握手**: 客户端收到 SYN+ACK 报文后，会发送一个 ACK 报文段，该报文段中序号 seq=x+1，确认号 ack=y+1，表明我已经收到了你的确认。此时客户端处于 ESTABLISHED 状态。

   其中第一次握手 seq=x=0, 因此第三次握手 seq=x+1=1，第二次握手 seq=y=0，因此 ack=y+1=1。

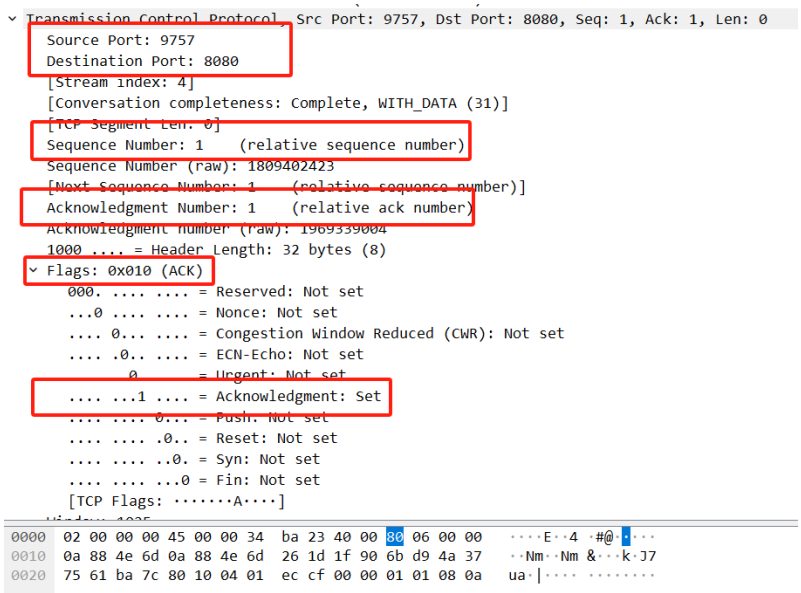


图 3.9: 第三次握手

- 可以看到源端口为 9757(客户端), 目的端口为 8080(服务端)
- Seq=1，第一次握手 seq=x=0, 因此第三次握手客户端 seq=x+1，表示实际已发送 1 个数据，与第二次握手服务器端的 ack number 匹配。
- acknowledge number=y+1=1，第二次握手服务器端的 seq=y=0。这里表示已经接收到第二次握手服务器端发送的一个数据，下次期望接受位置为 1 的数据。
- ACK 已设置。确认接收到数据包。
- SYN 已设置，标志位，表示请求连接。

3.3 http 报文与数据传输分析

3.3.1 http 报文介绍

下面是进行数据传输的相关 http 请求与响应。可以看到首先获取页面的 html，然后是 logo 图片，音频文件，最后是网页的图标.ico 文件。

总共有四轮 http 的请求与响应。

(ip.dst==10.136.78.109 || ip.src==10.136.78.109)&&tcp.port==8080&&http

No.	Time	Source	Destination	Protocol	Length	Info
82	38.007530	10.136.78.109	10.136.78.109	HTTP	529	GET / HTTP/1.1
93	38.024660	10.136.78.109	10.136.78.109	HTTP	528	HTTP/1.1 200 OK (text/html)
98	38.033020	10.136.78.109	10.136.78.109	HTTP	481	GET /static/logo1.jpg HTTP/1.1
231	38.037309	10.136.78.109	10.136.78.109	HTTP	416	HTTP/1.1 200 OK (JPEG JFIF image)
233	38.098900	10.136.78.109	10.136.78.109	HTTP	461	GET /static/introduction.m4a HTTP/1.1
840	38.109031	10.136.78.109	10.136.78.109	MP4	558	
842	38.287867	10.136.78.109	10.136.78.109	HTTP	490	GET /static/bitbug_favicon.ico HTTP/1.1
861	38.290558	10.136.78.109	10.136.78.109	HTTP	246	HTTP/1.1 200 OK (image/x-icon)

图 3.10: http 的请求与响应

1.*HTTP* 请求报文

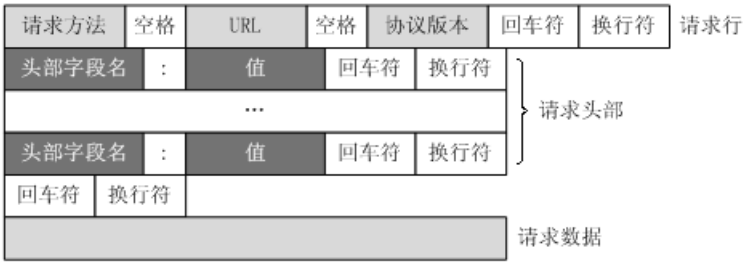


图 3.11: http 的请求报文

可以发现 *HTTP* 的请求由三部分组成: 请求行、请求头部、请求数据。

- 请求行: 用于描述客户端的请求方式 (GET/POST 等), 请求的资源名称 (URL) 以及使用的 HTTP 协议的版本号。它们用空格分隔。例如, GET /index.html HTTP/1.1。

  HTTP 协议的请求方法有 GET、POST、HEAD、PUT、DELETE、OPTIONS、TRACE、CONNECT。

- 请求头部：通知服务器有关于客户端请求的信息，典型的请求头有：

  User-Agent：产生请求的浏览器类型。

  Accept：客户端可识别的内容类型列表。

  Host：请求的主机名，允许多个域名同处一个 IP 地址，即虚拟主机。

  Connection：告诉服务器支持 keep-alive 特性

- 请求数据：当使用 POST 等方法时，通常需要客户端向服务器传递数据。这些数据就储存在请求正文中。

2.*HTTP* 响应报文

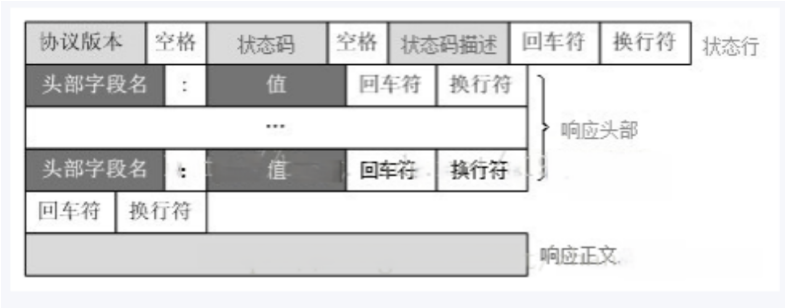


图 3.12: http 的响应报文

HTTP 响应报文主要由状态行、响应头部、空行以及响应数据组成

- 状态行：由 3 部分组成，分别为：协议版本，状态码，状态码描述。其中协议版本与请求报文一致，状态码描述是对状态码的简单描述。

  一些常见的状态码如下状态代码为 3 位数字。

  - 1xx：指示信息–表示请求已接收，继续处理。
  - 2xx：成功–表示请求已被成功接收、理解、接受。
  - 3xx：重定向–要完成请求必须进行更进一步的操作。
  - 4xx：客户端错误–请求有语法错误或请求无法实现。
  - 5xx：服务器端错误–服务器未能实现合法的请求。

- 响应头：描述服务器的基本信息，以及客户端如何处理数据
- 响应数据：服务器返回给客户端的数据

3.3.2 http 报文分析

1. 针对 http 请求进行分析

如下图3.13所示，我们针对 GET / HTTP/1.1 进行分析

- 请求行：请求方式为 GET，请求 URL：/ ，协议版本为 HTTP/1.1
- 请求头部：HOST、COnnection、User-Agent 等。
- 请求数据：无。

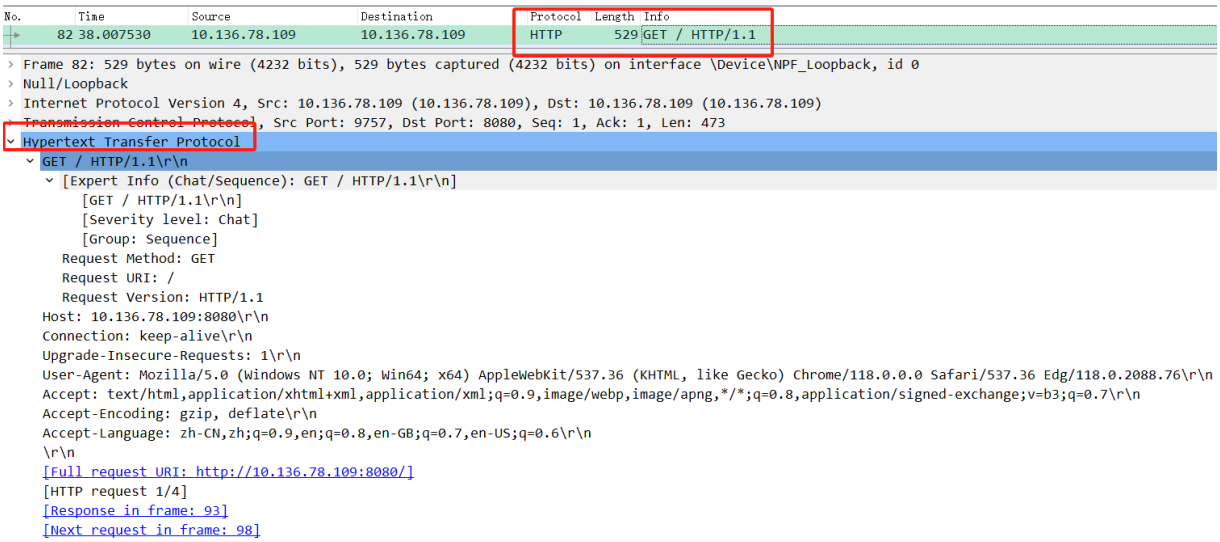


图 3.13: http 的请求示例

2. 针对 http 响应进行分析

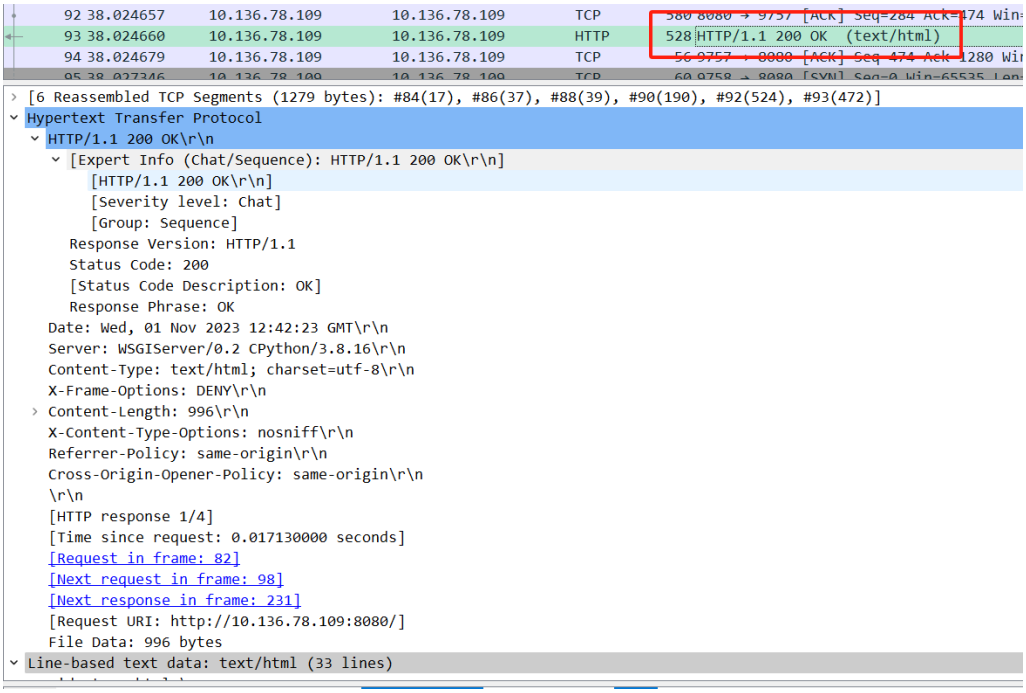


图 3.14: http 的响应示例

- 状态行：协议版本为 HTTP/1.1, 状态码为 200，说明已经被成功接受。
- 响应头：Date、Server、Content-type....
- 响应数据：Line-based text data。即接受的数据。如图3.15所示，我们得到了所编写网页的 html 代码

图 3.15: http 的响应数据展示

3.3.3 数据传输分析

在 HTTP 请求发送后，服务器需要一定的时间来处理请求并生成响应。在这个过程中，客户端和服务器之间会通过 TCP 连接进行数据传输，以确保数据的可靠传输。

对比如下的两个数据传输，可以发现在 http 响应时，出现 Reassembled TCP Segments 一栏。

它表示该 HTTP 响应的数据被分成了多个 TCP 数据包进行传输。以图3.16为示例，有 6 个 tcp 数据包，每个 TCP 数据包的大小分别为 17、37、39、190、524 和 472 字节。

由于 TCP 协议是一种面向连接的协议，它需要将数据分成多个数据包进行传输，以确保数据的可靠传输。在 HTTP 响应中，服务器会将处理结果封装成 HTTP 响应，并通过 TCP 连接发送给客户端。由于 HTTP 响应的数据可能比较大，因此需要将数据分成多个 TCP 数据包进行传输。

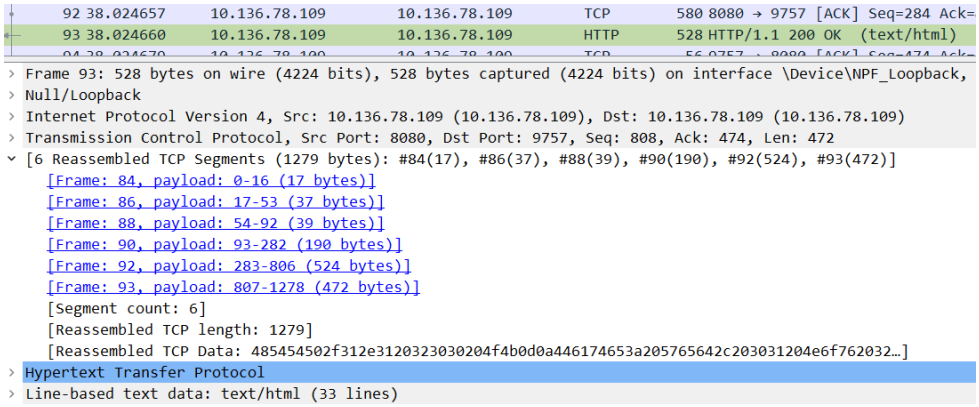


图 3.16: 获取 html 的数据传输

而在图3.17中，是关于音频的响应，可以发现由于音频文件较大，被分成了 536 个数据包，并在接收端进行了重组。

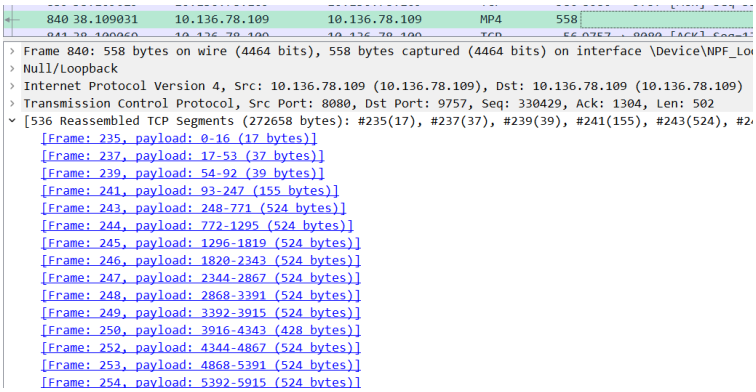


图 3.17: 获取音频 mp4.a 的数据传输

3.4 TCP 四次挥手分析

数据传输结束后，数据传输双方会使用四次挥手关闭连接。

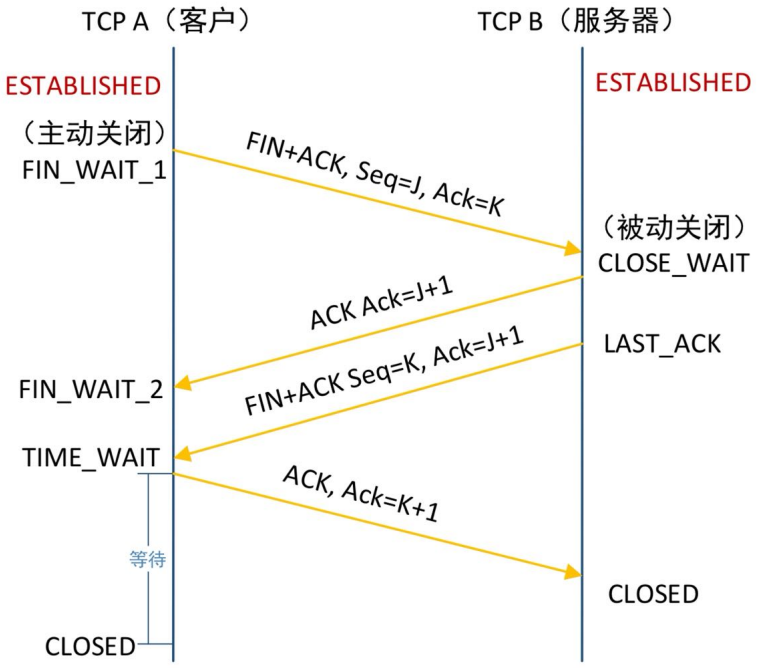


图 3.18: 四次挥手原理图

下面是实验中抓取的四次挥手过程，其中有两个端口 (9757、9758) 与服务器 (8080 端口) 完成四次挥手过程。其中以 9758 与 8080 之间的四次挥手作为分析示例。

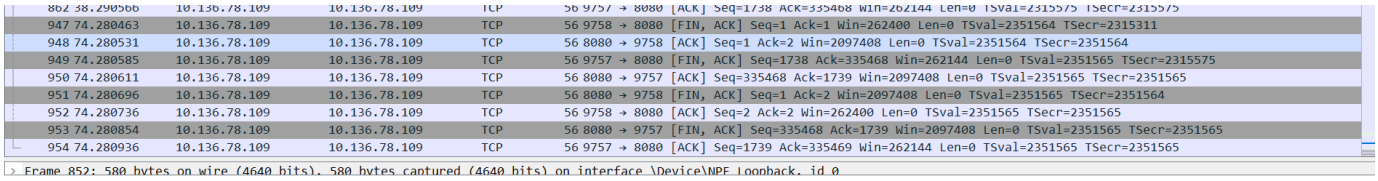


图 3.19: 四次挥手

1. **第一次挥手**：客户端发送一个 FIN+ACK 报文段，报文段中指定序号 seq=J。此时客户端处于 FIN_WAIT_1 状态。

   - 源端口为 9758(客户端), 目的端口为 8080(服务端)
   - Seq=1，即 J=1
   - acknowledge number=1，即 k=1。
   - FIN、ACK 进行设置

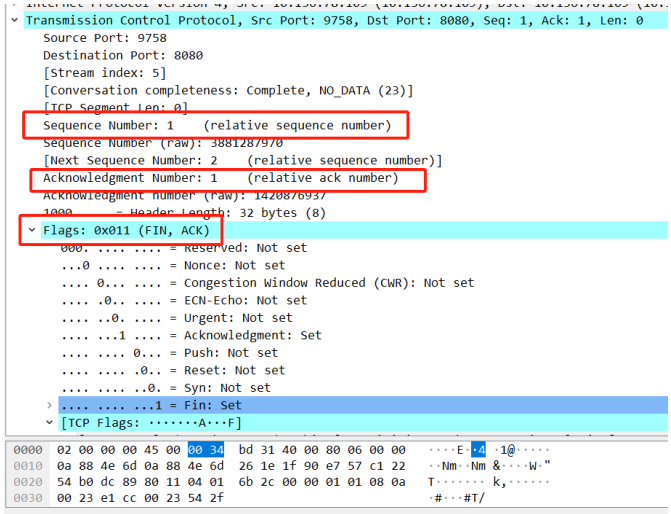


图 3.20: 第一次挥手

2. **第二次挥手**：服务器收到 FIN 报文后，立即发送一个 ACK 报文段，确认号为 ack=J+1。表明已经收到了客户端的报文。此时服务器处于 CLOSE_WAIT 状态。

   在第二次挥手和第三次挥手之间的时间段内，由于只是半关闭的状态，数据还是可以从服务器传送到客户端的。

   - 源端口为 8080(服务端), 目的端口为 9758(客户端)
   - acknowledge number=2, 即 ack=J+1=2
   - ACK 进行设置，表明已经收到了客户端的报文。

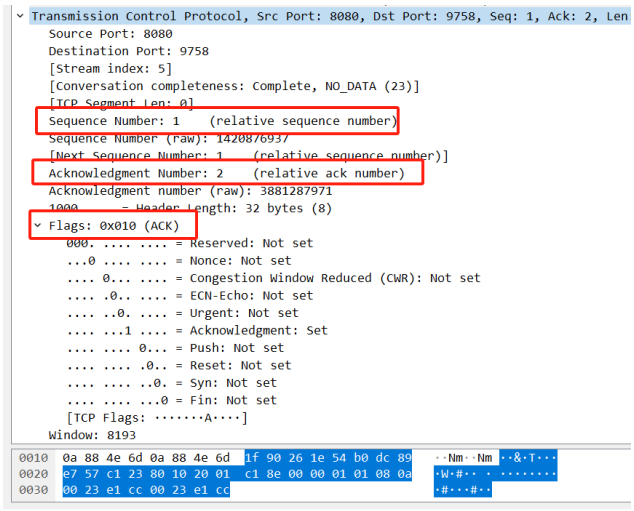


图 3.21: 第二次挥手

3. **第三次挥手**：如果数据传送完毕，服务器也想断开连接，那么就发送一个 FIN+ACK 报文，并重新指定一个序号 seq=K，确认号 ack=J+1，表明可以断开连接。

   - 源端口为 8080(服务端), 目的端口为 9758(客户端)
   - Seq=k=1。等于第一次挥手所发送的 ack 字段
   - acknowledge number=2, 即 ack=J+1=2
   - FIN+ACK 进行设置

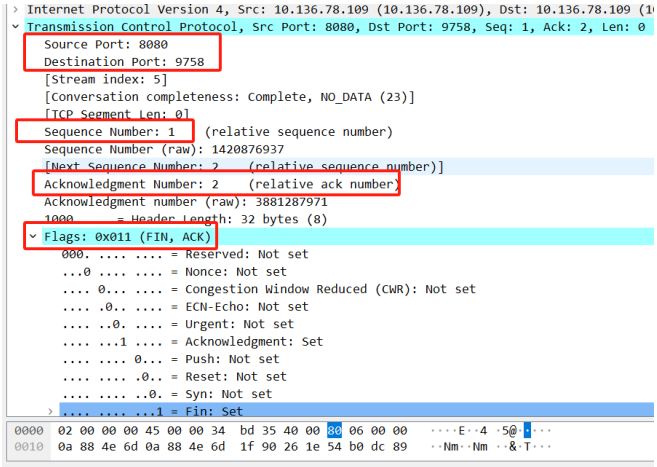


图 3.22: 第三次挥手

4. **第四次挥手**：客户端收到报文后，发出一个 ACK 报文段做出应答，确认号为 ack=K+1=2, seq=J+1=2。此时客户端处于 TIME_WAIT 状态，需要经过 2MSL 确保服务器收到自己的应答 ACK 报文后，才会进入 CLOSED 状态。

报文段最大生存时间 MSL，它是任何报文段被丢弃前在网络内的最长时间，超过这个时间报文将被丢弃。等待 2MSL 是为了确保服务器收到了最后一段 ACK 报文。

- 源端口为 9758(客户端), 目的端口为 8080(服务端)
- Seq=J+1=2。等于第三次挥手所发送的 ack 字段
- acknowledge number=2, 即 ack=K+1=2
- ACK 进行设置

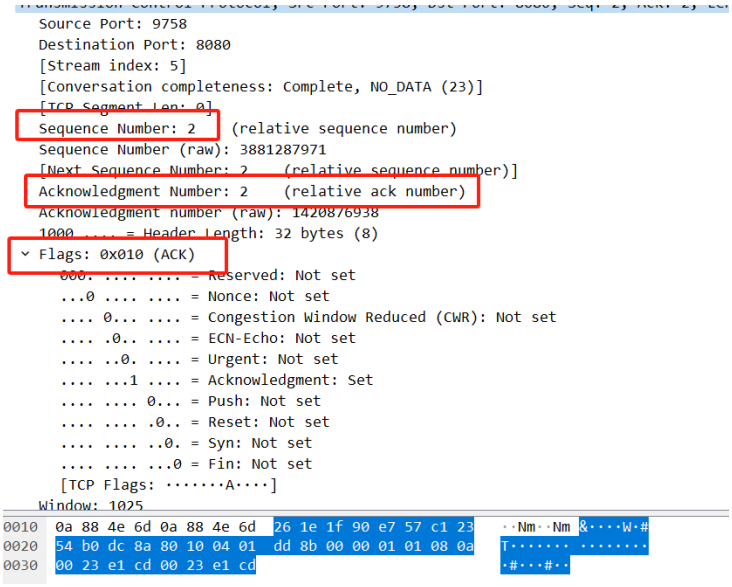


图 3.23: 第四次挥手

4 实验问题与思考

1. 在这次实验中，我发现了 http 请求与响应之间，数据传输是分成多个数据包传输的，即 Reassembled TCP Segments 。在之前的 http 响应之间我们也观察到了这一点。

在查阅资料后发现：

HTTP 响应中，服务器会将处理结果封装成 HTTP 响应，并通过 TCP 连接发送给客户端。由于 HTTP 响应的数据可能比较大，因此需要将数据分成多个 TCP 数据包进行传输。在 Wireshark 抓包工具中，如果一个 TCP 数据包的大小超过了 MTU（最大传输单元），则会被分成多个 TCP 数据包进行传输，并在每个 TCP 数据包的前面添加一个序号，以便在接收端进行重组。

2. 当多次请求且页面没有修改时，http 的响应中会显示 Not Modified，如图4.24

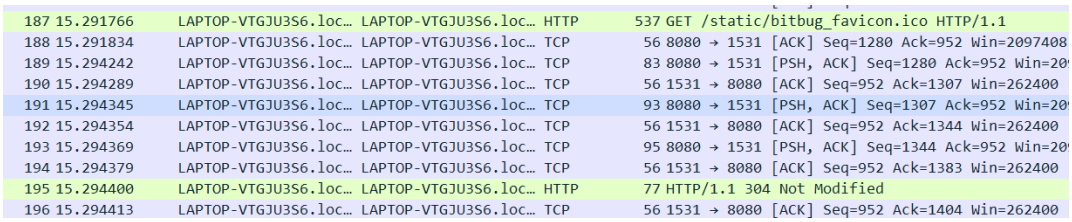


图 4.24: not modified 相关

其中图4.25的状态码 304 表示页面未修改, 这表示其可以直接使用浏览器缓存的内容。因此在捕获实验前，应该先清除缓存，再进行相关实验。

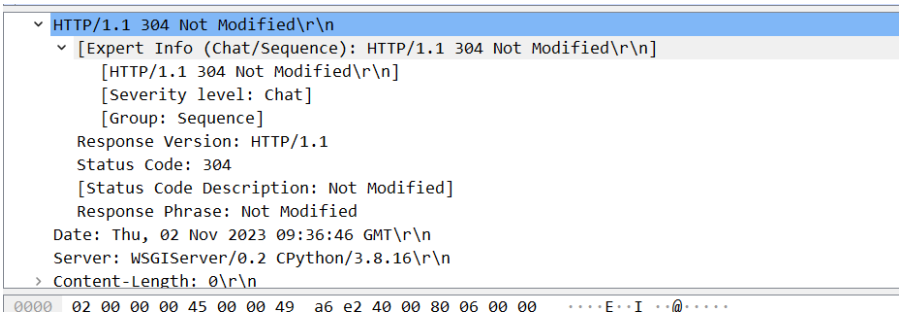


图 4.25: 304 状态码

3. 在这次实验中，如果服务器 (8080 端口) 停止服务后，浏览器并未退出，可以发现图4.26中的分组。

查阅相关资料后，发现：

RST 代表 TCP 连接复位，它是 TCP 协议中的一种控制报文。当一个 TCP 连接出现异常情况时，例如连接超时或者连接被非法关闭，TCP 会发送一个 RST 报文来终止连接。在 TCP 中，[RST,ACK] 表示收到了对方的复位请求，并且对方已经关闭了连接。

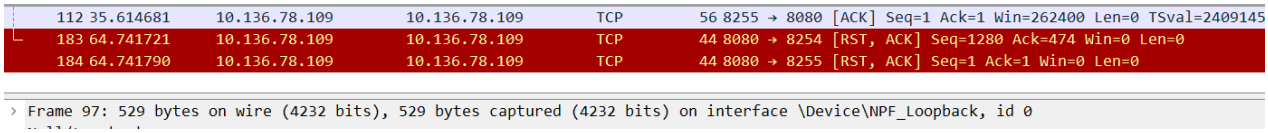


图 4.26: RST 相关