



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 服务及分析 HTTP  
交互过程

姓名：刘浩泽

学号：2212478

专业：计算机科学与技术

班级：计算机卓越班

2024 年 11 月 1 日

## 目录

<b>1 实验要求</b>	<b>2</b>
<b>2 搭建 Web 服务器</b>	<b>2</b>
<b>3 编写 HTML 文件并访问 Web 服务器</b>	<b>3</b>
<b>4 使用 Wireshark 捕获与 Web 服务器的交互过程，详细说明 HTTP 交互过程。</b>	<b>6</b>
4.1 WireShark 设置 . . . . .	6
4.2 HTTP 交互过程 . . . . .	7
4.2.1 当设置浏览器退出清除缓存时 . . . . .	8
4.2.2 当设置浏览器退出保留缓存时 . . . . .	12
<b>5 实验内容总结</b>	<b>13</b>

## 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。
- (2) 通过浏览器访问 Web 服务器, 获取自己编写的 HTML 文档, 并显示 Web 页面。
- (3) 在获取 Web 页面的同时, 使用 Wireshark 捕获与 Web 服务器的交互过程, 设置过滤器使 Wireshark 仅显示 HTTP 报文, 并详细说明 HTTP 交互过程。
- (4) 现场演示。
- (5) 提交 HTML 文档、Wireshark 捕获文件和实验报告。

## 2 搭建 Web 服务器

在本次实验中, 我选择在 Windows 系统上使用 Node.js 搭建 Web 服务器。以下是具体的搭建步骤:

- 1. 首先, 安装 Node.js 并初始化一个 Node.js 项目。在命令行中创建新的项目目录, 并使用 `npm init -y` 命令初始化一个新的 Node.js 项目, 生成 `package.json` 文件:

```
mkdir my-web-server cd my-web-server npm init -y
```

- 2. 接下来安装 Express 框架:

```
npm install express
```

Express 是一个基于 Node.js 平台的 Web 应用程序框架, 它提供了一系列强大的特性, 可以快速地构建 Web 应用程序。通过这个命令, 我们将 Express 添加为项目的依赖项, 并将其安装到项目中。

- 3. 在项目目录下创建 `server.js` 文件, 这个文件包含 Web 服务器的代码。在这个文件中, 我们使用 Express 框架来创建 Web 服务器, 并设置静态文件目录和路由。代码如下所示:

---

```
1 const express = require('express');
2 const path = require('path');
3 const app = express();
4 const PORT = 8888;
5 // 设置静态文件目录
6 // 这个中间件允许 Express 从 'public' 目录中提供静态资源, 如 HTML、CSS、JavaScript 和图像文件
7 app.use(express.static('public'));
8 // 创建路由
9 // 当客户端访问根路径 '/' 时, 服务器将响应 public 目录下的 index.html 文件
10 app.get('/', (req, res) => {
11     res.sendFile(path.join(__dirname, 'public', 'index.html'));
12 });
13 // 启动服务器
```

```
14 // 服务器将在端口 8888 上监听请求，并在控制台打印出服务器的运行地址
15 app.listen(PORT, () => {
16     console.log(`Server is running at http://localhost:${PORT}`);
17 });
```

---

具体来说，在 server.js 文件中：

我们首先引入 express 和 path 模块。express 模块提供了创建 Web 服务器的功能，path 模块则用于处理文件路径。

接着创建一个 Express 应用实例 app，并将监听端口设置为 8888。

通过 app.use(express.static('public')) 这行代码，我们将 public 目录设置为静态文件目录。这意味着客户端可以直接访问 public 目录中的 HTML、CSS、JavaScript 和图像等资源文件。

在 app.get('/', (req, res) => { ... }) 中，我们创建了一个根路由 /。当客户端访问根路径时，服务器将响应 public 目录下的 index.html 文件。

最后，我们使用 app.listen(PORT, () => { ... }) 启动服务器，并在控制台打印出服务器的运行地址。

完成以上三个步骤后，Web 服务器就搭建完毕了。接下来，我们可以通过以下步骤进行验证和访问：

(1) 在命令行中执行 node server.js 命令，启动 Web 服务器。控制台输出将显示服务器正在 http://localhost:8888 上运行。

(2) 打开浏览器，在地址栏输入 http://localhost:8888。此时，浏览器将显示 public 目录下的 index.html 文件，也就是我们编写的 HTML 文档。

(3) 我们可以继续在 public 目录下添加更多的 HTML、CSS 和 JavaScript 文件，并通过 http://localhost:8888 访问它们，从而在浏览器中预览和测试我们的 Web 应用程序。

### 3 编写 HTML 文件并访问 Web 服务器

Web 页面要求包含文本信息（至少包含专业、学号、姓名）、自己的 LOGO、自我介绍的音频。HTML 代码如下所示：

---

```
1 <!DOCTYPE html>
2 <html lang="zh">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title> 我的个人网页 </title>
7     <link rel="icon" href="logo.png" type="image/png">
8     <!-- <link rel="stylesheet" href="styles.css" -->
9     <style>
10         body {
11             display: flex;
12             justify-content: center;
```

```
13         align-items: center;
14         height: 100vh;
15         margin: 0;
16         background-color: #f0f4f8;
17         font-family: 'Arial', sans-serif;
18     }
19     .container {
20         text-align: center; /* 确保文本内容居中 */
21         background-color: #ffffff;
22         padding: 40px;
23         border-radius: 10px;
24         box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
25         width: 80%;
26         max-width: 600px;
27         transition: transform 0.3s ease-in-out;
28     }
29     .container:hover {
30         transform: translateY(-5px);
31         box-shadow: 0 8px 20px rgba(0, 0, 0, 0.15);
32     }
33     .logo {
34         width: 150px;
35         height: auto;
36         border: 2px solid #00796b; /* 添加边框 */
37         border-radius: 10px;
38         margin-bottom: 20px;
39     }
40     header h1 {
41         color: #00796b;
42     }
43     footer {
44         margin-top: 20px;
45         color: #666;
46     }
47     .container p {
48         margin: 10px 0;
49     }
50     .container h2 {
51         color: #00796b;
52         margin-top: 20px;
53     }
54     .container audio {
```

```
55         width: 100%;
56         max-width: 400px;
57         margin: 20px auto;
58     }
59     .github-link {
60         display: inline-block;
61         margin-top: 20px;
62         color: #00796b;
63         text-decoration: none;
64         border-bottom: 1px solid #00796b;
65         padding-bottom: 2px;
66         transition: color 0.3s ease-in-out;
67     }
68     .github-link:hover {
69         color: #005c50;
70         border-bottom-color: #005c50;
71     }
72 </style>
73 </head>
74 <body>
75     <div class="container">
76         <header>
77             
78             <h1> 欢迎来到我的个人网页 </h1>
79             <p> 这是一个展示个人信息和音频介绍的网页。</p>
80         </header>
81         <main>
82             <p><strong> 专业: </strong> 计算机科学与技术 </p>
83             <p><strong> 学号: </strong>2212478</p>
84             <p><strong> 姓名: </strong> 刘浩泽 </p>
85             <h2> 自我介绍音频 </h2>
86             <audio controls>
87                 <source src="introduction.mp3" type="audio/mpeg">
88                 您的浏览器不支持音频元素。
89             </audio>
90         </main>
91         <footer>
92             <a href="https://github.com/lhz191" class="github-link"> 访问我的 GitHub</a>
93             <p> 版权所有 &copy; 2212478 刘浩泽 </p>
94         </footer>
95     </div>
96 </body>
```

97 </html>

Web 页面效果如下:



图 3.1: Web 页面展示

从图中可以看到, 通过访问 `http://localhost:8888`, 用户可以在浏览器中查看并交互这个个人网页, 编写的 HTML 文件在 Web 服务器上成功展示了。页面包含了个人 Logo、基本信息和自我介绍音频, 整体布局和样式也符合预期要求。这说明 Web 服务器搭建正常, 能够正确地托管和渲染 HTML 页面内容。

## 4 使用 Wireshark 捕获与 Web 服务器的交互过程, 详细说明 HTTP 交互过程。

### 4.1 WireShark 设置

首先, 我们需要获取本机的 IP 地址。在命令行中输入 `ipconfig` 命令, 可以看到本机 IP 地址为 `192.168.188.1`。

接下来, 打开 Wireshark 并选择捕获过滤器。由于我们需要捕获本机与 Web 服务器之间的交互, 因此选择”Adapter for loopback traffic capture” 过滤器。

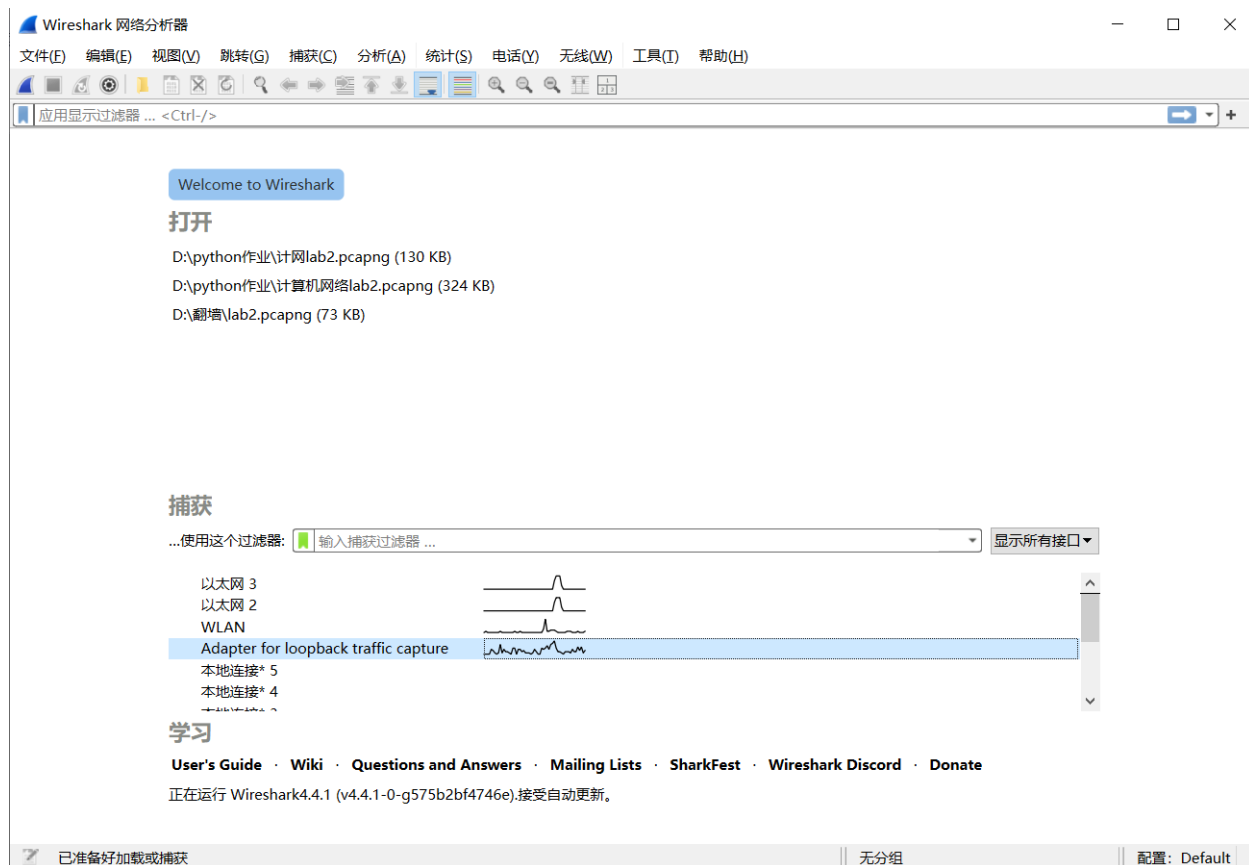


图 4.2: Wireshark 选择捕获过滤器

然后, 我们需要设置过滤条件。根据前面获取的 IP 地址和 Web 服务器监听的端口号 8888, 设置过滤条件为: `ip.addr == 192.168.188.1 && tcp.port == 8888`。这样就可以仅捕获本机与 Web 服务器之间的 TCP 流量。

## 4.2 HTTP 交互过程

通过上述 Wireshark 设置, 我们可以捕获到客户端 (本机) 与 Web 服务器之间的 HTTP 交互过程。如图 4.3 所示。



正在捕获 Adapter for loopback traffic capture

文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(I) 帮助(H)

ip.addr == 192.168.188.1 && tcp.port == 8888

No.	Time	Source	Destination	Protocol	Length	Info
1348	7.269117	192.168.188.1	192.168.188.1	TCP	64	58851 → 8888 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM TSval=11679218 TSecr=0
1349	7.269194	192.168.188.1	192.168.188.1	TCP	64	8888 → 58851 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM TSval=11679218 TSecr=11679218
1350	7.269235	192.168.188.1	192.168.188.1	TCP	56	58851 → 8888 [ACK] Seq=1 Ack=1 Win=2619136 Len=0 TSval=11679218 TSecr=11679218
1355	7.287763	192.168.188.1	192.168.188.1	HTTP	586	GET / HTTP/1.1
1356	7.287820	192.168.188.1	192.168.188.1	TCP	56	8888 → 58851 [ACK] Seq=1 Ack=531 Win=2619136 Len=0 TSval=11679236 TSecr=11679236
1358	7.289562	192.168.188.1	192.168.188.1	HTTP	321	HTTP/1.1 304 Not Modified
1359	7.289613	192.168.188.1	192.168.188.1	TCP	56	58851 → 8888 [ACK] Seq=531 Ack=266 Win=2618880 Len=0 TSval=11679238 TSecr=11679238
1377	7.312052	192.168.188.1	192.168.188.1	TCP	64	58852 → 8888 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM TSval=11679261 TSecr=0
1378	7.312170	192.168.188.1	192.168.188.1	TCP	64	8888 → 58852 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM TSval=11679261 TSecr=11679261
1379	7.312217	192.168.188.1	192.168.188.1	TCP	56	58852 → 8888 [ACK] Seq=1 Ack=1 Win=2619136 Len=0 TSval=11679261 TSecr=11679261
1401	7.398320	192.168.188.1	192.168.188.1	HTTP	546	GET /introduction.mp3 HTTP/1.1
1402	7.398395	192.168.188.1	192.168.188.1	TCP	56	8888 → 58851 [ACK] Seq=266 Ack=1021 Win=2618624 Len=0 TSval=11679347 TSecr=11679347
1403	7.400047	192.168.188.1	192.168.188.1	HTTP	324	HTTP/1.1 304 Not Modified
1404	7.400117	192.168.188.1	192.168.188.1	TCP	56	58851 → 8888 [ACK] Seq=1021 Ack=534 Win=2618624 Len=0 TSval=11679349 TSecr=11679349
1595	12.334864	192.168.188.1	192.168.188.1	TCP	56	58852 → 8888 [FIN, ACK] Seq=1 Ack=1 Win=2619136 Len=0 TSval=11684283 TSecr=11679261
1596	12.334896	192.168.188.1	192.168.188.1	TCP	56	8888 → 58852 [ACK] Seq=1 Ack=2 Win=2619136 Len=0 TSval=11684283 TSecr=11684283
1597	12.334947	192.168.188.1	192.168.188.1	TCP	56	58851 → 8888 [FIN, ACK] Seq=1021 Ack=534 Win=2618624 Len=0 TSval=11684283 TSecr=11679349
1598	12.334966	192.168.188.1	192.168.188.1	TCP	56	8888 → 58851 [ACK] Seq=534 Ack=1022 Win=2618624 Len=0 TSval=11684284 TSecr=11684283
1607	12.335368	192.168.188.1	192.168.188.1	TCP	56	8888 → 58852 [FIN, ACK] Seq=1 Ack=2 Win=2619136 Len=0 TSval=11684284 TSecr=11684283
1608	12.335403	192.168.188.1	192.168.188.1	TCP	56	58852 → 8888 [ACK] Seq=2 Ack=2 Win=2619136 Len=0 TSval=11684284 TSecr=11684284
1611	12.335809	192.168.188.1	192.168.188.1	TCP	56	8888 → 58851 [FIN, ACK] Seq=534 Ack=1022 Win=2618624 Len=0 TSval=11684284 TSecr=11684283
1612	12.335844	192.168.188.1	192.168.188.1	TCP	56	58851 → 8888 [ACK] Seq=1022 Ack=535 Win=2618624 Len=0 TSval=11684284 TSecr=11684284

> Frame 1404: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface {Device\NPF\_{Loopback}, Null/Loopback

> Internet Protocol Version 4, Src: 192.168.188.1, Dst: 192.168.188.1

> Transmission Control Protocol, Src Port: 58851, Dst Port: 8888, Seq: 1021, Ack: 534, Len: 0

0000 02 00 00 00 45 00 00 34 e7 75 04 00 00 06 00 00 .....E..4..u@.@...  
0010 c0 a8 bc 01 c0 a8 bc 01 e5 e3 22 b8 09 5d 33 ad .....[3]..  
0020 0a 81 0d 4e 80 10 27 f5 b9 b0 00 00 01 01 08 0a .....N.....  
0030 00 b2 36 75 00 b2 36 75 .....6u..6u

图 4.3: HTTP 交互过程

下面我们分两种情况详细分析一下 HTTP 的交互过程。

#### 4.2.1 当设置浏览器退出清除缓存时

在这种情况下, 由于客户端的浏览器已清除了缓存, 因此访问 Web 页面所需的资源都需要重新从服务端获取, 无法使用之前缓存的内容。由图 4.4 我们可以观察到完整的 TCP 连接建立过程 (三次握手)。

##### (1) TCP 建立连接: 三次握手

当设置浏览器退出清除缓存时.pcapng

文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(I) 帮助(H)

ip.addr == 192.168.188.1 && tcp.port == 8888

No.	Time	Source	Destination	Protocol	Length	Info
439	6.546773	192.168.188.1	192.168.188.1	TCP	64	60945 → 8888 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM TSval=13230590 TSecr=0
440	6.546830	192.168.188.1	192.168.188.1	TCP	64	8888 → 60945 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM TSval=13230591 TSecr=13230590
441	6.546880	192.168.188.1	192.168.188.1	TCP	56	60945 → 8888 [ACK] Seq=1 Ack=1 Win=2618880 Len=0 TSval=13230591 TSecr=13230591
442	6.560093	192.168.188.1	192.168.188.1	HTTP	536	GET / HTTP/1.1
443	6.560156	192.168.188.1	192.168.188.1	TCP	56	8888 → 60945 [ACK] Seq=1 Ack=481 Win=2619136 Len=0 TSval=13230604 TSecr=13230604
444	6.562091	192.168.188.1	192.168.188.1	HTTP	3486	HTTP/1.1 200 OK (text/html)
445	6.562196	192.168.188.1	192.168.188.1	TCP	56	60945 → 8888 [ACK] Seq=481 Ack=3431 Win=2619136 Len=0 TSval=13230606 TSecr=13230606
450	6.579391	192.168.188.1	192.168.188.1	TCP	64	60946 → 8888 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM TSval=13230623 TSecr=0
451	6.579499	192.168.188.1	192.168.188.1	TCP	64	8888 → 60946 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM TSval=13230623 TSecr=13230623
452	6.579551	192.168.188.1	192.168.188.1	TCP	56	60946 → 8888 [ACK] Seq=1 Ack=1 Win=2619136 Len=0 TSval=13230623 TSecr=13230623
453	6.581282	192.168.188.1	192.168.188.1	HTTP	480	GET /logo.png HTTP/1.1
454	6.581319	192.168.188.1	192.168.188.1	TCP	56	8888 → 60945 [ACK] Seq=3431 Ack=905 Win=2618880 Len=0 TSval=13230625 TSecr=13230625
455	6.582704	192.168.188.1	192.168.188.1	TCP	65539	8888 → 60945 [ACK] Seq=3431 Ack=905 Win=2618880 Len=65483 TSval=13230626 TSecr=13230625 [TCP PDU reassembled in 458]
456	6.582723	192.168.188.1	192.168.188.1	TCP	414	8888 → 60945 [PSH, ACK] Seq=68914 Ack=905 Win=2618880 Len=358 TSval=13230626 TSecr=13230625 [TCP PDU reassembled in 458]
457	6.582781	192.168.188.1	192.168.188.1	TCP	56	60945 → 8888 [ACK] Seq=905 Ack=69272 Win=2619136 Len=0 TSval=13230626 TSecr=13230626
458	6.582941	192.168.188.1	192.168.188.1	HTTP	21874	HTTP/1.1 200 OK (JPEG JFIF image)
459	6.582968	192.168.188.1	192.168.188.1	TCP	56	60945 → 8888 [ACK] Seq=905 Ack=91090 Win=2597376 Len=0 TSval=13230627 TSecr=13230627
472	6.645188	192.168.188.1	192.168.188.1	HTTP	450	GET /introduction.mp3 HTTP/1.1
473	6.645249	192.168.188.1	192.168.188.1	TCP	56	8888 → 60945 [ACK] Seq=91090 Ack=1299 Win=2618368 Len=0 TSval=13230689 TSecr=13230689
474	6.646794	192.168.188.1	192.168.188.1	TCP	65539	8888 → 60945 [ACK] Seq=91090 Ack=1299 Win=2618368 Len=65483 TSval=13230690 TSecr=13230689
475	6.646815	192.168.188.1	192.168.188.1	TCP	471	8888 → 60945 [PSH, ACK] Seq=156573 Ack=1299 Win=2618368 Len=415 TSval=13230690 TSecr=13230689
476	6.646868	192.168.188.1	192.168.188.1	TCP	56	60945 → 8888 [ACK] Seq=1299 Ack=156988 Win=2619136 Len=0 TSval=13230691 TSecr=13230690
477	6.647083	192.168.188.1	192.168.188.1	TCP	65539	8888 → 60945 [ACK] Seq=156988 Ack=1299 Win=2618368 Len=65483 TSval=13230691 TSecr=13230691
478	6.647102	192.168.188.1	192.168.188.1	TCP	109	8888 → 60945 [PSH, ACK] Seq=222471 Ack=1299 Win=2618368 Len=53 TSval=13230691 TSecr=13230691
479	6.647144	192.168.188.1	192.168.188.1	TCP	56	60945 → 8888 [ACK] Seq=1299 Ack=222524 Win=2619136 Len=0 TSval=13230691 TSecr=13230691
480	6.647357	192.168.188.1	192.168.188.1	TCP	65539	8888 → 60945 [ACK] Seq=222524 Ack=1299 Win=2618368 Len=65483 TSval=13230691 TSecr=13230691
481	6.647395	192.168.188.1	192.168.188.1	TCP	109	8888 → 60945 [PSH, ACK] Seq=288007 Ack=1299 Win=2618368 Len=53 TSval=13230691 TSecr=13230691
482	6.647488	192.168.188.1	192.168.188.1	TCP	56	60945 → 8888 [ACK] Seq=1299 Ack=288060 Win=2553600 Len=0 TSval=13230691 TSecr=13230691

> Frame 684: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface {Device\NPF\_{Loopback}, Null/Loopback

> Internet Protocol Version 4, Src: 192.168.188.1, Dst: 192.168.188.1

> Transmission Control Protocol, Src Port: 60945, Dst Port: 8888, Seq: 1300, Ack: 4482364, Len: 0

0000 02 00 00 00 45 00 00 28 ea 2e 40 00 00 06 00 00 .....E...(@.@...  
0010 c0 a8 bc 01 c0 a8 bc 01 ee 11 22 b8 4c 5d 62 cc .....[L]b..  
0020 24 1d bc 5b 50 14 00 00 16 10 00 00 .....[P].....

图 4.4: TCP 建立过程: 三次握手

- 客户端首先发起 TCP 连接请求, 向服务器的 8888 端口发送 SYN 报文 (Packet 439)。这个 SYN 报文的序号为 0, 窗口大小为 65535, 并且包含了一些 TCP 选项, 如 MSS、窗口缩放等。

```
> Frame 439: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface \Device\NPF_{...}, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 192.168.188.1, Dst: 192.168.188.1
> Transmission Control Protocol, Src Port: 60945, Dst Port: 8888, Seq: 0, Len: 0
  Source Port: 60945
  Destination Port: 8888
  [Stream index: 19]
  [Stream Packet Number: 1]
  [Conversation completeness: Complete, WITH_DATA (63)]
  [TCP Segment Len: 0]
  Sequence Number: 0 (relative sequence number)
  Sequence Number (raw): 1281187256
  [Next Sequence Number: 1 (relative sequence number)]
  Acknowledgment Number: 0
  Acknowledgment number (raw): 0
  1010 .... = Header Length: 40 bytes (10)
  Flags: 0x002 (SYN)
    000. .... = Reserved: Not set
    ...0 .... = Accurate ECN: Not set
    ....0... = Congestion Window Reduced: Not set
    ....0.. = ECN-Echo: Not set
    .....0.. = Urgent: Not set
    ....0... = Acknowledgment: Not set
    .....0... = Push: Not set
    .....0... = Reset: Not set
    ....1... = Syn: Set
    .....0 = Fin: Not set
  [TCP Flags: .....S.]
```

图 4.5: 第一次握手

可以看到第一次握手客户端发送一个带有 SYN 标志位的报文给服务器, 这个报文表示客户端请求建立连接。并且报文序列号为 0。

- 服务器收到客户端的 SYN 报文后, 回复 SYN-ACK 报文 (Packet 440)。这个 SYN-ACK 报文的序号为 0, 确认号为 1, 窗口大小也为 65535, 同时携带了与客户端相同的 TCP 选项。

```
> Frame 440: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface \Device\NPF_{...}, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 192.168.188.1, Dst: 192.168.188.1
> Transmission Control Protocol, Src Port: 8888, Dst Port: 60945, Seq: 0, Ack: 1, Len: 0
  Source Port: 8888
  Destination Port: 60945
  [Stream index: 19]
  [Stream Packet Number: 2]
  [Conversation completeness: Complete, WITH_DATA (63)]
  [TCP Segment Len: 0]
  Sequence Number: 0 (relative sequence number)
  Sequence Number (raw): 601446175
  [Next Sequence Number: 1 (relative sequence number)]
  Acknowledgment Number: 1 (relative ack number)
  Acknowledgment number (raw): 1281187257
  1010 .... = Header Length: 40 bytes (10)
  Flags: 0x012 (SYN, ACK)
    000. .... = Reserved: Not set
    ...0 .... = Accurate ECN: Not set
    ....0... = Congestion Window Reduced: Not set
    ....0.. = ECN-Echo: Not set
    .....0.. = Urgent: Not set
    ....1... = Acknowledgment: Set
    .....0... = Push: Not set
    .....0... = Reset: Not set
    ....1... = Syn: Set
    .....0 = Fin: Not set
  [TCP Flags: .....A..S.]
```

图 4.6: 第二次握手

可以看到第二次握手服务器收到客户端的 SYN 报文后, 回应一个带有 SYN 和 ACK 标志位的报文。这个报文表示服务器同意建立连接, 并且确认收到了客户端的 SYN 请求, 报文序列号为 0, 确认号为 1。

- 客户端收到服务器的 SYN-ACK 报文后, 回复一个 ACK 报文 (Packet 441), 确认号为 1, 窗口大小为 261888。至此, 第一条 TCP 连接建立完成。

```

> Frame 441: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface \Device\NPF_{...}, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 192.168.188.1, Dst: 192.168.188.1
< Transmission Control Protocol, Src Port: 60945, Dst Port: 8888, Seq: 1, Ack: 1, Len: 0
  Source Port: 60945
  Destination Port: 8888
  [Stream index: 19]
  [Stream Packet Number: 3]
  [Conversation completeness: Complete, WITH_DATA (63)]
  [TCP Segment Len: 0]
  Sequence Number: 1 (relative sequence number)
  Sequence Number (raw): 1281187257
  [Next Sequence Number: 1 (relative sequence number)]
  Acknowledgment Number: 1 (relative ack number)
  Acknowledgment number (raw): 601446176
  1000 .... = Header Length: 32 bytes (8)
  Flags: 0x010 (ACK)
    000. .... = Reserved: Not set
    ...0 .... = Accurate ECN: Not set
    ....0... = Congestion Window Reduced: Not set
    ....0... = ECN-Echo: Not set
    ....0... = Urgent: Not set
    ....1... = Acknowledgment: Set
    ....0... = Push: Not set
    ....0... = Reset: Not set
    ....0... = Syn: Not set
    ....0... = Fin: Not set
  [TCP Flags: .....A....]

```

图 4.7: 第三次握手

可以看到第三次握手客户端收到服务器的 SYN-ACK 报文后, 发送一个带有 ACK 标志位的报文给服务器, 表示客户端确认收到了服务器的响应。此时, 连接建立完成。报文序列号为 1, 确认号为 1。

在第一条连接建立后不久, 客户端又发起第二条 TCP 连接请求。过程与第一条连接类似, 都是通过三次握手建立连接 (Packets 450-452)。

注意到客户端访问一次 Web 页面时, 通过两个不同的端口与服务器建立了两次 TCP 连接, 查阅资料后得知, 在 HTTP 通信中, 客户端通常会为每个请求/响应建立新的 TCP 连接。**第一次连接用于请求和响应 HTML 页面。第二次连接用于请求和响应页面中引用的其他资源, 如图片、音频等。**HTTP 通信过程中客户端与服务器建立了两次 TCP 连接, 分别用于获取 HTML 页面和页面中引用的其他资源。

## (2) TCP 资源传输过程

由于设置浏览器退出时清除缓存, 因此客户端访问 Web 页面时所需的资源需要重新从服务器端获取, 需要一个较长的传输过程。在传输过程中, 我注意到服务器会向客户端连续发送带有 ACK 标志位的数据包, 最终再发送带有 ACK 和 PSH 标志的数据包, 表示数据传输完成。而客户端最后也会回复一个带有 ACK 标志位的数据包, 确认收到了所有数据。这是 TCP 协议中正常的传输行为。

458	6.582941	192.168.188.1	192.168.188.1	HTTP	21874 HTTP/1.1 200 OK (JPEG JFIF image)
459	6.582968	192.168.188.1	192.168.188.1	TCP	56 60945 → 8888 [ACK] Seq=905 Ack=91090 Win=2597376 Len=0 TSval=13230627 TSecr=13230627
472	6.645188	192.168.188.1	192.168.188.1	HTTP	450 GET /introduction.mp3 HTTP/1.1
473	6.645249	192.168.188.1	192.168.188.1	TCP	56 8888 → 60945 [ACK] Seq=91090 Ack=1299 Win=2618368 Len=0 TSval=13230689 TSecr=13230689
474	6.646794	192.168.188.1	192.168.188.1	TCP	65539 8888 → 60945 [ACK] Seq=91090 Ack=1299 Win=2618368 Len=65483 TSval=13230690 TSecr=13230689
475	6.646815	192.168.188.1	192.168.188.1	TCP	471 8888 → 60945 [PSH, ACK] Seq=156573 Ack=1299 Win=2618368 Len=415 TSval=13230690 TSecr=13230689
476	6.646868	192.168.188.1	192.168.188.1	TCP	56 60945 → 8888 [ACK] Seq=1299 Ack=156988 Win=2619136 Len=0 TSval=13230691 TSecr=13230690
477	6.647083	192.168.188.1	192.168.188.1	TCP	65539 8888 → 60945 [ACK] Seq=156988 Ack=1299 Win=2618368 Len=65483 TSval=13230691 TSecr=13230691
478	6.647102	192.168.188.1	192.168.188.1	TCP	109 8888 → 60945 [PSH, ACK] Seq=222471 Ack=1299 Win=2618368 Len=53 TSval=13230691 TSecr=13230691
479	6.647144	192.168.188.1	192.168.188.1	TCP	56 60945 → 8888 [ACK] Seq=1299 Ack=222524 Win=2619136 Len=0 TSval=13230691 TSecr=13230691
480	6.647357	192.168.188.1	192.168.188.1	TCP	65539 8888 → 60945 [ACK] Seq=222524 Ack=1299 Win=2618368 Len=65483 TSval=13230691 TSecr=13230691
481	6.647395	192.168.188.1	192.168.188.1	TCP	109 8888 → 60945 [PSH, ACK] Seq=288007 Ack=1299 Win=2618368 Len=53 TSval=13230691 TSecr=13230691
482	6.647488	192.168.188.1	192.168.188.1	TCP	56 60945 → 8888 [ACK] Seq=1299 Ack=288060 Win=2553600 Len=0 TSval=13230691 TSecr=13230691
483	6.647665	192.168.188.1	192.168.188.1	TCP	65539 8888 → 60945 [ACK] Seq=288060 Ack=1299 Win=2618368 Len=65483 TSval=13230691 TSecr=13230691
484	6.647680	192.168.188.1	192.168.188.1	TCP	109 8888 → 60945 [PSH, ACK] Seq=353543 Ack=1299 Win=2618368 Len=53 TSval=13230691 TSecr=13230691
485	6.647741	192.168.188.1	192.168.188.1	TCP	56 60945 → 8888 [ACK] Seq=1299 Ack=353596 Win=2488064 Len=0 TSval=13230691 TSecr=13230691
486	6.647768	192.168.188.1	192.168.188.1	TCP	56 [TCP Window Update] 60945 → 8888 [ACK] Seq=1299 Ack=353596 Win=2619136 Len=0 TSval=13230691 TSecr=13230691

图 4.8: 资源传输过程

在 TCP 协议中, PSH 位的作用是: 当发送端的应用程序向 TCP 层交出数据时, 它会设置 PSH 标志位, 告知 TCP 层应该尽快将这些数据发送出去, 而不要等待更多数据积累。这可以减少延迟, 接

收端应用程序无需等待更多数据到达, 就可以立即处理当前收到的数据。在上述应用场景中, 服务器在向客户端传输完所有数据后, 发送了一个同时设置了 ACK 和 PSH 标志位的数据包。ACK 标志位表示确认收到了之前的数据, PSH 标志位表示应该立即将这些最后的数据推送给接收端应用程序。这样做可以确保数据的及时处理, 有助于提高应用程序的响应速度和用户体验。

并且在资源传输的过程中, 客户端会定期向服务器发送窗口更新报文, 如图 4.8 最后一行所示, 用于通知服务器自己当前的接收窗口大小已经发生变化, 服务器收到这种窗口更新报文后, 就可以相应地调整自己的发送速率, 避免向客户端发送过多的数据导致接收缓冲区溢出。这种窗口更新是 TCP 流量控制的重要组成部分, 确保了数据传输的有效性和稳定性, 是正常的 TCP 通信行为。通过这种方式, 双方能够协调流量, 优化网络资源的使用, 提高整体通信效率。

当客户端完成对服务器资源的接收后, 用于资源传输的端口会关闭与服务器端口的 TCP 连接。

678	6.675311	192.168.188.1	192.168.188.1	TCP	109 8888 → 60945 [PSH, ACK] Seq=4416775 Ack=1299 Win=2618368 Len=53 TSval=13230719 TSecr=13230719
679	6.675441	192.168.188.1	192.168.188.1	TCP	56 60945 → 8888 [ACK] Seq=1299 Ack=4416828 Win=1636096 Len=0 TSval=13230719 TSecr=13230719
680	6.675695	192.168.188.1	192.168.188.1	TCP	65539 8888 → 60945 [ACK] Seq=4416828 Ack=1299 Win=2618368 Len=65483 TSval=13230719 TSecr=13230719
681	6.675717	192.168.188.1	192.168.188.1	TCP	109 8888 → 60945 [PSH, ACK] Seq=4482311 Ack=1299 Win=2618368 Len=53 TSval=13230719 TSecr=13230719
682	6.675853	192.168.188.1	192.168.188.1	TCP	56 60945 → 8888 [FIN, ACK] Seq=1299 Ack=4482364 Win=1570560 Len=0 TSval=13230720 TSecr=13230719
683	6.675900	192.168.188.1	192.168.188.1	TCP	56 8888 → 60945 [ACK] Seq=4482364 Ack=1300 Win=2618368 Len=0 TSval=13230720 TSecr=13230720
684	6.675952	192.168.188.1	192.168.188.1	TCP	44 60945 → 8888 [RST, ACK] Seq=1300 Ack=4482364 Win=0 Len=0

图 4.9: 资源传输完毕, 客户端接收端口关闭连接

- 首先, 客户端向服务器发送一个包含 FIN 和 ACK 标志位的报文, 表示它希望关闭与服务器的连接, 同时确认收到了服务器传输的所有数据。此报文的序列号为 1299, 确认号为 4482364。
- 接着, 服务器响应客户端的 FIN 请求, 发送一个包含 ACK 标志位的报文, 确认序列号为 1300, 表示已经接收到客户端的连接关闭请求。
- 然后, 客户端发送一个包含 RST 和 ACK 标志位的报文, 这通常表示客户端主动重置了连接, 而不是经历正常的四次挥手关闭流程。报文的序列号为 1300, 确认号为 4482364。

在 TCP 连接中, RST 标志通常表示连接被重置, 客户端希望立刻终止连接, 而不进行正常的关闭流程。这可能是因为资源接收已经完成, 客户端不再需要与服务器保持连接。发送 RST 报文可以快速释放连接相关的资源。

这样的连接关闭过程是 TCP 协议中的正常行为。客户端主动发送 FIN 报文表示自己已经完成数据接收, 服务器回复 ACK 以确认收到。最后客户端发送 RST 报文则是为了快速终止这个连接, 以便后续可以建立新的连接。

### (3) TCP 连接释放过程: 四次挥手

685	6.684692	192.168.188.1	192.168.188.1	HTTP	568 GET /logo.png HTTP/1.1
686	6.684772	192.168.188.1	192.168.188.1	TCP	56 8888 → 60946 [ACK] Seq=1 Ack=513 Win=2619136 Len=0 TSval=13230728 TSecr=13230728
687	6.686328	192.168.188.1	192.168.188.1	HTTP	323 HTTP/1.1 304 Not Modified
688	6.686407	192.168.188.1	192.168.188.1	TCP	56 60946 → 8888 [ACK] Seq=513 Ack=268 Win=2618880 Len=0 TSval=13230730 TSecr=13230730
1072	11.069943	192.168.188.1	192.168.188.1	TCP	56 60946 → 8888 [FIN, ACK] Seq=513 Ack=268 Win=2618880 Len=0 TSval=13235114 TSecr=13230730
1073	11.069971	192.168.188.1	192.168.188.1	TCP	56 8888 → 60946 [ACK] Seq=268 Ack=514 Win=2619136 Len=0 TSval=13235114 TSecr=13235114
1080	11.070382	192.168.188.1	192.168.188.1	TCP	56 8888 → 60946 [FIN, ACK] Seq=268 Ack=514 Win=2619136 Len=0 TSval=13235114 TSecr=13235114
1081	11.070427	192.168.188.1	192.168.188.1	TCP	56 60946 → 8888 [ACK] Seq=514 Ack=269 Win=2618880 Len=0 TSval=13235114 TSecr=13235114

图 4.10: TCP 连接释放过程: 四次挥手

- 客户端首先发送一个包含 FIN 和 ACK 标志位的报文, 序列号为 513, 确认号为 268。这表示客户端希望主动关闭连接, 并确认收到了之前的数据。
- 服务器收到客户端的包含 FIN 标志位的报文后, 回复一个包含 ACK 标志位的报文以确认收到。序列号为 268, 确认号为 514。这表示服务器已经收到了客户端的连接关闭请求, 并进行了确认。



- 此时, 尽管客户端已经发起了关闭连接请求, 但服务器仍然可以继续向客户端传输数据。服务器可以利用这个“半关闭”的状态, 将一些剩余的数据发送给客户端。
- 服务器随后也发送一个包含 FIN 和 ACK 标志位的报文, 序列号为 268, 确认号为 514。这表示服务器也希望关闭与客户端的 TCP 连接。
- 客户端收到服务器包含 FIN 标志位的报文后, 回复一个包含 ACK 标志位的报文以确认收到。序列号为 514, 确认号为 269。这表示客户端已经收到了服务器的连接关闭请求, 并进行了确认。

#### 4.2.2 当设置浏览器退出保留缓存时

TCP 建立连接的三次握手过程与之前相同, 这里我们主要分析存在差异的地方。

##### (1) TCP 资源传输过程

由于设置浏览器退出时保留缓存, 因此客户端访问 Web 页面时所需的资源不需要重新从服务器端获取, 而是可以直接从本地缓存中获取资源。这样可以显著减少网络传输, 提高资源加载的效率。

No.	Time	Source	Destination	Protocol	Length	Info
691	4.161185	192.168.188.1	192.168.188.1	TCP	64	50879 → 8888 [SYN, ACK] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM TSval=32981597 TSecr=0
692	4.161241	192.168.188.1	192.168.188.1	TCP	64	8888 → 50879 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM TSval=32981597 TSecr=32981597
693	4.161280	192.168.188.1	192.168.188.1	TCP	56	50879 → 8888 [ACK] Seq=1 Ack=1 Win=2619136 Len=0 TSval=32981597 TSecr=32981597
694	4.173079	192.168.188.1	192.168.188.1	HTTP	622	GET / HTTP/1.1
695	4.173150	192.168.188.1	192.168.188.1	TCP	56	8888 → 50879 [ACK] Seq=1 Ack=567 Win=2619136 Len=0 TSval=32981609 TSecr=32981609
696	4.173988	192.168.188.1	192.168.188.1	HTTP	321	HTTP/1.1 304 Not Modified
697	4.174022	192.168.188.1	192.168.188.1	TCP	56	50879 → 8888 [ACK] Seq=567 Ack=266 Win=2618880 Len=0 TSval=32981610 TSecr=32981610
698	4.190392	192.168.188.1	192.168.188.1	TCP	64	50880 → 8888 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM TSval=32981626 TSecr=0
699	4.190477	192.168.188.1	192.168.188.1	TCP	64	8888 → 50880 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM TSval=32981626 TSecr=32981626
700	4.190574	192.168.188.1	192.168.188.1	TCP	56	50880 → 8888 [ACK] Seq=1 Ack=1 Win=2619136 Len=0 TSval=32981626 TSecr=32981626
701	4.192808	192.168.188.1	192.168.188.1	HTTP	568	GET /logo.png HTTP/1.1
702	4.192858	192.168.188.1	192.168.188.1	TCP	56	8888 → 50879 [ACK] Seq=266 Ack=1079 Win=2618624 Len=0 TSval=32981628 TSecr=32981628
703	4.193789	192.168.188.1	192.168.188.1	HTTP	323	HTTP/1.1 304 Not Modified
704	4.193826	192.168.188.1	192.168.188.1	TCP	56	50879 → 8888 [ACK] Seq=1079 Ack=533 Win=2618624 Len=0 TSval=32981629 TSecr=32981629
745	4.246355	192.168.188.1	192.168.188.1	HTTP	546	GET /introduction.mp3 HTTP/1.1
746	4.246427	192.168.188.1	192.168.188.1	TCP	56	8888 → 50879 [ACK] Seq=533 Ack=1569 Win=2618112 Len=0 TSval=32981682 TSecr=32981682
749	4.247251	192.168.188.1	192.168.188.1	HTTP	324	HTTP/1.1 304 Not Modified
750	4.247276	192.168.188.1	192.168.188.1	TCP	56	50879 → 8888 [ACK] Seq=1569 Ack=801 Win=2618368 Len=0 TSval=32981683 TSecr=32981682

图 4.11: 资源传输过程

- 客户端发起 GET /logo.png 的 HTTP 请求。
- 服务器回复一个包含 ACK 标志位的报文, 确认收到了客户端的请求。
- 服务器返回 HTTP 状态码 304 Not Modified, 表示客户端请求的资源没有更新。
- 客户端回复一个包含 ACK 标志位的报文, 确认收到了服务器的响应。

从上述过程可以看出, 由于浏览器设置为退出时保留缓存, 因此客户端访问 Web 页面时所需的资源不需要重新从服务器端获取。服务器只需要回复一个 HTTP 304 Not Modified 状态码, 表示资源没有更新, 客户端可以继续使用缓存中的资源。这种情况下, 就不需要经历完整的资源传输过程, 节省了网络带宽和服务器压力。

##### (2) TCP 连接释放过程: 四次挥手

906	9.717385	192.168.188.1	192.168.188.1	TCP	56	50880 → 8888 [FIN, ACK] Seq=1 Ack=1 Win=2619136 Len=0 TSval=32987153 TSecr=32981626
907	9.717418	192.168.188.1	192.168.188.1	TCP	56	8888 → 50880 [ACK] Seq=1 Ack=2 Win=2619136 Len=0 TSval=32987153 TSecr=32987153
908	9.717481	192.168.188.1	192.168.188.1	TCP	56	50879 → 8888 [FIN, ACK] Seq=1569 Ack=801 Win=2618368 Len=0 TSval=32987153 TSecr=32981682
909	9.717506	192.168.188.1	192.168.188.1	TCP	56	8888 → 50879 [ACK] Seq=801 Ack=1570 Win=2618112 Len=0 TSval=32987153 TSecr=32987153
918	9.718140	192.168.188.1	192.168.188.1	TCP	56	8888 → 50880 [FIN, ACK] Seq=1 Ack=2 Win=2619136 Len=0 TSval=32987154 TSecr=32987153
920	9.718186	192.168.188.1	192.168.188.1	TCP	56	50880 → 8888 [ACK] Seq=2 Ack=2 Win=2619136 Len=0 TSval=32987154 TSecr=32987154
926	9.718663	192.168.188.1	192.168.188.1	TCP	56	8888 → 50879 [FIN, ACK] Seq=801 Ack=1570 Win=2618112 Len=0 TSval=32987154 TSecr=32987153
927	9.718691	192.168.188.1	192.168.188.1	TCP	56	50879 → 8888 [ACK] Seq=1570 Ack=882 Win=2618368 Len=0 TSval=32987154 TSecr=32987154

图 4.12: TCP 连接释放过程: 四次挥手

TCP 释放过程与之前类似，不同的是由于 Web 页面资源缓存在了本地，没有进行实际的资源传输，因此负责资源接收的端口也经历四次挥手过程正常关闭，不需要在资源接收完毕后提前发送包含 RST 和 ACK 标志位的报文来提前结束连接。由图 4.12 可以看出，此时两个端口都是在页面关闭时进行 TCP 连接释放的。

- 首先，客户端主动关闭第一个端口的 TCP 连接。客户端发送一个包含 FIN 和 ACK 标志位的报文 (Packet 906)，报文的序列号为 1，确认号为 1。表示客户端希望主动关闭与服务器的一个 TCP 连接。
- 服务器收到客户端的 FIN 报文后，回复一个 ACK 报文 (Packet 907)，报文的序列为 1，确认为 2。表示服务器已经收到了客户端的连接关闭请求，并进行了确认。
- 然后，客户端主动关闭第二个端口的 TCP 连接。客户端发送一个包含 FIN 和 ACK 标志位的报文 (Packet 908)。报文的序列号为 1569，确认号为 801。表示客户端希望主动关闭另一个与服务器的 TCP 连接
- 服务器收到客户端的 FIN 报文后，回复一个 ACK 报文 (Packet 909)，报文的序列号为 801，确认号为 1570。表示服务器已经收到了客户端的连接关闭请求，并进行了确认。
- 服务器随后发送一个包含 FIN 和 ACK 标志位的报文 (Packet 918)，报文的序列号为 1，确认号为 2。表示服务器也希望主动关闭与客户端的第一个 TCP 连接。
- 客户端收到服务器的 FIN 报文后，回复一个 ACK 报文 (Packet 920)，报文的序列号为 2，确认号为 2。表示客户端已经收到了服务器的连接关闭请求，并进行了确认。
- 服务器随后也发送一个包含 FIN 和 ACK 标志位的报文 (Packet 926)，报文的序列号为 801，确认号为 1570。表示服务器也希望主动关闭与客户端的第二个 TCP 连接。
- 客户端收到服务器的 FIN 报文后，回复一个 ACK 报文 (Packet 927)，报文的序列号为 1570，确认号为 802。表示客户端已经收到了服务器的连接关闭请求，并进行了确认。

## 5 实验内容总结

本次实验探讨了 TCP 连接的建立和释放过程，以及在浏览器设置保留或者清除缓存的情况下，TCP 在资源传输和连接释放过程中的差异。主要包括以下几个方面：

- 分析 TCP 连接的三次握手建立过程。客户端发起连接请求，服务器确认并同意连接，客户端最终确认连接建立。
- 分析 TCP 连接的四次挥手释放过程。客户端主动发起关闭连接请求，服务器确认客户端的关闭请求，服务器主动发起关闭连接请求，客户端确认服务器的关闭请求。
- 探讨在浏览器设置清除缓存的情况下：
  - TCP 资源传输过程的差异：由于设置浏览器退出时清除缓存，因此客户端访问 Web 页面时所需的资源需要重新从服务器端获取，需要一个较长的传输过程。在传输过程中，服务器会向客户端发送带有 ACK 和 PSH 标志的数据包，告知 TCP 层应该尽快将这些数据发送出去，以减少延迟，提高响应速度。在传输过程中，客户端还会定期向服务器发送窗口更新报文，用于通知服务器自己当前的接收窗口大小，以优化数据传输的效率。

- TCP 连接释放过程的差异: 客户端发送一个包含 RST 和 ACK 标志位的报文, 表示客户端希望立刻终止连接, 而不进行正常的关闭流程。这是因为资源接收已经完成, 客户端不再需要与服务器保持连接。发送 RST 报文可以快速释放连接相关的资源。
- 探讨在浏览器设置保留缓存的情况下:
  - TCP 资源传输过程的差异: 客户端可以直接从缓存中获取资源, 无需从服务器下载; 服务器只需返回 HTTP 304 Not Modified 状态码; 大幅减少网络传输, 提高资源加载效率。
  - TCP 连接释放过程的差异: 由于无实际资源传输, 负责资源接收的端口仍然经历正常的四次挥手过程, 确保连接能够被安全、可靠地终止。

在实现过程中, 我注意到以下几个关键点:

- 三次握手的重要性: 通过三次握手过程, 确保了客户端和服务端之间的连接建立是可靠的, 避免了因网络延迟或丢包导致的连接问题。
- 四次挥手的必要性: 四次挥手过程确保了数据的完整传输和连接的有序关闭, 避免了数据丢失和资源浪费, 体现了 TCP 的可靠性设计。
- HTTP 状态码的作用: HTTP 304 Not Modified 状态码的使用, 展示了有效的资源管理策略, 减少了不必要的数据传输, 提高了性能。
- 流量控制的实现: TCP 通过窗口更新报文有效地管理了数据传输速率, 避免了接收缓冲区的溢出。这一机制是保证 TCP 可靠性和高效性的关键之一。
- RST 报文的特殊含义: 使用 RST 报文终止连接的情况下, 强调了客户端在完成资源接收后可能需要快速释放连接资源, 这在实际应用中是非常重要的。

通过这个实验, 我了解到了 TCP 协议在连接建立、资源传输和连接释放过程中的具体工作机制。特别是在浏览器设置保留缓存的情况下, TCP 传输过程可以得到优化, 从而提高网络性能和用户体验。

本次实验涉及到了 TCP 连接的基本原理、网络通信过程中的优化策略等内容, 对于后续深入学习计算机网络课程奠定了良好的基础。