

计算机网络实验报告

Lab2 配置Web服务器，编写简单页面，分析交互过程

网络安全空间学院 信息安全专业

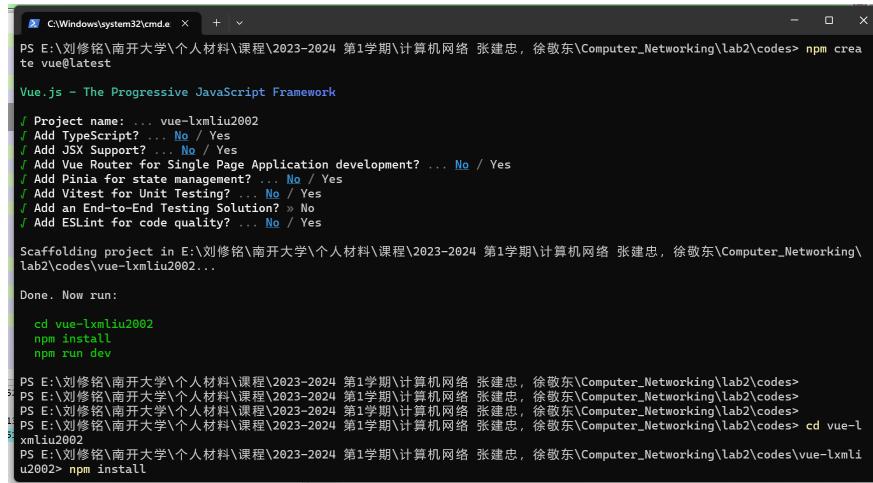
2112492 刘修铭 1063

https://github.com/lxmliu2002/Computer_Networking

一、Web服务器配置

本项目借助Node.js，使用Vue框架实现Web服务器配置。

使用命令行，输入相关创建指令，可以看到项目得以成功创建。



```
C:\Windows\system32\cmd.exe + PS E:\刘修铭\南开大学\个人材料\课程\2023-2024 第1学期\计算机网络 张建忠, 徐敬东\Computer_Networking\lab2\codes> npm create vue@latest

Vue.js - The Progressive JavaScript Framework

Project name: ... vue-lxmliu2002
Add TypeScript? ... No / Yes
Add JSX Support? ... No / Yes
Add Vue Router for Single Page Application development? ... No / Yes
Add Pinia for state management? ... No / Yes
Add Vitest for Unit Testing? ... No / Yes
Add an End-to-End Testing Solution? > No
Add ESLint for code quality? ... No / Yes

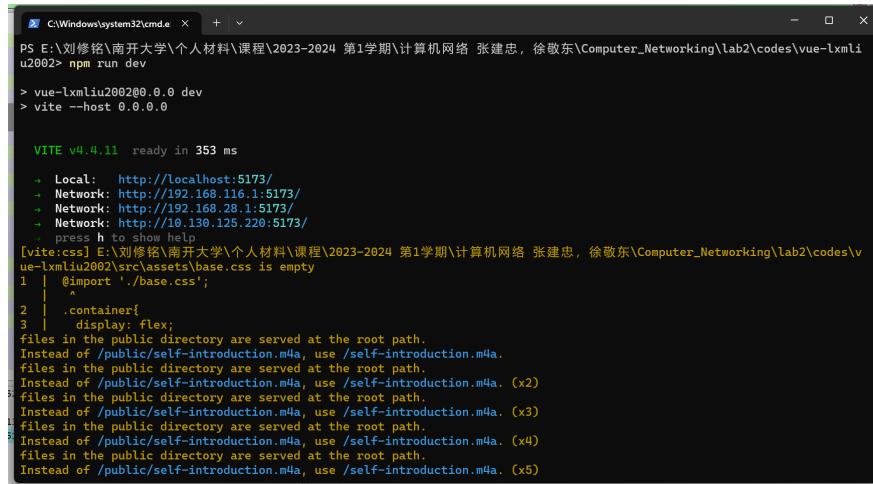
Scaffolding project in E:\刘修铭\南开大学\个人材料\课程\2023-2024 第1学期\计算机网络 张建忠, 徐敬东\Computer_Networking\lab2\codes\vue-lxmliu2002...

Done. Now run:

cd vue-lxmliu2002
npm install
npm run dev

PS E:\刘修铭\南开大学\个人材料\课程\2023-2024 第1学期\计算机网络 张建忠, 徐敬东\Computer_Networking\lab2\codes>
PS E:\刘修铭\南开大学\个人材料\课程\2023-2024 第1学期\计算机网络 张建忠, 徐敬东\Computer_Networking\lab2\codes>
PS E:\刘修铭\南开大学\个人材料\课程\2023-2024 第1学期\计算机网络 张建忠, 徐敬东\Computer_Networking\lab2\codes> cd vue-lxmliu2002
PS E:\刘修铭\南开大学\个人材料\课程\2023-2024 第1学期\计算机网络 张建忠, 徐敬东\Computer_Networking\lab2\codes\vue-lxmliu2002> npm install
```

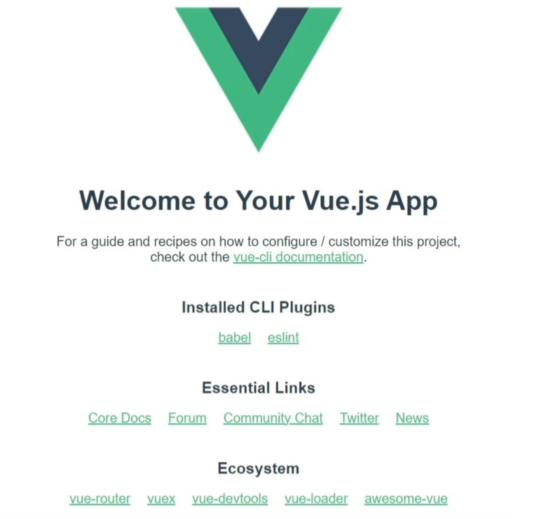
接着输入 `npm run dev`，可以看到服务器成功启动。



```
C:\Windows\system32\cmd.exe + PS E:\刘修铭\南开大学\个人材料\课程\2023-2024 第1学期\计算机网络 张建忠, 徐敬东\Computer_Networking\lab2\codes> vue-lxmliu2002@0.0.0 dev
> vite --host 0.0.0.0

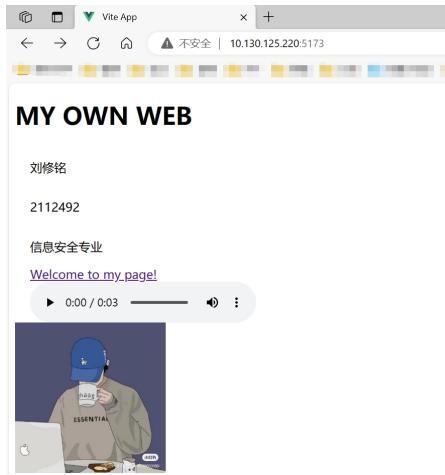
VITE v4.4.1 ready in 353 ms
- Local: http://localhost:5173/
- Network: http://192.168.116.1:5173/
- Network: http://192.168.28.1:5173/
- Network: http://10.130.125.220:5173/
- press h to show help
[vite:css] E:\刘修铭\南开大学\个人材料\课程\2023-2024 第1学期\计算机网络 张建忠, 徐敬东\Computer_Networking\lab2\codes\vue-lxmliu2002\src\assets\base.css is empty
1 | @import './base.css';
2 | .container{
3 |   display: flex;
files in the public directory are served at the root path.
Instead of /public/self-introduction.m4a, use /self-introduction.m4a.
files in the public directory are served at the root path.
Instead of /public/self-introduction.m4a, use /self-introduction.m4a. (x2)
files in the public directory are served at the root path.
Instead of /public/self-introduction.m4a, use /self-introduction.m4a. (x3)
files in the public directory are served at the root path.
Instead of /public/self-introduction.m4a, use /self-introduction.m4a. (x4)
files in the public directory are served at the root path.
Instead of /public/self-introduction.m4a, use /self-introduction.m4a. (x5)
```

打开浏览器，输入 `http://10.130.125.220:5173` 得到如下界面。



二、简单页面编写

在给定框架的基础上，进行内容删改，使之成为一个自己的个人主页，包括专业、学号、姓名、LOGO、自我介绍音频信息等内容。为避免报告冗长，已将源代码附后。



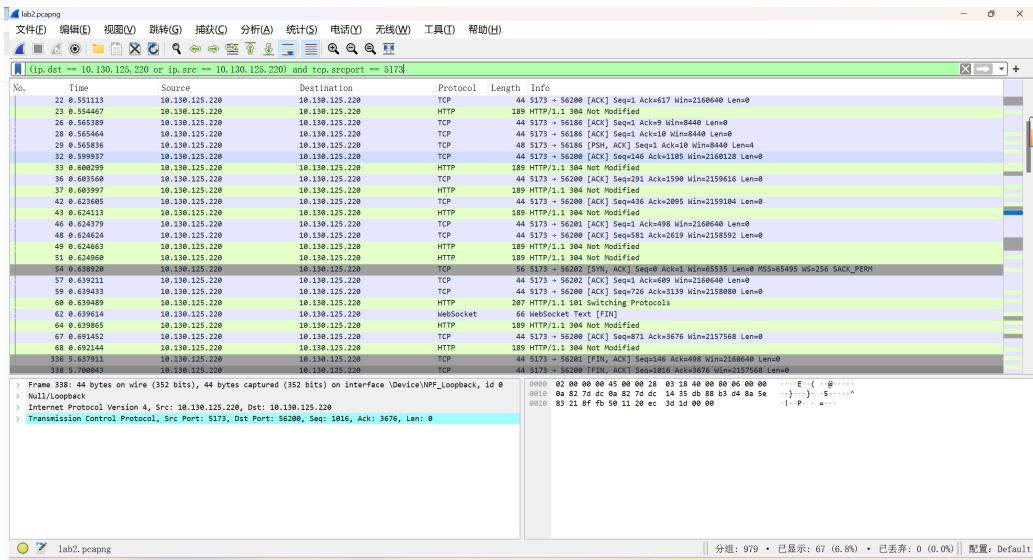
三、交互过程分析

(一) 准备工作

- 下载安装Wireshark抓包软件
- 启动软件，选择 `Adapter for loopback traffic capture` 接口
- 按照前面创建的设定设置过滤器 (`ip.dst == 10.130.125.220 or ip.src == 10.130.125.220`)
`and tcp.srcport == 5173`

(二) 数据分析

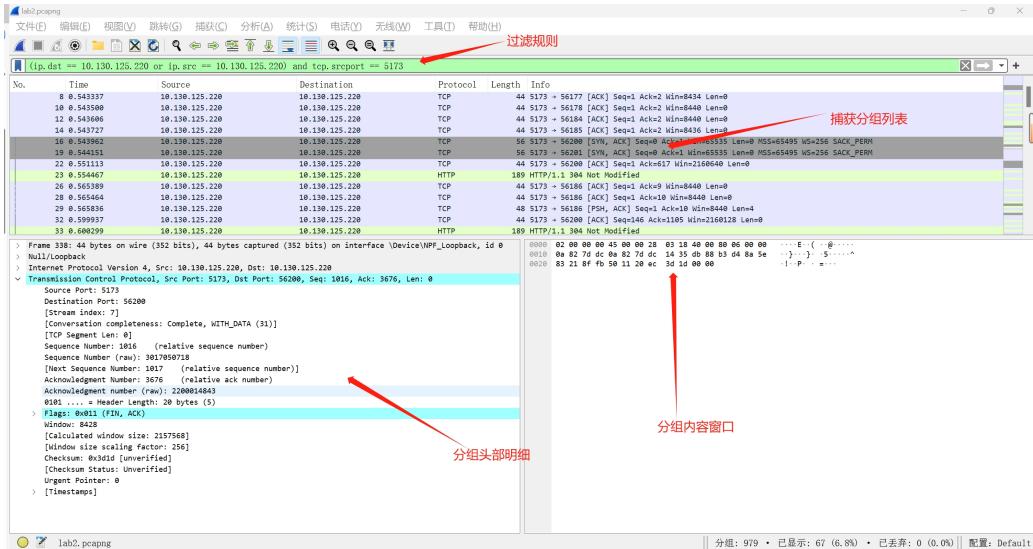
按照上面设定，运行一段时间，得到如下的抓包结果。



接下来将对抓取到的数据包进行分析，进而分析整个交互过程。

1. Wireshark结构

Wireshark主要由过滤规则、捕获分组列表、分组头部明细、分组内容窗口等部分组成。



下面着重分析 分组头部明细 部分。

```
> Frame 338: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 10.130.125.220, Dst: 10.130.125.220
> Transmission Control Protocol, Src Port: 5173, Dst Port: 56200, Seq: 1016, Ack: 3676, Len: 0
```

- Frame : 物理层的数据帧概况
- Internet Protocol Version 4 : 互联网IP包头部信息
- Transmission Control Protocol : 传输层的数据段头部信息

展开 Transmission Control Protocol，可以看到其与TCP报文数据段格式相对应。

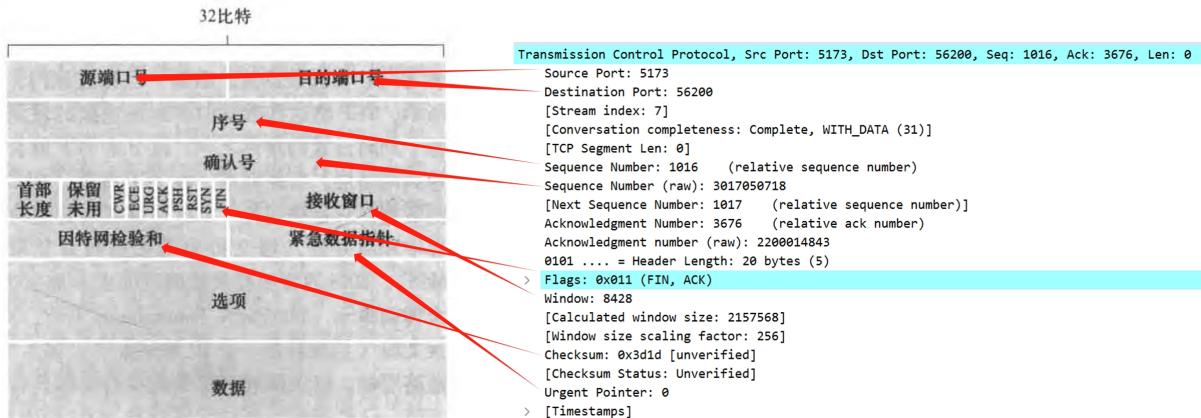


图 3-29 TCP 报文段结构

- TCP源端口 (Source Port) : 占16位 (bit) , 源计算机上的应用程序的端口号
- TCP目的端口 (Destination Port) : 占16位 (bit) , 目标计算机的应用程序端口号
- TCP序列号 (Sequence Number) : 占32位 (bit) , 表示本报文段所发送数据的第一个字节的编号。在TCP连接中, 所传送的字节流的每一个字节都会按顺序编号
- TCP确认号: 占32位 (bit) , 表示接收方期望收到发送方下一个报文段的第一个字节数据的编号
- 数据偏移 (即首部长度, Header Length) : 占4位 (bit) , 它指出 TCP 报文段的数据起始处距离 TCP 报文段的起始处有多远, 也就是TCP首部的长度
- 保留字段 (Reserved) : 占 6 位 (bit) 。为 TCP 将来的发展预留空间, 目前必须全部为 0
- 标志位字段: 各占1位 (bit)
 - URG (Urgent) : 表示本报文段中发送的数据是否包含紧急数据。URG=1 时表示有紧急数据。当 URG=1 时, 后面的紧急指针字段才有效
 - ACK: 表示前面的确认号字段是否有效。ACK=1 时表示有效。只有当 ACK=1 时, 前面的确认号字段才有效。TCP 规定, 连接建立后, ACK 必须为 1
 - PSH (Push) : 告诉对方收到该报文段后是否立即把数据推送给上层。如果值为 1, 表示应当立即把数据提交给上层, 而不是缓存起来
 - RST: 表示是否重置连接。如果 RST=1, 说明 TCP 连接出现了严重错误 (如主机崩溃) , 必须释放连接, 然后再重新建立连接
 - SYN: 在建立连接时使用, 用来同步序号
 - 当 SYN=1, ACK=0 时, 表示这是一个请求建立连接的报文段
 - 当 SYN=1, ACK=1 时, 表示对方同意建立连接
 - SYN=1 时, 说明这是一个请求建立连接或同意建立连接的报文。只有在前两次握手中 SYN 才为 1
 - FIN: 标记数据是否发送完毕。如果 FIN=1, 表示数据已经发送完成, 可以释放连接
- 窗口大小字段 (Window Size) : 占16位 (bit) , TCP流量控制, 表示目前还有多少空间, 能接收多少数据量
- TCP 校验位 (TCP Checksum) : 占 16 位 (bit) , 它用于确认传输的数据是否有损坏。发送端基于数据内容校验生成一个数值, 接收端根据接收的数据校验生成一个值。两个值必须相同, 才能证明数据是有效的。如果两个值不同, 则丢掉这个数据包。

- 紧急指针 (Urgent Pointer) : 占 16 位 (bit) , 仅当前面的 URG 控制位为 1 时才有意义, 它指出本数据段中为紧急数据的字节数, 当所有紧急数据处理完后, TCP 就会告诉应用程序恢复到正常操作。即使当前窗口大小为 0, 也是可以发送紧急数据的, 因为紧急数据无须缓存。
- 可选项字段: 可以自定义头部字段, 长度不定, 但长度必须是 32bits 的整数倍, 最多40个字节
- TCP数据: 数据部分(可以不发送任何数据)

2. TCP三次握手建立连接

TCP使用三次握手建立连接, 防止已经失效的报文段突然又被服务端接收

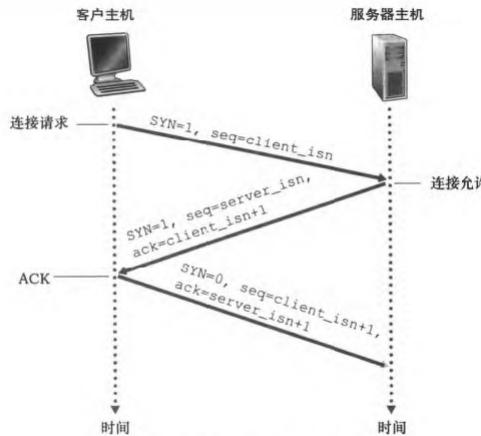
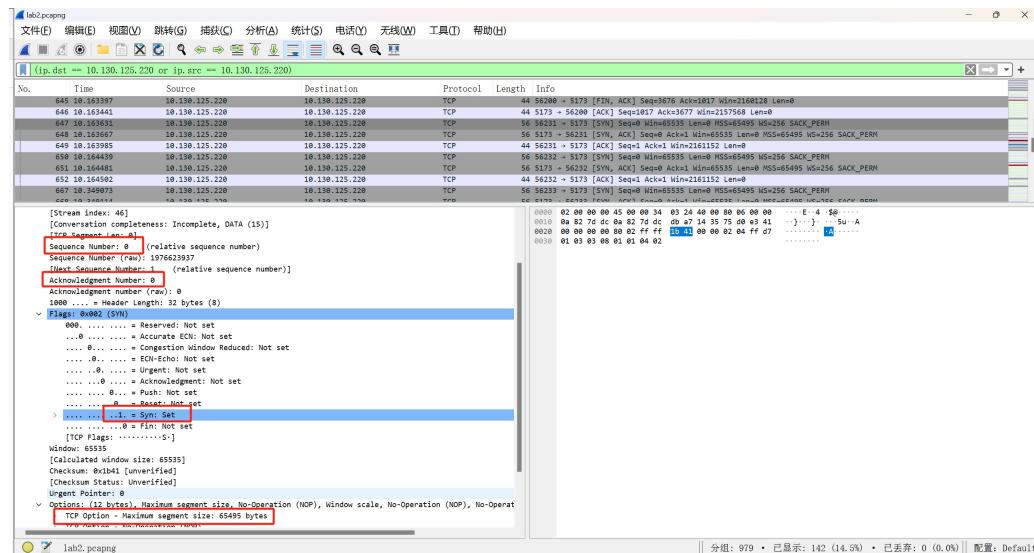


图 3-39 TCP 三次握手: 报文段交换

在wireshark中可以看到三次握手的过程。

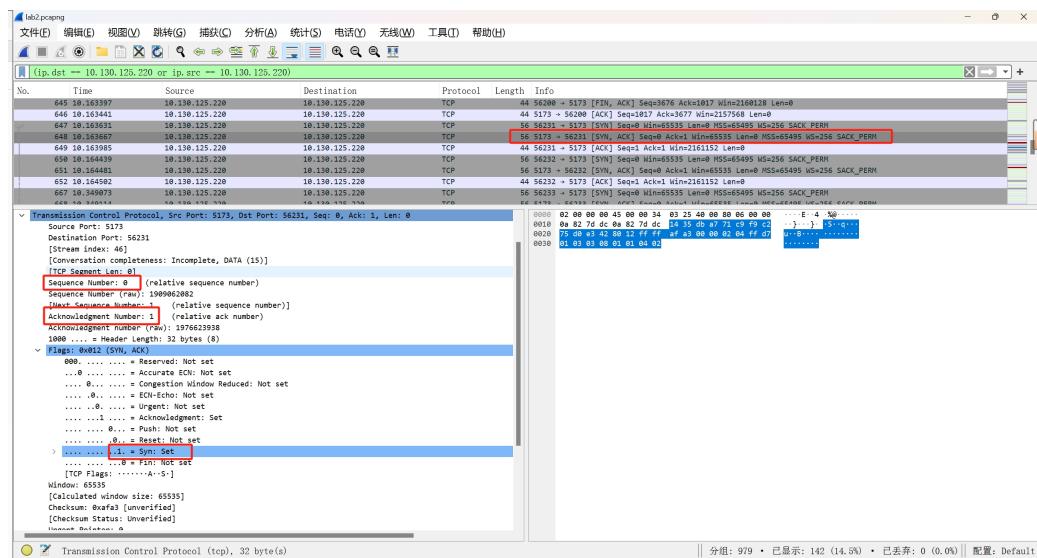
647 10.163631	10.130.125.220	10.130.125.220	TCP	56 56231 -> 5173 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
648 10.163667	10.130.125.220	10.130.125.220	TCP	56 5173 -> 56231 [SYN, ACK] Seq=1 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
649 10.163985	10.130.125.220	10.130.125.220	TCP	44 56231 -> 5173 [ACK] Seq=1 Ack=1 Win=2161152 Len=0

- 第一次握手: 建立连接时, 客户端发送SYN到服务器, 并进入SYN_SENT状态, 等待服务器确认。



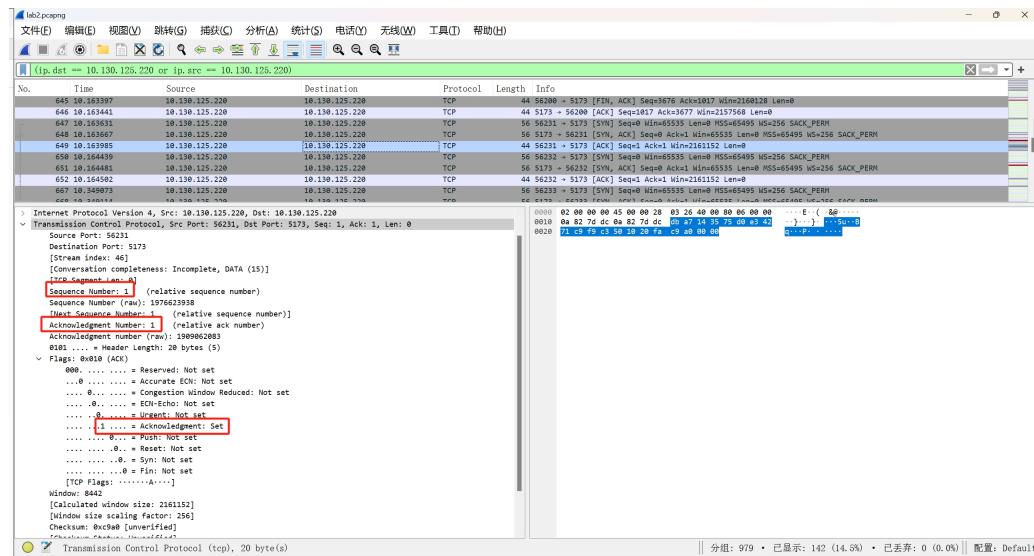
- Source Port: 56231
- Destination Port: 5173
- SYN : 标志位, 表示请求建立连接
- Seq = 0 : 初始建立连接值为0, 数据包的相对序列号从0开始, 表示当前还没有发送数据
 - 0为相对序列号, 实际序列号为1976623937

- Ack =0: 初始建立连接值为0, 已经收到包的数量, 表示当前没有接收到数据
- WIN = 65535来自Window size: 65535
- MSS = 65495来自 Maximum segment size: 65495byte , 最长报文段, TCP包所能携带的最大数据量, 不包含TCP头和Option。一般为MTU值减去IPv4头部(至少20字节)和TCP头部(至少20字节)得到
- WS = 256 来自windows scale : 8 (multiply by 256): 窗口扩张, 放在TCP头之外的Option, 向对方声明一个shift count, 作为2的指数, 再乘以TCP定义的接收窗口, 得到真正的TCP窗口
- TCP客户端向服务器端发送连接请求报文段, 即**客户端**发送SYN到**服务器端**, 并进入SYN_SENT状态, 等待**服务器端**确认
- 第一次握手成功说明客户端的数据可以被服务端收到, 说明客户端的发功能可用, 说明服务端的收功能可用。但客户端自己不知道数据是否被接收
- 第二次握手: 服务器收到请求后, 回送SYN+ACK信令到客户端, 此时服务器进入SYN_RECV状态。



- Source Port: 5173
- Destination Port: 56231
- Seq = 0 : 初始建立值为0, 表示当前还没有发送数据
 - 0为相对序列号, 实际序列号为1909062082, 与第一次握手不同
- Ack = 1 : 表示当前端成功接收的数据位数, 虽然客户端没有发送任何有效数据, 确认号还是被加1, 因为包含SYN或FIN标志位
- 第二次握手时, **服务器端**为该TCP连接分配缓存和变量
- **服务器端**收到数据包后由标志位 SYN=1 得知**客户端**请求建立连接, 然后便发送确认报文段(SYN+ACK信令)到**客户端**, 接着**服务器**进入SYN_RECV状态
- 第二次握手成功说明服务端的数据可以被客户端收到, 说明服务端的发功能可用, 说明客户端的收功能可用。同时客户端知道自己的数据已经正确到达服务端, 自己的发功能正常。但是服务端自己不知道数据是否被接收

- 第三次握手：客户端收到SYN+ACK包，向服务器发送确认ACK包，客户端进入ESTABLISHED状态，服务器收到请求后也进入ESTABLISHED状态，完成三次握手，此时TCP连接成功，客户端与服务器开始传送数据。



- Source Port: 56231
- Destination Port: 5173
- Seq = 1 : 表示当前已经发送1个数据
 - 实际值为1976623938
- Ack = 1 : 表示当前端成功接收的数据位数
 - 实际值为1909062083
- ACK : 标志位, 表示已经收到记录
- 第三次握手时, 客户端为该TCP连接分配缓存和变量
- 客户端收到服务器端确认后, 检查 ack 是否为 x+1, ACK 是否为 1, 如果正确则再发送一个确认报文段给服务器, 同时客户端进入ESTABLISHED状态, 服务器收到请求后也进入ESTABLISHED状态, 三次握手完成
- 第三次握手成功说明服务端知道自己的数据已经正确到达客户端端, 自己的发功能正常。至此服务成功建立

3. 发送与收取数据

客户端和服务端建立连接后, 开始传输数据。

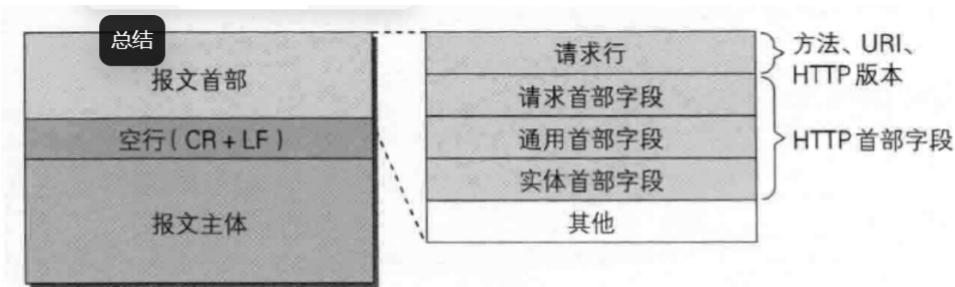
源 IP	目标 IP	协议	序列号	确认号	窗口大小	长度
10.130.125.220	10.130.125.220	TCP	56	56232 → 5173 [SYN]	Seq=0 Win=65535 Len=0 MSS=65495 WS=256	44
10.130.125.220	10.130.125.220	TCP	56	5173 → 56232 [SYN, ACK]	Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256	44
10.130.125.220	10.130.125.220	TCP	44	56232 → 5173 [ACK]	Seq=1 Ack=1 Win=2161152 Len=0 MSS=65495 WS=256	44
10.130.125.220	10.130.125.220	TCP	56	56232 → 5173 [SYN]	Seq=0 Win=65535 Len=0 MSS=65495 WS=256	44
10.130.125.220	10.130.125.220	TCP	56	5173 → 56232 [SYN, ACK]	Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256	44
10.130.125.220	10.130.125.220	TCP	44	56232 → 5173 [ACK]	Seq=1 Ack=1 Win=2161152 Len=0 MSS=65495 WS=256	44
10.130.125.220	10.130.125.220	HTTP	652	65231 → 10.130.125.220 GET / HTTP/1.1	Seq=0 Ack=609 Win=2160640 Len=0	207
10.130.125.220	10.130.125.220	HTTP	44	5173 → 56232 [ACK]	Seq=1 Ack=609 Win=2160640 Len=0	44
10.130.125.220	10.130.125.220	TCP	44	5173 → 56232 [ACK]	Seq=1 Ack=609 Win=2160640 Len=0	44
10.130.125.220	10.130.125.220	WebSocket	66	56232 → 10.130.125.220 Text [FIN]	Seq=0 Ack=609 Win=2160640 Len=0	66
10.130.125.220	10.130.125.220	TCP	44	56232 → 5173 [ACK]	Seq=609 Ack=186 Win=2160896 Len=0	44
10.130.125.220	10.130.125.220	HTTP	660	10.130.125.220 → 56231 GET / HTTP/1.1	Seq=0 Ack=186 Win=2160896 Len=0	189
10.130.125.220	10.130.125.220	HTTP	44	5173 → 56231 [ACK]	Seq=1 Ack=617 Win=2160640 Len=0	44
10.130.125.220	10.130.125.220	HTTP	189	56231 → 10.130.125.220 GET / HTTP/1.1	Seq=1 Ack=617 Win=2160640 Len=0	189

- 浏览器向域名发出HTTP请求;

- 通过TCP->IP (DNS) ->MAC (ARP) ->网关->目的主机；
- 目的主机收到数据帧，通过IP->TCP->HTTP，HTTP协议单元会回应HTTP协议格式封装好的HTML形式数据（HTTP响应）；
- 该HTML数据通过TCP->IP (DNS) ->MAC (ARP) ->网关->我的主机；
- 我的主机收到数据帧，通过IP->TCP->HTTP->浏览器，浏览器以网页形式显示HTML内容。
- HTTP请求由三部分组成：请求行，消息报文，请求正文；格式为：Method Request - URI HTTP - Version CRLF；Method表示请求方法；Request-URI是一个统一资源标识符；HTTP-Version表示请求的HTTP协议版本；CRLF表示回车和换行。
- HTTP响应由三部分组成：状态行，消息报头，响应正文；格式为：HTTP-Version Status-Code Reason-Phrase CRLF；HTTP-Version表示服务器HTTP协议的版本，Status-Code表示服务器发回的响应状态代码，Reason-Phrase表示状态代码的文本描述。

(1) HTTP请求报文

一个HTTP请求报文由请求行（request line）、请求头部（request header）、空行和请求数据4个部分构成。



图：请求报文

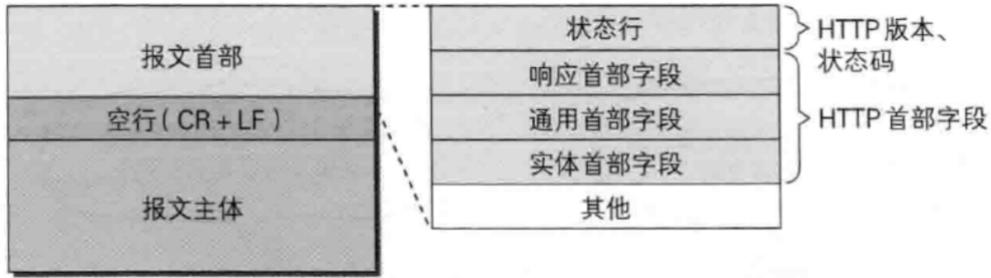
- **请求行**，给出了请求类型，URI给出请求的资源位置(/)。HTTP中的请求类型包括
 - 请求行数据格式由三个部分组成：请求方法、URI、HTTP协议版本，他们之间用空格分隔。
- **请求头**，主要是用于说明请求源、连接类型、以及一些Cookie信息等
 - 请求头部紧跟着请求行，该部分主要是用于描述请求正文。

请求头	说明
Host	接受请求的服务器地址，如IP:端口号、域名
Connection	连接的相关属性
User-Agent	发送请求的应用名称
Accept	通知服务端可发送的编码格式
Accept-Encoding	通知服务端可发送的数据压缩格式
Accept-Language	通知服务端可发送的语言

- **请求正文**，一般用于存放POST请求类型的请求正文

(2) HTTP响应报文

HTTP响应报文由状态行（HTTP版本、状态码（数字和原因短语））、响应头部、空行和响应体4个部分构成。



图：响应报文

- **状态行**, 主要给出响应HTTP协议的版本号、响应返回状态码、响应描述, 同样是单行显示。
 - 1XX 请求正在处理
 - 2XX 请求成功
 - 302 Found 资源的URI已临时定位到其他位置, 客户端不会更新URI。
 - 303 See Other 资源的URI已更新, 明确表示客户端要使用GET方法获取资源。
 - 304 Not Modified 当客户端附带条件请求访问资源时资源已找到但未符合条件请求。
 - 400 请求报文中存在语法错误
 - 403 对请求资源的访问被服务器拒绝
 - 404 请求资源不存在
 - 405 请求的方式不支持
 - 5XX 服务器错误
 - 503 服务器暂时处于超负载状态或正在进行停机维护
- **响应头部**, 与响应头部类似, 主要是返回一些服务器的基本信息, 以及一些Cookie值等
- **响应体**, 为请求需要得到的具体数据, 可以为任何类型数据, 一般网页浏览返回的为html文件内容

(3) HTTP请求与响应

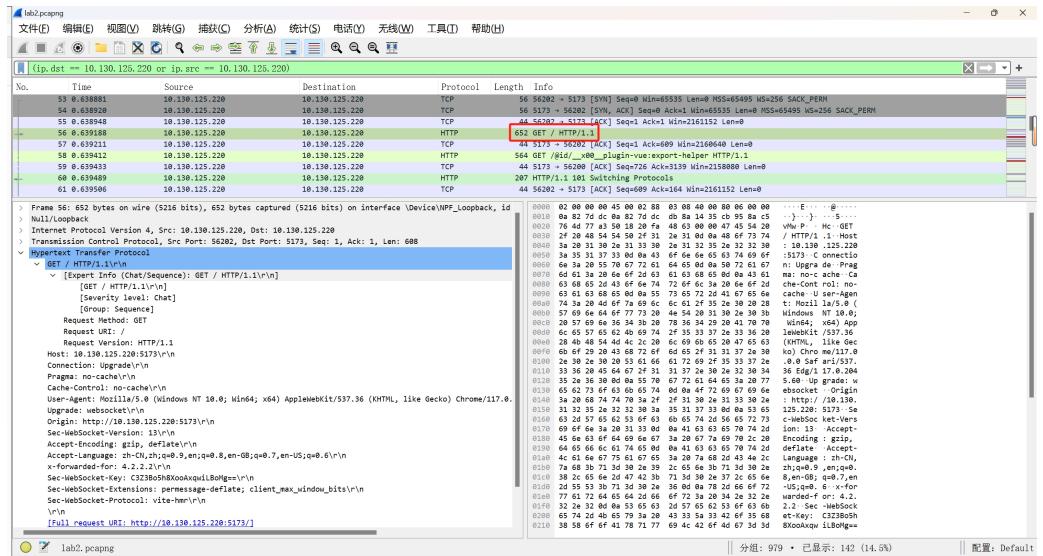
在本次实验数据中, 可以看到在三次握手建立连接与四次挥手关闭连接中间有三次HTTP请求与响应。

序号	源IP	目的IP	协议	内容
53	0.638881	10.130.125.220	TCP	56 56202 → 5173 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
54	0.638920	10.130.125.220	TCP	56 5173 → 56202 [SYN, ACK] Seq=65535 Len=0 MSS=65495 WS=256 SACK_PERM
55	0.638948	10.130.125.220	TCP	44 56202 → 5173 [ACK] Seq=1 Ack=1 Win=2161152 Len=0
56	0.639188	10.130.125.220	HTTP	652 GET / HTTP/1.1
57	0.639211	10.130.125.220	TCP	44 5173 → 56202 [ACK] Seq=1 Ack=609 Win=2160640 Len=0
58	0.639412	10.130.125.220	HTTP	564 GET /index.html?_x000d__x0009_plugin-vue-export-helper HTTP/1.1
59	0.639443	10.130.125.220	TCP	44 5173 → 56202 [ACK] Seq=726 Ack=3139 Win=2158080 Len=0
60	0.639489	10.130.125.220	HTTP	207 HTTP/1.1 101 Switching Protocols
61	0.639506	10.130.125.220	TCP	44 56202 → 5173 [ACK] Seq=609 Ack=164 Win=2161152 Len=0
62	0.639514	10.130.125.220	WebSocket	66 WebSocket Text [FIN]
63	0.639523	10.130.125.220	TCP	44 56202 → 5173 [ACK] Seq=609 Ack=186 Win=2160896 Len=0
64	0.639865	10.130.125.220	HTTP	189 HTTP/1.1 304 Not Modified
65	0.639883	10.130.125.220	TCP	44 56200 → 5173 [ACK] Seq=3139 Ack=871 Win=2160384 Len=0
66	0.691413	10.130.125.220	HTTP	581 GET /favicon.ico HTTP/1.1
67	0.691452	10.130.125.220	TCP	44 5173 → 56200 [ACK] Seq=3671 Ack=3676 Win=2157568 Len=0
68	0.692144	10.130.125.220	HTTP	189 HTTP/1.1 304 Not Modified
69	0.692167	10.130.125.220	TCP	44 56200 → 5173 [ACK] Seq=3676 Ack=1016 Win=2160128 Len=0
336	5.637911	10.130.125.220	TCP	44 5173 → 56201 [FIN, ACK] Seq=146 Ack=498 Win=2160640 Len=0
337	5.637941	10.130.125.220	TCP	44 56201 → 5173 [ACK] Seq=498 Ack=147 Win=2161152 Len=0
338	5.700043	10.130.125.220	TCP	44 5173 → 56200 [FIN, ACK] Seq=1016 Ack=3676 Win=2157568 Len=0
339	5.700066	10.130.125.220	TCP	44 56200 → 5173 [ACK] Seq=3676 Ack=1017 Win=2160128 Len=0

三轮
HTTP
请求与响应

下面以第一轮请求与响应为例进行分析。

客户端向服务器端发送请求



• 请求行

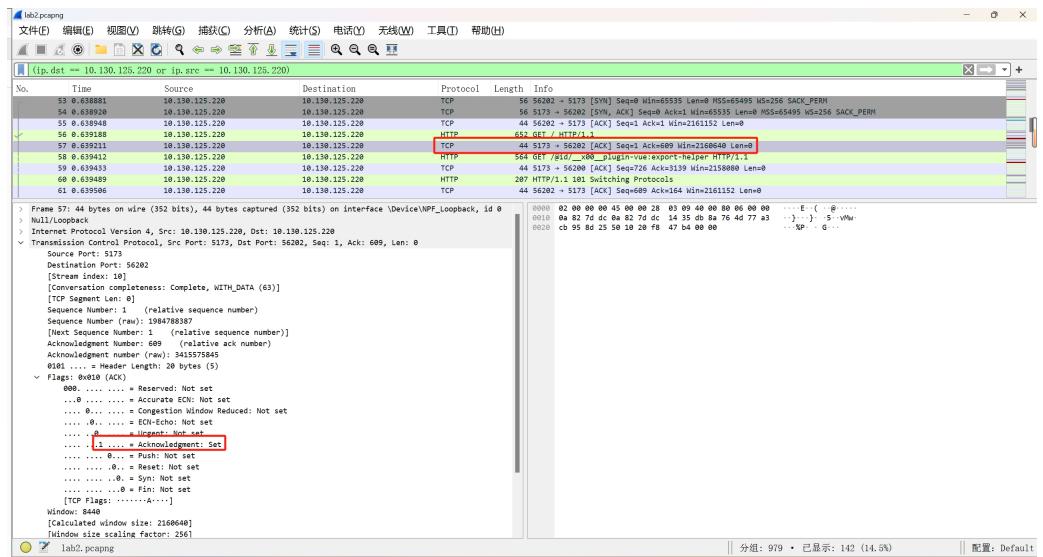
- 请求方式: GET
- 请求URI: /
- 协议版本: HTTP/1.1

• 请求头

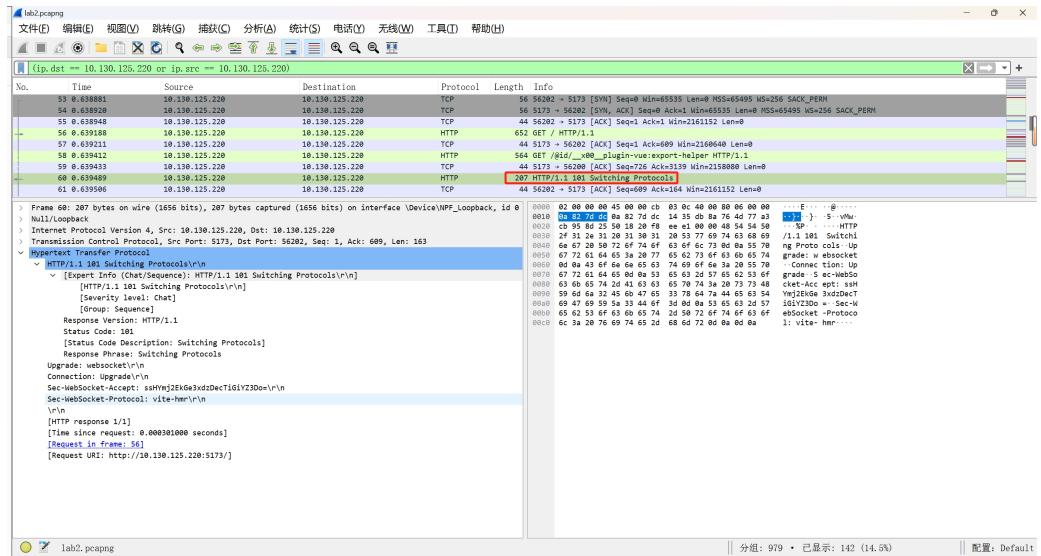
- Host
- Connection
- User-Agent

.....

服务器端回复ACK表示收到请求



服务器端向客户端发送响应报文



• 状态行

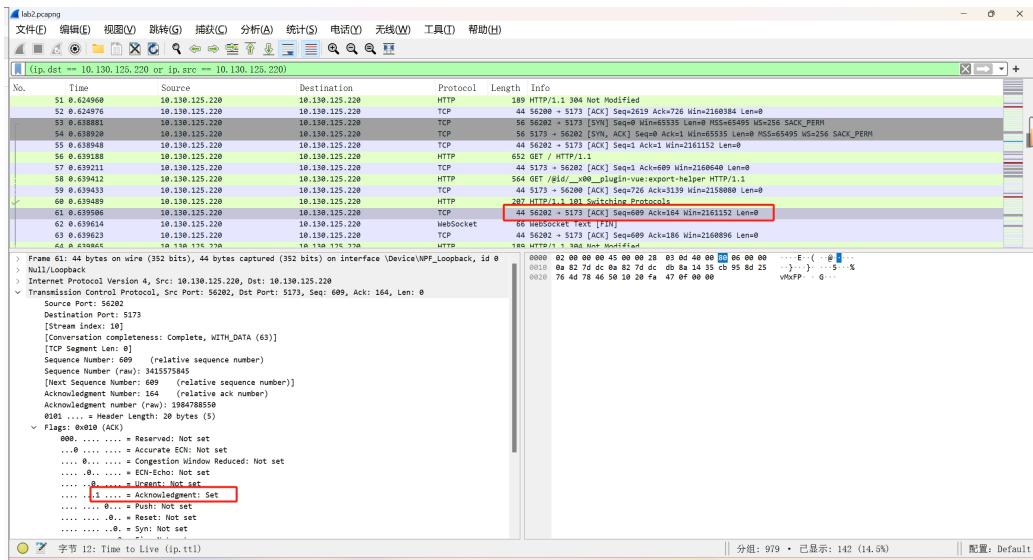
- 协议版本: HTTP/1.1
- 状态码: 101, 表示正在处理
- 状态响应短语: Switching Protocols
 - 这个状态代码通常在WebSocket握手过程中使用。当客户端发送一个带有"Upgrade"头，表明它希望切换到WebSocket协议时，服务器可以用"Switching Protocols"状态代码作为响应。这表示服务器同意切换到WebSocket协议，然后双方可以开始使用WebSocket进行通信。

• 响应头部

- Upgrade
- Connection
- Sec-WebSocket-Accept
- Sec-WebSocket-Protocol

• 响应体

客户端回复ACK表示收到请求



4. TCP四次挥手关闭连接

TCP使用四次挥手关闭连接，客户端和服务端分别释放连接

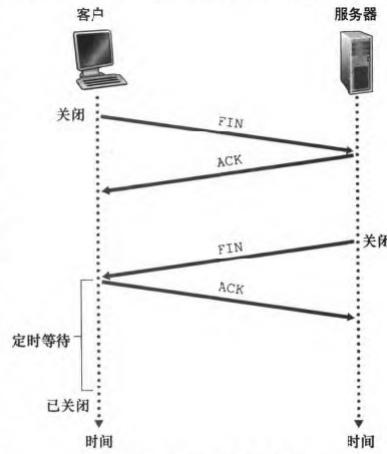
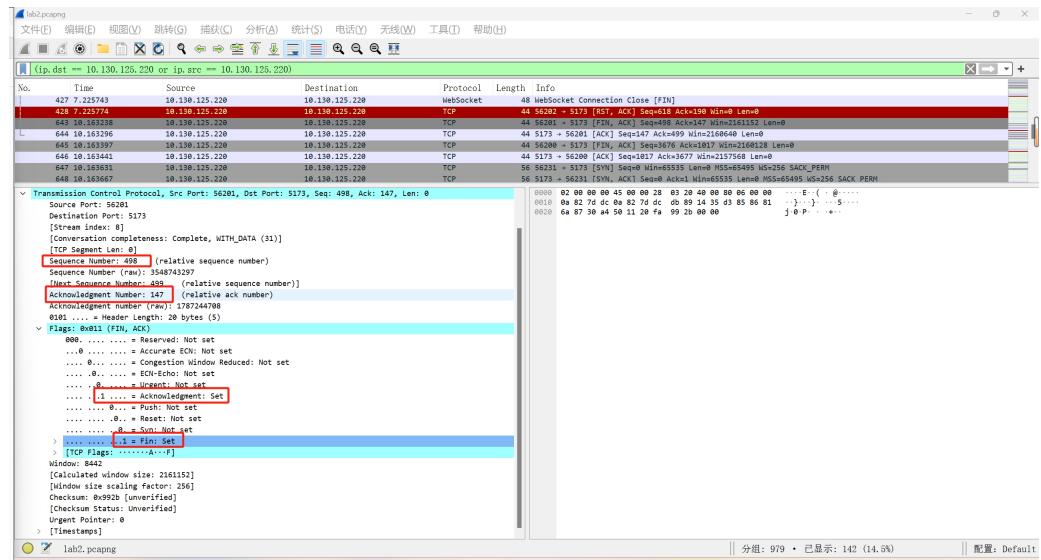


图 3-40 关闭一条 TCP 连接

在wireshark中可以看到四次挥手的过程。

643	10.163238	10.130.125.220	10.130.125.220	TCP	44 56201 + 5173 [FIN, ACK] Seq=498 Ack=147 Win=2161152 Len=0
644	10.163296	10.130.125.220	10.130.125.220	TCP	44 5173 + 56201 [ACK] Seq=147 Ack=499 Win=2160640 Len=0
645	10.163397	10.130.125.220	10.130.125.220	TCP	44 56200 + 5173 [FIN, ACK] Seq=3676 Ack=1017 Win=2160128 Len=0
646	10.163441	10.130.125.220	10.130.125.220	TCP	44 5173 + 56200 [ACK] Seq=1017 Ack=3677 Win=2157568 Len=0

- 第一次挥手：主动关闭方发送一个FIN并进入FIN_WAIT1状态

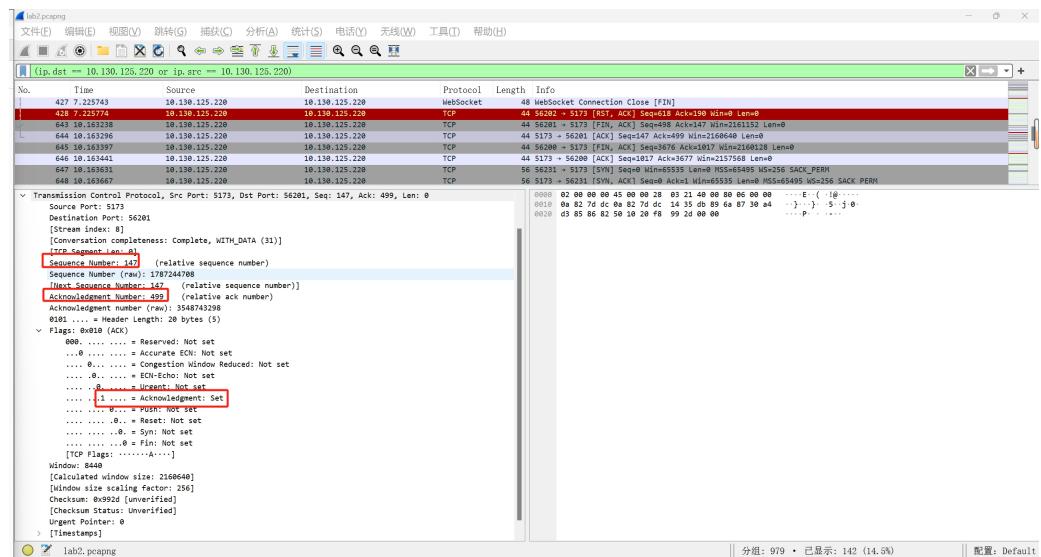


- Source Port: 5173
- Destination Port: 56231
- Seq = 498
- Ack = 147
- ACK、FIN：标志位

○ 客户端向服务器端发送主动关闭连接报文

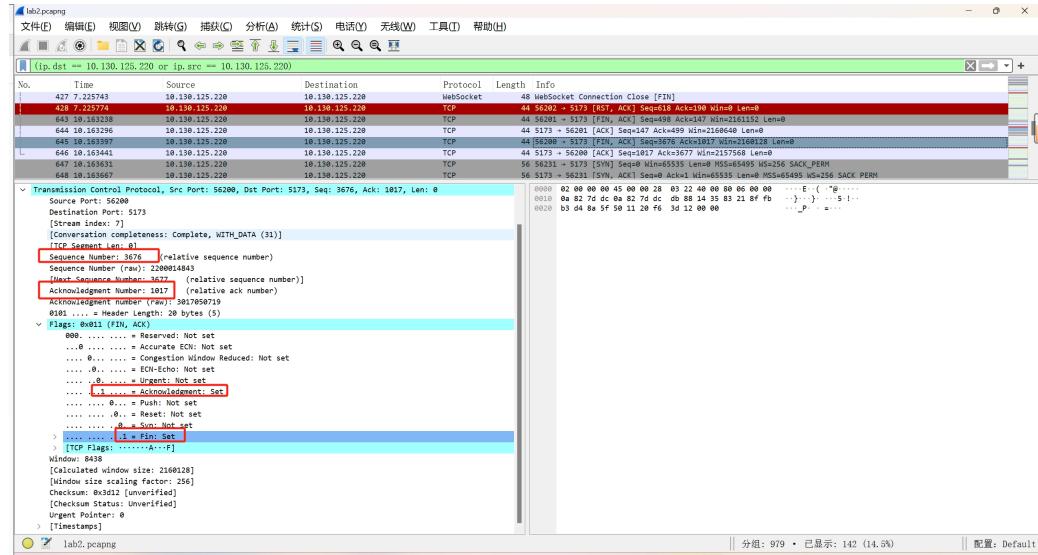
○ 客户端进入FIN_WAIT1状态，等待服务器端回复

- 第二次挥手：被动关闭方接收到主动关闭方发送的FIN并发送ACK，此时被动关闭方进入CLOSE_WAIT状态；主动关闭方收到被动关闭方的ACK后，进入FIN_WAIT2状态



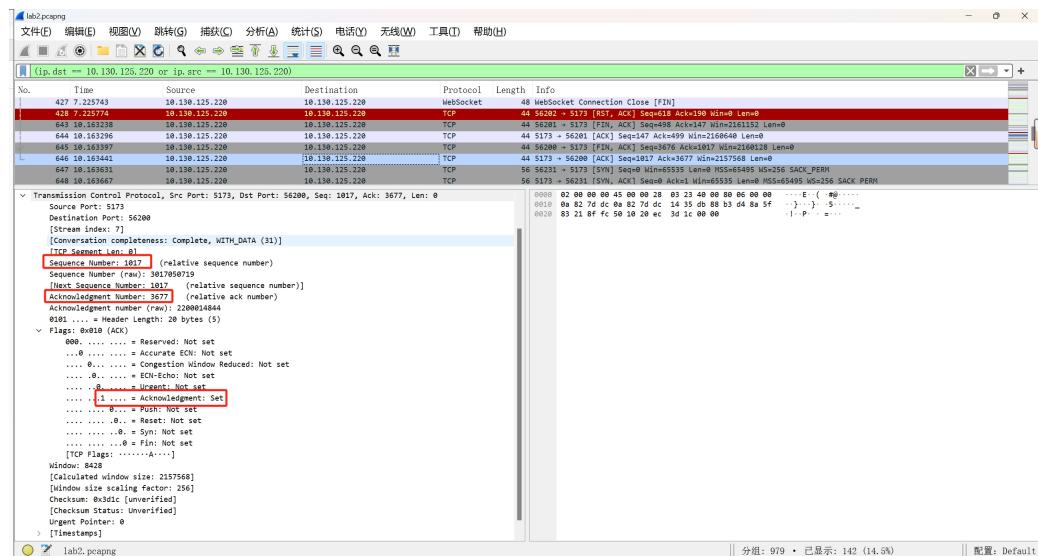
- Source Port: 56231
- Destination Port: 5173
- Seq = 147
- Ack = 499
- ACK：标志位

- 服务器端接收报文，并回复确认报文
- 服务器端进入CLOSE_WAIT状态；客户端收到服务器端的确认报文后进入FIN_WAIT2状态
- 第三次挥手：被动关闭方发送一个FIN并进入LAST_ACK状态



- Source Port: 5173
- Destination Port: 56200
- Seq = 3676
- Ack = 1017
- ACK、FIN：标志位

- 服务器端在确认数据传输完毕后向客户端发送结束报文
- 服务器端进入LAST_ACK状态，等待客户端最后确认
- 第四次挥手：主动关闭方收到被动关闭方发送的FIN并发送ACK，此时主动关闭方进入TIME_WAIT状态，经过2MSL时间后关闭连接；被动关闭方收到主动关闭方的ACK后，关闭连接



- Source Port: 56200
- Destination Port: 5173
- Seq = 1017

- Ack = 3677
- ACK：标志位
- **客户端**回复确认报文，然后进入TIME_WAIT状态，经过2MSL时间后关闭连接（MSL是数据分节在网络中存活的最长时间）
- **服务器端**接收到确认报文后关闭连接

比较主流的文章都说关闭TCP会话是四次挥手，但是实际上为了提高效率通常合并第二、三次的挥手，即三次挥手。