

计算机网络课程实验报告

Lab2：配置Web服务器 分析HTTP交互过程

网络空间安全学院 信息安全-法学双学位班 2210653 王崇宇

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。
- （2）通过浏览器获取自己编写的Web页面。
- （3）使用Wireshark捕获浏览器与Web服务器的交互过程，使用Wireshark过滤器使其只显示HTTP协议。
- （4）现场演示。
- （5）提交HTML文档、Wireshark捕获文件和实验报告，对HTTP交互过程进行详细说明。
- 注：页面不要太复杂，包含所要求的基本信息即可。使用HTTP，不要使用HTTPS。

二、实验分析

- 可以看出我们本次实验的主要任务分为三个部分，其重点在于**分析交互过程**。此前我们需要完成前两个步骤，故我们依次完成。

（一）配置Web服务器

- 网上关于如何配置Web服务器的方法可谓琳琅满目，事实上本次实验仅仅要求我们部署一个本地服务器即可，并且没有限制系统、语言以及框架，我们选择借助Node.js，使用parcel-bundler打包配置Web服务器。

Node.js简介

- Node.js是一个基于Chrome V8引擎的JavaScript运行时环境，专为服务器端开发设计。它的非阻塞、事件驱动的架构使得处理并发任务变得高效，尤其适合实时应用和数据密集的后端服务。Node.js允许开发者在服务器端使用JavaScript代码，常见的用途包括构建Web服务器、API服务以及各种工具。
- Node.js的关键特点包括：
 - 事件驱动、非阻塞I/O：通过异步处理，Node.js在处理高并发时性能出色。
 - 跨平台支持：支持Linux、Windows、macOS，适用于服务器开发。
 - npm生态系统：拥有丰富的第三方包和模块，极大扩展了Node.js的功能。

Parcel-Bundler简介

- Parcel-Bundler是一款快速、零配置的Web应用打包工具，开发体验友好。它可以自动进行代码分割、支持各种类型的资源（JavaScript、CSS、HTML、图片等），而且不需要复杂的配置文件。对于初学者和小型项目来说，Parcel非常便捷，同时也适合现代化的前端开发。
- Parcel的关键特点包括：
 - 零配置：不需要复杂的配置文件即可完成打包。
 - 快速打包：Parcel利用多核处理和文件缓存，加快打包速度。

- 支持多种资源：支持JavaScript、CSS、图片、字体等资源的打包。
- 自动热重载：在开发模式下，修改代码后会自动更新页面，无需手动刷新。
- 代码分割和树摇：优化构建后的包体积，提高应用性能。

其他可用方式简介

- **Flask**
- 简介：Flask是Python开发的一款轻量级Web框架，核心理念是“微框架”，只包含最基本的功能，适合小型或中型项目。它具有较高的灵活性，并支持扩展插件，因此可以根据需求添加额外功能。
- 特点：
 - 轻量级、简单易用：只包含核心模块，无需复杂配置。
 - 可扩展：通过插件或自定义代码实现更复杂的功能（例如数据库、认证、会话等）。
 - 路由和模板支持：使用Jinja2模板和Werkzeug路由系统。
- 适用场景：适合小型Web应用、API开发和快速原型设计。
- **Django**
- 简介：Django是Python的全栈Web框架，以快速开发和清晰的设计为目标。Django内置了许多功能，如ORM（对象关系映射）、认证系统、管理后台等，适合大型和复杂的Web应用开发。
- 特点：
 - 全面功能：自带ORM、认证、会话管理、表单处理、管理后台等。
 - 高度安全：默认包含XSS、SQL注入防护等安全机制。
 - 遵循MVC架构：使用“模型-视图-模板”（MVT）结构，清晰明了。
 - 广泛的社区支持：提供丰富的插件和第三方扩展。
- 适用场景：适合大型Web应用、电子商务网站、内容管理系统（CMS）等。
- **Bottle**
- 简介：Bottle是一个非常小巧的Python Web框架，整个框架只有一个文件。它非常轻量，适合微服务或嵌入式应用，并内置HTTP服务器，但也可以与其他Web服务器一起使用。
- 特点：
 - 单文件实现：仅一个文件，非常易于部署和管理。
 - 无依赖：没有强制依赖，非常轻量。
 - 内置模板和路由：支持基本的路由和简单的模板功能。
- 适用场景：适合构建小型Web应用、微服务API、嵌入式系统。
- **Tornado**
- 简介：Tornado是一个异步的Web框架和网络库，设计初衷是为高并发场景服务。Tornado可以同时处理成千上万的客户端连接，特别适合实时Web服务。
- 特点：
 - 异步支持：内置异步I/O模型，适合高并发和实时应用。
 - WebSocket支持：天然支持WebSocket连接，适合构建实时应用（如聊天系统）。
 - 高性能：单线程高效处理，并发性能出色。
- 适用场景：适合实时Web应用、聊天室、推送服务和需要高并发的应用。
- **Aiohttp**
- 简介：Aiohttp是一个异步HTTP客户端和服务器框架，使用Python的asyncio库进行异步操作，适合高并发Web应用。Aiohttp支持HTTP服务器和客户端，允许开发者在单一代码库中管理异步HTTP请求。
- 特点：
 - 异步/非阻塞：基于asyncio，使用async和await进行异步编程。
 - 内置WebSocket：提供对WebSocket的支持，适合实时通信。
 - 轻量和灵活：比Tornado更轻量化，但同样支持高并发。
- 适用场景：适合异步API、实时Web应用、需要大量HTTP请求的应用。

(二) 制作Web页面

- 我们的主要任务是制作简单的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。我们需要编写一个html文件来实现这一点。
- 随后我们需要通过浏览器获取自己编写的Web页面。

(三) 获取Web页面

- 我们使用HTTP，打开浏览器输入`http://127.0.0.1:8080`得到编写的页面。

(四) 分析交互过程

- Wireshark的抓包界面主要分为三个面板：**数据包列表 (Packet List)** 面板、**数据包详情 (Packet Details)** 面板、**数据包字节 (Packet Bytes)** 面板，分别是捕获到的数据包、以层次结构显示数据包详情、以十六进制和ASCII编码显示的数据包原始字节。
- 对于**数据包列表面板**又分为以下层次：
 - **Frame**：物理层数据帧概况。
 - **链路层**：这一部分显示因情况而不同。
 - **Internet Protocol Version 4**：互联网层IP包头部信息。
 - **Transmission Control Protocol**：传输层的数据段头部信息，我们本次实验使用的是TCP。
 - **Hypertext Transfer Protocol**：应用层的信息，本次实验是HTTP协议。
- 此外，主要是TCP协议的相关知识，本部分不再做理论叙述，具体将结合实验结果进行分析。

三、实验具体过程

(一) 配置Web服务器

- 本项目借助Node.js，使用Parcel-Bundler打包配置Web服务器。
- 我们首先需要从官网下载Node.js，并按照相应要求配置npm以及Parcel-Bundler并将其配置到环境变量中以便使用相应指令。

node_modules	2024/10/29 14:41
CHANGELOG.md	2024/7/8 19:33
corepack	2023/6/23 14:01
corepack.cmd	2023/6/23 14:01
install_tools.bat	2023/6/23 14:02
LICENSE	2024/7/8 19:33
node.exe	2024/7/8 19:57
node_etw_provider.man	2024/7/8 19:33
nodevars.bat	2023/6/23 14:02
npm	2024/5/2 22:47
npm.cmd	2024/5/2 22:47
npx	2024/5/2 22:47
npx.cmd	2024/5/2 22:47
README.md	2024/7/8 19:33

- 随后我们使用命令行，输入`npm start`指令，可以看到项目得以成功运行。

```
c:\npm start
Microsoft Windows [版本 10.0.19044.2006]
(c) Microsoft Corporation。保留所有权利。
D:\1. 大三上资料\工学\计算机网络\实验\实验 (wcy)\lab2\wangchongyu_web>npm start
> wangchongyu_web@1.0.0 start
> node ./node_modules/parcel-bundler/bin/cli serve ./src/index.html --port 8080 --host 0.0.0.0
Server running at http://0.0.0.0:8080
✓ Built in 579ms.
```

(二) 制作Web页面

- 在给定框架的基础上，我们对其内容进行删改，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。

```html

```
<!DOCTYPE html>
<html lang="en">
 <head>
 <meta charset="UTF-8">
 <title>MY FIRST WEB SERVER</title>
 </head>
 <body>
 <h1>
 MY FIRST WEB SERVER
 </h1>
 <div class="container">
 <div class="info">
 <p>
 王崇宇
 </p>
 <p>
 2210653
 </p>
 <p>
 信息安全-法学双学位班
 </p>

 Cilck here for more details!

 <audio controls="">
 <source src="/introduction.77444568.m4a" type="audio/mpeg">
 Here is my self-introduction.
 </audio>
 </div>
 </div>
 <div class="logo">

 </div>
 </body>
</html>
```

## (三) 获取Web页面

- 我们使用HTTP，打开浏览器输入`http://127.0.0.1:8080`得到编写的页面。



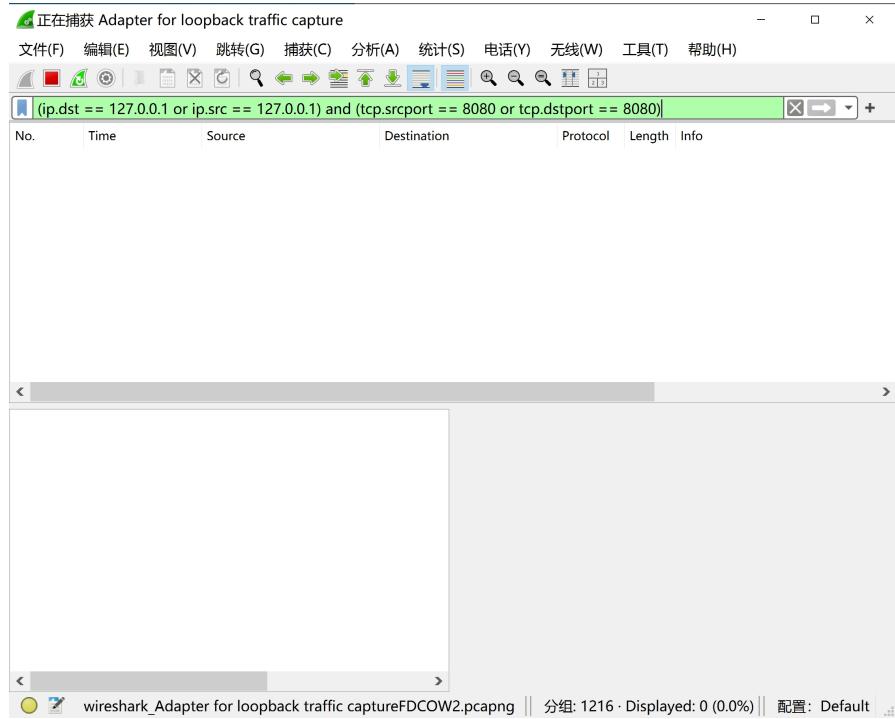
## (四) 分析交互过程

### (一) 准备工作

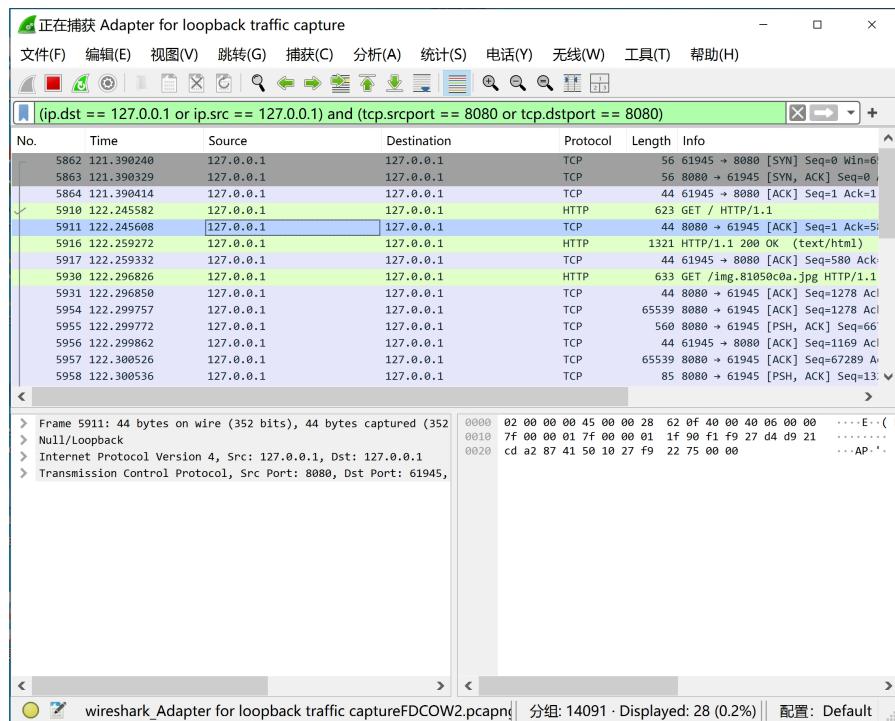
- 首先我们需要下载安装Wireshark抓包软件。
- 随后启动软件，选择**Adapter for loopback traffic capture**接口。这是由于我们建立的是本地服务器，并且使用的是本地浏览器访问
- 按照前面创建的设定设置过滤器  
`(ip.dst == 127.0.0.1 or ip.src == 127.0.0.1) and (tcp.srcport == 8080 or tcp.dstport == 8080)`。
- 若我们设置过滤器  
`(ip.dst == 127.0.0.1 or ip.src == 127.0.0.1) and (tcp.srcport == 8080 or tcp.dstport == 8080) and http`，这样能够确保我们监听的端口正确且使用Wireshark过滤器使其只显示HTTP协议，这样便满足了实验的要求。

### (二) 交互过程捕获

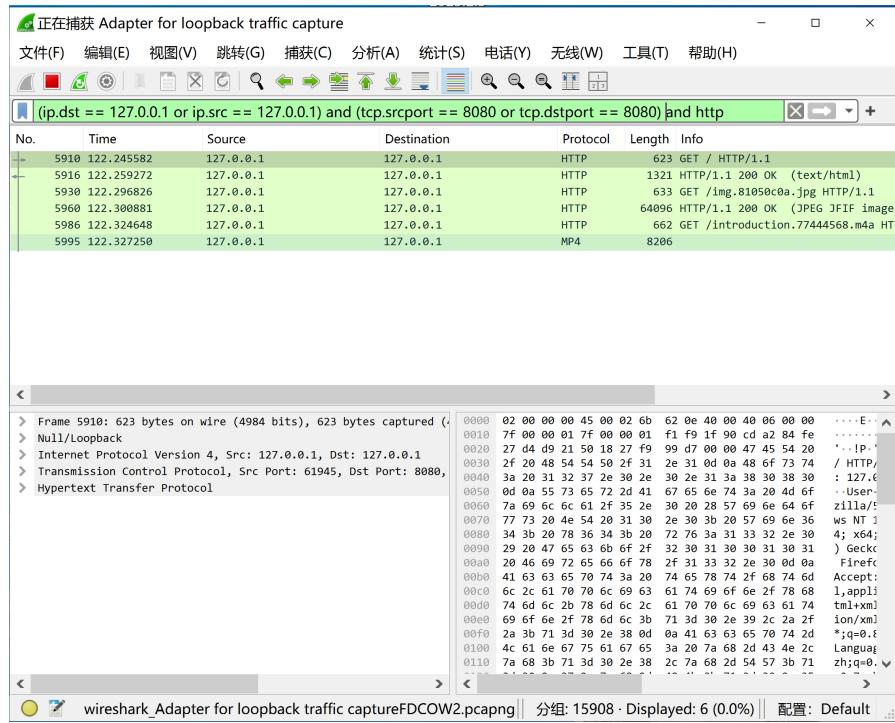
- 我们按照上述内容进行设定，可以看到开始运行前并无捕获数据包。



- 按照上面设定，我们打开Web并运行一段时间，得到如下的抓包结果。



- 随后我们加入http条件进行进一步过滤，可以得到如下的结果。



### (三) 具体过程分析

- 下面，我将结合具体的截图，参考理论课的相关知识，对于交互过程进行具体分析

#### 1、TCP报文段格式

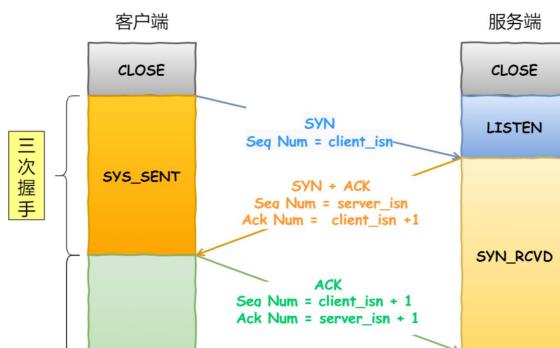
- 下面我将简要介绍一下TCP报文段格式：
- 源端口号** 发送方的TCP端口号，16bits
- 目的端口号** 接收方的TCP端口号，16bits
- 发送序号** 数据部分第一个字节的序列号，32bits
- 确认序号** 期望接收的下一个报文段的数据的第一个字节的序列号，32bits
- 头长度** 指出报文数据距TCP报头的起始处有多远(TCP报文头长度)，4bits
- 未用** 目前置零，6bits【事实上后两位用于显式拥塞通知】
- 标志位**
  - U (URG)**：紧急比特，1bit；当 URG=1 时，表明紧急指针字段有效。它告诉系统此报文段中有紧急数据，应尽快传送(相当于高优先级的数据)。
  - A (ACK)**：确认比特，1bit；只有当 ACK=1 时确认号字段才有，当 ACK=0 时，确认号无效。
  - P (PSH)**：推送比特，1bit；接收方 TCP 收到推送比特置1的报文段，就尽快地交付给接收应用进程，而不再等到整个缓存都填满了后再向上交付。
  - R (RST)**：复位比特，1bit；当 RST=1 时，表明TCP连接中出现严重差错(如由于主机崩溃或其他原因)，必须释放连接，然后再重新建立运输连接。
  - S (SYN)**：同步比特，1bit；同步比特 SYN 置为 1，就表示这是一个连接请求或连接接受报文。
  - F (FIN)**：终止比特，1bit；用来释放一个连接。当 FIN=1 时，表明此报文段的发送端的数据已发送完毕，并要求释放运输连接。
- 接收窗口通告** 窗口字段用来控制对方发送的数据量，单位为字节。TCP 连接的一端根据设置的缓存空间大小确定自己的接收窗口大小，然后通知对方以确定对方的发送窗口的上限，16bits
- 校验和** 校验和字段检验的范围包括首部、数据以及伪头部这三部分。在计算校验和时，要在 TCP 报文段的前面加上 12 字节的伪首部，16bits
- 紧急数据指针字** 紧急指针指出在本报文段中的紧急数据的最后一个字节的序号，仅当 URG=1 时有效，16bits

- **选项字段** 长度可变，包括窗口缩放选项、MSS、时间戳等
- **数据** 应用层进程提交的数据



## 2、TCP三次握手建立连接

- TCP协议是面向连接的（connection-oriented），是因为在一个应用进程向另一个应用进程发送数据之前，这两个进程必须先进行相互“握手”，即它们必须**相互发送某些预备报文段**，以建立确保数据传输的参数。
- 我们熟知的**三次握手**就是TCP建立连接的过程中客户端和服务器端之间交换的三个TCP报文段，主要目的是：
  - 确认存在：即客户端和服务器端知晓对方已在线。
  - 协商参数：即保证数据传输的一些参数。
  - 分配资源：即分配实体资源保证数据传输。
- 在这里对两个概念进行说明：
  - **客户端**：主动发起TCP建连请求的应用进程。
  - **服务器端**：被动接收TCP建连请求的应用进程。



- 我们可以看到TCP使用三次握手建立连接的过程

1722 29.477629	127.0.0.1	127.0.0.1	TCP	56 63940 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
1723 29.477680	127.0.0.1	127.0.0.1	TCP	56 8080 → 63940 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
1724 29.477782	127.0.0.1	127.0.0.1	TCP	44 63940 → 8080 [ACK] Seq=1 Ack=1 Win=2619648 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。
  - 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，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。

- Source Port: 63940.
- Destination Port: 8080.
- 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。
- 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

```

> Frame 1722: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
 Transmission Control Protocol, Src Port: 63940, Dst Port: 8080, Seq: 0, Len: 0
 Source Port: 63940
 Destination Port: 8080
 [Stream index: 88]
 [Stream Packet Number: 1]
 [Conversation completeness: Complete, WITH_DATA (31)]
 [TCP Segment Len: 0]
 Sequence Number: 0 (relative sequence number)
 Sequence Number (raw): 791927224
 [Next Sequence Number: 1 (relative sequence number)]
 Acknowledgment Number: 0
 Acknowledgment number (raw): 0
 1000 = Header Length: 32 bytes (8)
 Flags: 0x0002 (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
 0... = Syn: Set
 0..0 = Fin: Not set
 [TCP Flags:S.]
 Window: 65535
 [Calculated window size: 65535]
 Checksum: 0x5d49 [unverified]
 [Checksum Status: Unverified]

```

## • 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。
  - 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。
  - Source Port: 8080。
  - Destination Port: 63940。
  - Seq = 0：初始建立值为0，表示当前还没有发送数据。
  - 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。

```

> Frame 1723: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
 Transmission Control Protocol, Src Port: 8080, Dst Port: 63940, Seq: 0, Ack: 1, Len: 0
 Source Port: 8080
 Destination Port: 63940
 [Stream index: 88]
 [Stream Packet Number: 2]
 [Conversation completeness: Complete, WITH_DATA (31)]
 [TCP Segment Len: 0]
 Sequence Number: 0 (relative sequence number)
 Sequence Number (raw): 3018862352
 [Next Sequence Number: 1 (relative sequence number)]
 Acknowledgment Number: 1 (relative ack number)
 Acknowledgment number (raw): 791927225
 1000 = Header Length: 32 bytes (8)
 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.]
 Window: 65535
 [Calculated window size: 65535]
 Checksum: 0x7197 [unverified]

```

### • 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。
  - 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，自己的发功能正常。至此服务成功建立。
    - Source Port: 63940。
    - Destination Port: 8080。
    - Seq = 1：表示当前已经发送1个数据。
    - Ack = 1：表示当前端成功接收的数据位数。
    - ACK：标志位，表示已经收到记录。

```

> Frame 1724: 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: 127.0.0.1, Dst: 127.0.0.1
 Transmission Control Protocol, Src Port: 63940, Dst Port: 8080, Seq: 1, Ack: 1, Len: 0
 Source Port: 63940
 Destination Port: 8080
 [Stream index: 88]
 [Stream Packet Number: 3]
 [Conversation completeness: Complete, WITH_DATA (31)]
 [TCP Segment Len: 0]
 Sequence Number: 1 (relative sequence number)
 Sequence Number (raw): 791927225
 [Next Sequence Number: 1 (relative sequence number)]
 Acknowledgment Number: 1 (relative ack number)
 Acknowledgment number (raw): 3018862353
 0101 = Header Length: 20 bytes (5)
 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....]
 Window: 10233
 [Calculated window size: 2619648]
 [Window size scaling factor: 256]

```

## 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。

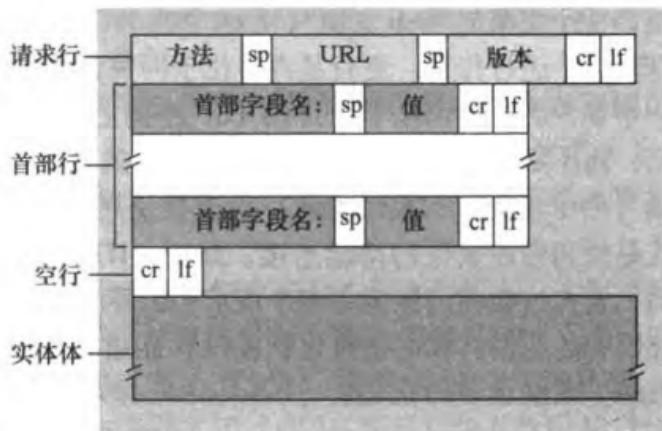
lab2						
1764 30.153366	127.0.0.1	127.0.0.1	HTTP	537 GET / HTTP/1.1		
1765 30.153393	127.0.0.1	127.0.0.1	TCP	44 8080 → 63940 [ACK] Seq=1 Ack=494 Win=2619648 Len=0		
1770 30.155830	127.0.0.1	127.0.0.1	HTTP	1321 HTTP/1.1 200 OK (text/html)		
1771 30.155872	127.0.0.1	127.0.0.1	TCP	44 63940 → 8080 [ACK] Seq=494 Ack=1278 Win=2618368 Len=0		
1786 30.192948	127.0.0.1	127.0.0.1	HTTP	545 GET /img_8105c0a.jpg HTTP/1.1		
1787 30.192987	127.0.0.1	127.0.0.1	TCP	44 8080 → 63940 [ACK] Seq=1278 Ack=995 Win=2619136 Len=0		
1804 30.196637	127.0.0.1	127.0.0.1	TCP	65539 8080 → 63940 [ACK] Seq=1278 Ack=995 Win=2619136 Len=65495 [TCP PDU reassembled in 1819]		
1805 30.196651	127.0.0.1	127.0.0.1	TCP	560 8080 → 63940 [PSH, ACK] Seq=66773 Ack=995 Win=2619136 Len=516 [TCP PDU reassembled in 1819]		
1806 30.196728	127.0.0.1	127.0.0.1	TCP	44 63940 → 8080 [ACK] Seq=995 Ack=67289 Win=2619648 Len=0		
1809 30.197228	127.0.0.1	127.0.0.1	TCP	65539 8080 → 63940 [ACK] Seq=67289 Ack=995 Win=2619136 Len=65495 [TCP PDU reassembled in 1819]		
1810 30.197238	127.0.0.1	127.0.0.1	TCP	85 8080 → 63940 [PSH, ACK] Seq=132784 Ack=995 Win=2619136 Len=41 [TCP PDU reassembled in 1819]		
1811 30.197300	127.0.0.1	127.0.0.1	TCP	44 63940 → 8080 [ACK] Seq=995 Ack=132825 Win=2619648 Len=0		
1819 30.197820	127.0.0.1	127.0.0.1	HTTP	64096 HTTP/1.1 200 OK (JPEG JFIF image)		

## 4、HTTP报文段格式

- HTTP报文：**HTTP 报文是在应用程序之间发送的数据块,这些数据块将通过以文本形式的元信息开头，用于HTTP 协议交互。请求端(客户端)的 HTTP 报文叫做**请求报文**，响应端(服务器端)的叫做**响应报文**。
- 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。
  - CRLF表示回车和换行。
- 具体过程。
  - 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；
  - 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。

### (1) HTTP请求报文

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



- 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
  - 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。

- 方法字段：GET、POST、HEAD、PUT、DELETE等。
- URL字段：统一资源定位器，带有请求对象的标识，<协议>://<主机>:<端口>/<路径>。
- HTTP版本字段。
- **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。
  - 包含若干个属性，格式为“属性名:属性值”，服务端据此获取客户端的信息。与缓存相关的规则信息，均包含在 header 中，请求头可大致分为四种类型：通用首部字段、请求首部字段、响应首部字段、实体首部字段。

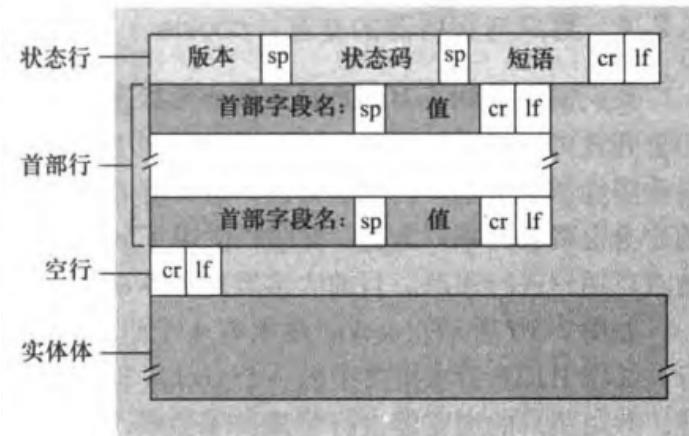
请求头	描述
Host	主机ip地址或域名 (接受请求的服务器地址)
User-Agent	客户端相关信息，如操作系统、浏览器等信息 (发出请求的应用程序名称)
Accept	指定客户端接收信息类型，如：image/jpg, text/html, application/json
Accept-Charset	客户端接受的字符集，如gb2312、iso-8859-1
Accept-Encoding	可接受的内容编码，如gzip
Accept-Language	接受的语言，如Accept-Language:zh-cn
Authorization	客户端提供给服务端，进行权限认证的信息
Cookie	携带的cookie信息
Referer	当前文档的URL，即从哪个链接过来的
Content-Type	请求体内容类型，如Content-Type: application/x-www-form-urlencoded
Content-Length	数据长度
Cache-Control	缓存机制，如Cache-Control:no-cache
Pragma	防止页面被缓存，和Cache-Control:no-cache作用一样

CSDN @Ysning88

- **请求正文**，一般用于存放POST请求类型的请求正文，可以是多种类型：图片、音频、视频等。

## (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。



- **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。

响应头	描述
Server	HTTP服务器的软件信息
Date	响应报文的时间
Expires	指定缓存过期时间
Set-Cookie	设置Cookie
Last-Modified	资源最后修改时间
Content-Type	响应的类型和字符集，如：Content-Type: text/html; charset=utf-8
Content-Length	内容长度
Connection	如Keep-Alive，表示保持tcp连接不关闭，不会永久保持连接，服务器可设置
Location	指明重定向的位置，新的URL地址，如304的情况

CSDN @Ysming88

- 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。

```
— GET /somedir/page.html HTTP/1.1
 Host: www.someschool.edu
 Connection: close
 User-agent: Mozilla/5.0
 Accept-language: fr
```

```
→HTTP/1.1 200 OK
 Connection: close
 Date: Tue, 18 Aug 2015 15:44:04 GMT
 Server: Apache/2.2.3 (CentOS)
 Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
 Content-Length: 6821
 Content-Type: text/html

— (data data data data data ...)
```

### (3) HTTP请求与响应

- 根据实验要求，我们编写简单网页中包含文本、图片和音频三种文件类型，因此我们在本次实验中会出现三轮HTTP请求应答，三轮的请求应答整体类似。

No.	Time	Source	Destination	Protocol	Length	Info
1764	30.153366	127.0.0.1	127.0.0.1	HTTP	537	GET / HTTP/1.1
1765	30.153393	127.0.0.1	127.0.0.1	TCP	44	8080 → 63940 [ACK] Seq=1 Ack=494 Win=2619648 Len=0
1770	30.155830	127.0.0.1	127.0.0.1	HTTP	1321	HTTP/1.1 200 OK (text/html)
1771	30.155872	127.0.0.1	127.0.0.1	TCP	44	63940 → 8880 [ACK] Seq=494 Ack=1278 Win=2618368 Len=0
1786	30.192948	127.0.0.1	127.0.0.1	HTTP	545	GET /img/81905c0a.jpg HTTP/1.1
1787	30.192987	127.0.0.1	127.0.0.1	TCP	44	8080 → 63940 [ACK] Seq=1278 Ack=995 Win=2619136 Len=0
1804	30.196637	127.0.0.1	127.0.0.1	TCP	65539	8080 → 63940 [ACK] Seq=1278 Ack=995 Win=2619136 Len=65495 [TCP PDU reassembled in 1819]
1805	30.196651	127.0.0.1	127.0.0.1	TCP	560	8080 → 63940 [PSH, ACK] Seq=66773 Ack=995 Win=2619136 Len=16 [TCP PDU reassembled in 1819]
1806	30.196728	127.0.0.1	127.0.0.1	TCP	44	63940 → 8080 [ACK] Seq=995 Ack=67289 Win=2619648 Len=0
1809	30.197228	127.0.0.1	127.0.0.1	TCP	65539	8080 → 63940 [ACK] Seq=67289 Ack=995 Win=65495 [TCP PDU reassembled in 1819]
1810	30.197238	127.0.0.1	127.0.0.1	TCP	585	8080 → 63940 [PSH, ACK] Seq=13278 Ack=995 Win=2619136 Len=41 [TCP PDU reassembled in 1819]
1811	30.197300	127.0.0.1	127.0.0.1	TCP	44	63940 → 8080 [ACK] Seq=995 Ack=132825 Win=2619648 Len=0
1819	30.197820	127.0.0.1	127.0.0.1	HTTP	64096	HTTP/1.1 200 OK (JPEG/JFIF image)
1821	30.197895	127.0.0.1	127.0.0.1	TCP	44	63940 → 8080 [ACK] Seq=995 Ack=196877 Win=2555648 Len=0
1846	30.227992	127.0.0.1	127.0.0.1	HTTP	537	GET /favicon.ico HTTP/1.1
1847	30.228026	127.0.0.1	127.0.0.1	TCP	44	8080 → 63940 [ACK] Seq=196877 Ack=1488 Win=2618624 Len=0
1870	30.228963	127.0.0.1	127.0.0.1	TCP	56	63941 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
1871	30.229009	127.0.0.1	127.0.0.1	TCP	44	8080 → 63941 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
1872	30.229067	127.0.0.1	127.0.0.1	TCP	44	63941 → 8080 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
1875	30.229228	127.0.0.1	127.0.0.1	HTTP	574	GET /introduction.7744568.m4a HTTP/1.1
1876	30.229254	127.0.0.1	127.0.0.1	TCP	44	8080 → 63941 [ACK] Seq=1 Ack=531 Win=2619648 Len=0
1883	30.229495	127.0.0.1	127.0.0.1	TCP	44	63940 → 8080 [FIN, ACK] Seq=1488 Ack=196877 Win=2555648 Len=0
1884	30.229518	127.0.0.1	127.0.0.1	TCP	44	8080 → 63940 [ACK] Seq=196877 Ack=1489 Win=2618624 Len=0
1891	30.231639	127.0.0.1	127.0.0.1	TCP	44	8080 → 63940 [FIN, ACK] Seq=196877 Ack=1489 Win=2618624 Len=0
1892	30.231688	127.0.0.1	127.0.0.1	TCP	44	63940 → 8080 [ACK] Seq=1489 Ack=196878 Win=2555648 Len=0
1893	30.234272	127.0.0.1	127.0.0.1	TCP	65539	8080 → 63941 [ACK] Seq=1 Ack=531 Win=2619648 Len=65495 [TCP PDU reassembled in 1896]
1894	30.234409	127.0.0.1	127.0.0.1	TCP	607	8080 → 63941 [PSH, ACK] Seq=65494 Ack=531 Win=2619648 Len=563 [TCP PDU reassembled in 1896]
1895	30.234595	127.0.0.1	127.0.0.1	TCP	44	63941 → 8080 [ACK] Seq=531 Ack=66859 Win=2619648 Len=0
1896	30.235723	127.0.0.1	127.0.0.1	MP4	8206	
1897	30.235924	127.0.0.1	127.0.0.1	TCP	44	63941 → 8080 [ACK] Seq=531 Ack=74221 Win=2611456 Len=0
2360	30.252460	127.0.0.1	127.0.0.1	TCP	44	8080 → 63941 [FIN, ACK] Seq=74221 Ack=531 Win=2619648 Len=0
2361	35.252507	127.0.0.1	127.0.0.1	TCP	44	63941 → 8080 [ACK] Seq=531 Ack=74222 Win=2611456 Len=0

- 下面以第一轮请求与响应为例进行分析。

### 1. 客户端向服务器端发送请求

#### • 请求行

- 方法字段：GET

- URL：/

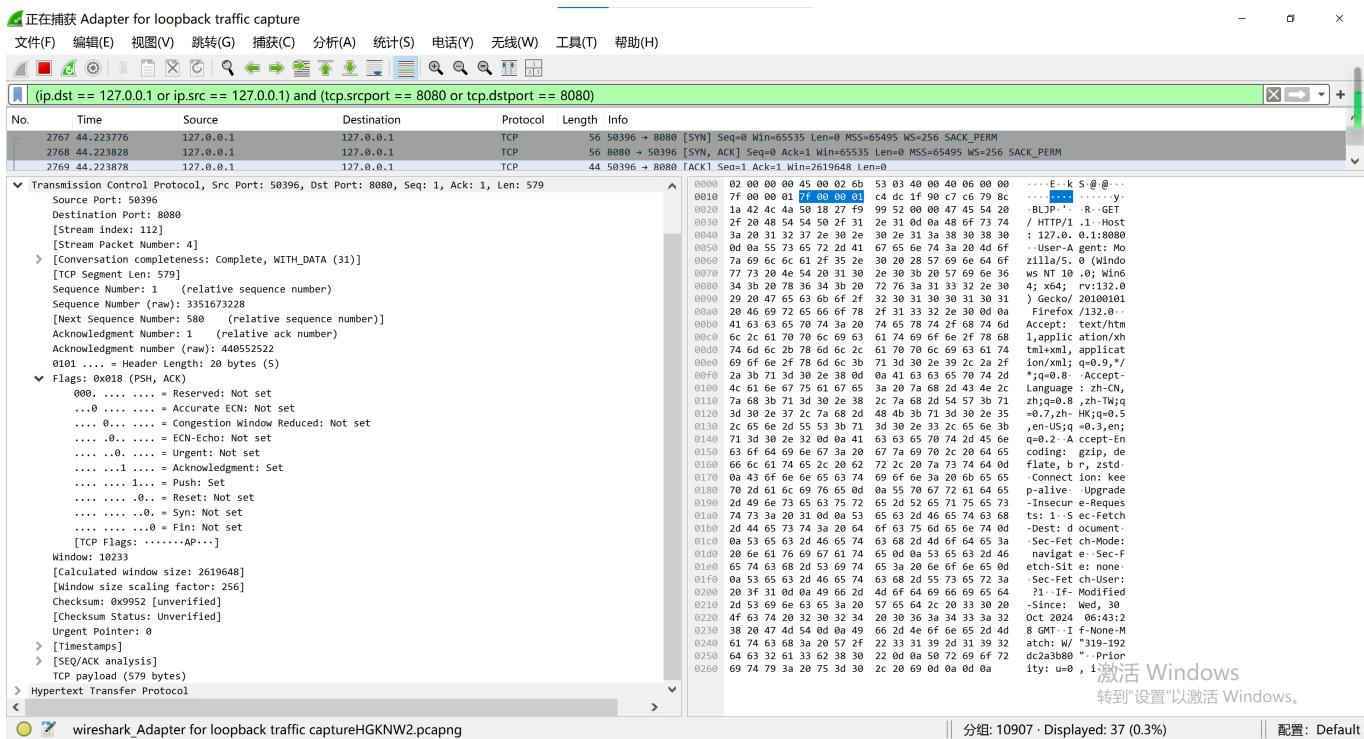
## ◦ HTTP版本字段：HTTP/1.1

### • 请求头

- Host: 主机名——127.0.0.1:8080
- Connection: keep-alive——客户端与服务器端的请求响应保持连接。
- sec-ch-ua: 在以逗号分隔的列表中提供与浏览器关联的每个品牌的品牌和重要版本。
- Accept: 用户可以识别的类型列表。
- Accept-Encoding: 客户可识别的数据编码。
- Accept-Language: 浏览器支持的语言类型。

### • 关于请求报文的TCP报文段标志位分析：PSH、ACK。

- HTTP报文被分为多段，在使用TCP传输时，应用层会出于某些原因（超出MSS）使得一个报文段不能包含所有的协议数据单元（PDU），那么将会把一个完整的消息拆分为多段，每一段（除最有一段外）将会打上标签**TCP segment of a reassembled PDU**。
- 我们解释PSH的作用：当两个应用程序进行交互式的通信时，有时在一端的应用进程希望在键入一个命令后立即就能够收到对方的响应。在这种情况下，TCP可以使用推送(push)操作。这时，发送端TCP将推送比特PSH置为1，并立即创建一个报文段发送出去。



## 2. 服务器端回复ACK表示收到请求

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(T) 帮助(H)

(ip.dst == 127.0.0.1 or ip.src == 127.0.0.1) and (tcp.srcport == 8080 or tcp.dstport == 8080)

No.	Time	Source	Destination	Protocol	Length	Info
2767	44.223776	127.0.0.1	127.0.0.1	TCP	56	50396 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
2768	44.223828	127.0.0.1	127.0.0.1	TCP	56	8080 → 50396 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
2769	44.223878	127.0.0.1	127.0.0.1	TCP	44	50396 → 8080 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
2838	44.342684	127.0.0.1	127.0.0.1	HTTP	623	GET / HTTP/1.1
2839	44.342110	127.0.0.1	127.0.0.1	TCP	44	8080 → 50396 [ACK] Seq=1 Ack=580 Win=2619648 Len=0

▼ Transmission Control Protocol, Src Port: 8080, Dst Port: 50396, Seq: 1, Ack: 580, Len: 0

Source Port: 8080  
Destination Port: 50396  
[Stream Index: 112]  
[Stream Packet Number: 5]  
[Conversation completeness: Complete, WITH\_DATA (31)]  
[TCP Segment Len: 0]  
Sequence Number: 1 (relative sequence number)  
Sequence Number (raw): 440952522  
[Next Sequence Number: 1 (relative sequence number)]  
Acknowledgment Number: 580 (relative ack number)  
Acknowledgment number (raw): 351673807  
0101 .... = Header Length: 20 bytes (5)  
▼ 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  
....0.... .... = Push: Not set  
....0.... .... = Reset: Not set  
....0.... .... = Syn: Not set  
....0.... .... = Fin: Not set  
[TCP Flags: .....A.....]  
Window: 10233  
[Calculated window size: 2619648]  
[Window size scaling factor: 256]  
Checksum: 0xfcb9 [unverified]  
[Checksum Status: Unverified]  
Urgent Pointer: 0  
> [Timestamps]  
> [SEQ/ACK analysis]

激活 Windows  
转到“设置”以激活 Windows。

wireshark\_Adapter for loopback traffic captureHGKNW2.pcapng

分组: 21947 · 显示: 37 (0.2%) 配置: Default

### 3. 服务器端向客户端发送响应报文

#### • 状态行

- 协议版本字段: HTTP/1.1
- 状态码: 200
- 状态信息: OK

#### • 相应头

- Content-Type: 文件类型
- Content-Length: 消息长度
- File-Data: 响应报文大小

#### • 相应体

- 我们编写的html文档。

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(T) 帮助(H)

(ip.dst == 127.0.0.1 or ip.src == 127.0.0.1) and (tcp.srcport == 8080 or tcp.dstport == 8080)

No.	Time	Source	Destination	Protocol	Length	Info
2767	44.223776	127.0.0.1	127.0.0.1	TCP	56	50396 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
2768	44.223828	127.0.0.1	127.0.0.1	TCP	56	8080 → 50396 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
2769	44.223878	127.0.0.1	127.0.0.1	TCP	44	50396 → 8080 [ACK] Seq=1 Ack=1 Win=2619648 Len=0

▼ Null/Loopback

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

▼ Transmission Control Protocol, Src Port: 8080, Dst Port: 50396, Seq: 1, Ack: 580, Len: 1277

Source Port: 8080  
Destination Port: 50396  
[Stream Index: 112]  
[Stream Packet Number: 6]  
[Conversation completeness: Complete, WITH\_DATA (31)]  
[TCP Segment Len: 1277]  
Sequence Number: 1 (relative sequence number)  
Sequence Number (raw): 440952522  
[Next Sequence Number: 1278 (relative sequence number)]  
Acknowledgment Number: 580 (relative ack number)  
Acknowledgment number (raw): 351673807  
0101 .... = Header Length: 20 bytes (5)  
▼ Flags: 0x010 (PSH, 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  
....0.... .... = Push: Set  
....0.... .... = Reset: Not set  
....0.... .... = Sync: Not set  
....0.... .... = Fin: Not set  
[TCP Flags: .....AP.....]  
Window: 10233  
[Calculated window size: 2619648]  
[Window size scaling factor: 256]  
Checksum: 0x5d63 [unverified]  
[Checksum Status: Unverified]  
Urgent Pointer: 0  
> [Timestamps]  
> [SEQ/ACK analysis]

激活 Windows  
转到“设置”以激活 Windows。

wireshark\_Adapter for loopback traffic captureHGKNW2.pcapng

分组: 26430 · 显示: 37 (0.1%) 配置: Default

## 4. 客户端回复ACK表示收到请求

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(T) 帮助(H)

(ip.dst == 127.0.0.1 or ip.src == 127.0.0.1) and (tcp.sport == 8080 or tcp.dport == 8080)

No.	Time	Source	Destination	Protocol	Length	Info
2767	44.223776	127.0.0.1	127.0.0.1	TCP	56	50396 → 8880 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
2768	44.223828	127.0.0.1	127.0.0.1	TCP	56	8880 → 50396 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
2769	44.223878	127.0.0.1	127.0.0.1	TCP	44	50396 → 8880 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
2838	44.342684	127.0.0.1	127.0.0.1	HTTP	623	GET / HTTP/1.1
2839	44.342110	127.0.0.1	127.0.0.1	TCP	44	8880 → 50396 [ACK] Seq=1 Ack=580 Win=2619648 Len=0
2844	44.354303	127.0.0.1	127.0.0.1	HTTP	1321	HTTP/1.1 200 OK (text/html)
2845	44.354351	127.0.0.1	127.0.0.1	TCP	44	50396 → 8880 [ACK] Seq=580 Ack=1278 Win=2618368 Len=0

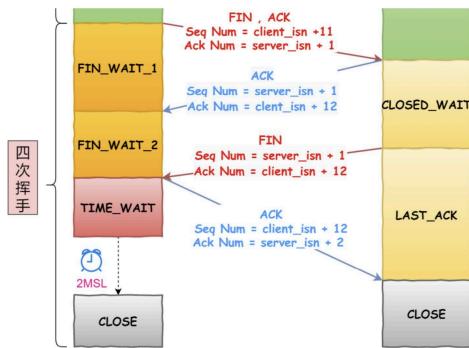
Null/Loopback  
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
Transmission Control Protocol, Src Port: 50396, Dst Port: 8080, Seq: 580, Ack: 1278, Len: 0  
Source Port: 50396  
Destination Port: 8080  
[Stream Index: 112]  
[Stream Packet Number: 7]  
[Conversation completeness: Complete, WITH\_DATA (31)]  
[TCP Segment Len: 0]  
Sequence Number: 580 (relative sequence number)  
Sequence Number (raw): 3351673807  
[Next Sequence Number: 580 (relative sequence number)]  
Acknowledgment Number: 1278 (relative ack number)  
Acknowledgment number (raw): 440553799  
0101 .... = Header Length: 20 bytes (5)  
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  
....0....1.... = Acknowledgment: Set  
....0....0.... = Push: Not set  
....0....0.... = Reset: Not set  
....0....0.... = Syn: Not set  
....0....0....0 = Fin: Not set  
[TCP Flags: .....A....]  
Window: 10228  
[Calculated window size: 2618368]  
[Window size scaling factor: 256]

激活 Windows  
转到“设置”以激活 Windows。

wireshark\_Adapter for loopback traffic captureHGKNW2.pcapng

## 5、TCP四次挥手关闭连接（此处以服务器端主动关闭连接为例）

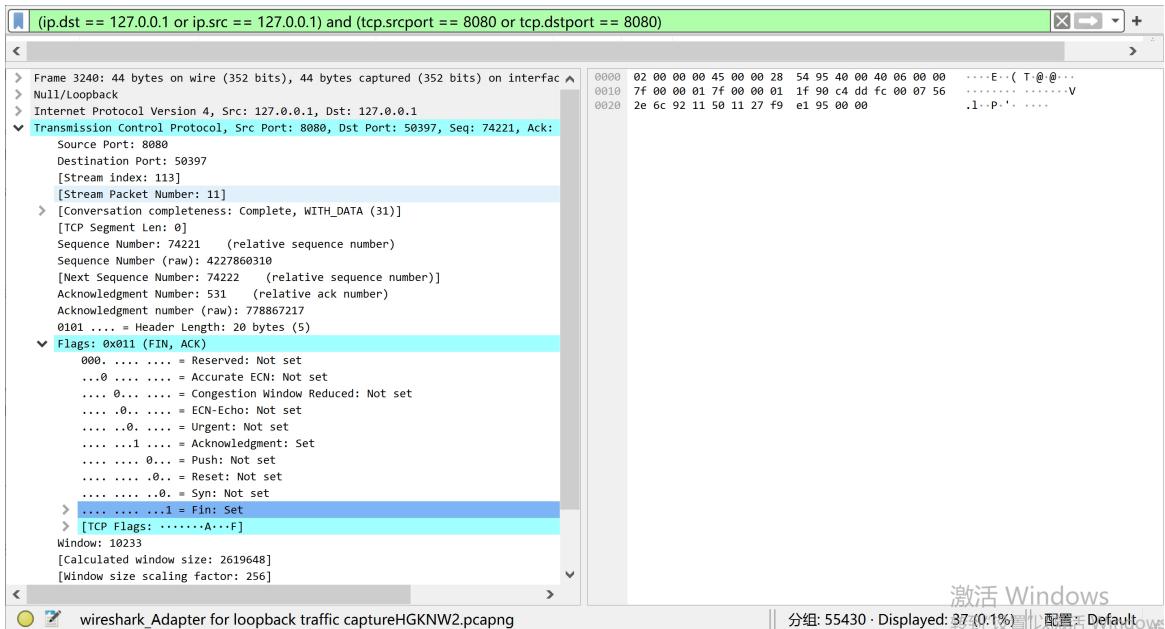
- TCP使用四次挥手关闭连接，客户端和服务端分别释放连接。



- 在wireshark中，我们可以看到四次挥手的过程。

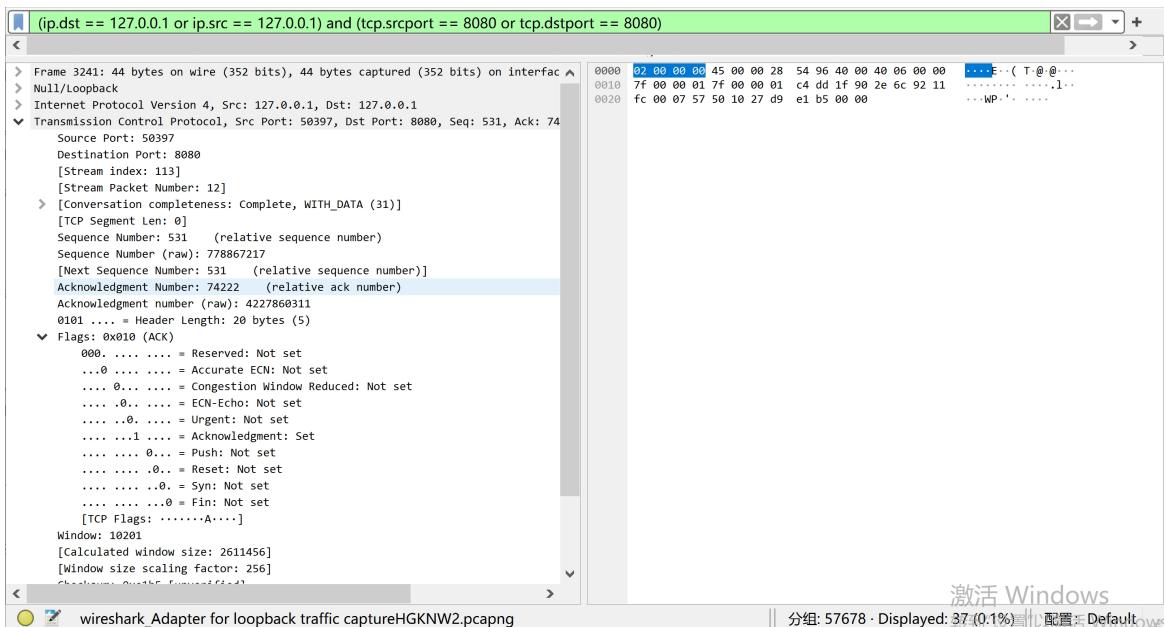
3240	49.432589	127.0.0.1	127.0.0.1	TCP	44	8080 → 50397 [FIN, ACK] Seq=74221 Ack=531 Win=2619648 Len=0
3241	49.432623	127.0.0.1	127.0.0.1	TCP	44	50397 → 8080 [ACK] Seq=531 Ack=74222 Win=2611456 Len=0
3242	49.432735	127.0.0.1	127.0.0.1	TCP	44	50397 → 8080 [FIN, ACK] Seq=531 Ack=74222 Win=2611456 Len=0
3243	49.432758	127.0.0.1	127.0.0.1	TCP	44	8080 → 50397 [ACK] Seq=74222 Ack=532 Win=2619648 Len=0

- 第一次挥手：主动关闭方发送一个FIN并进入FIN\_WAIT1状态



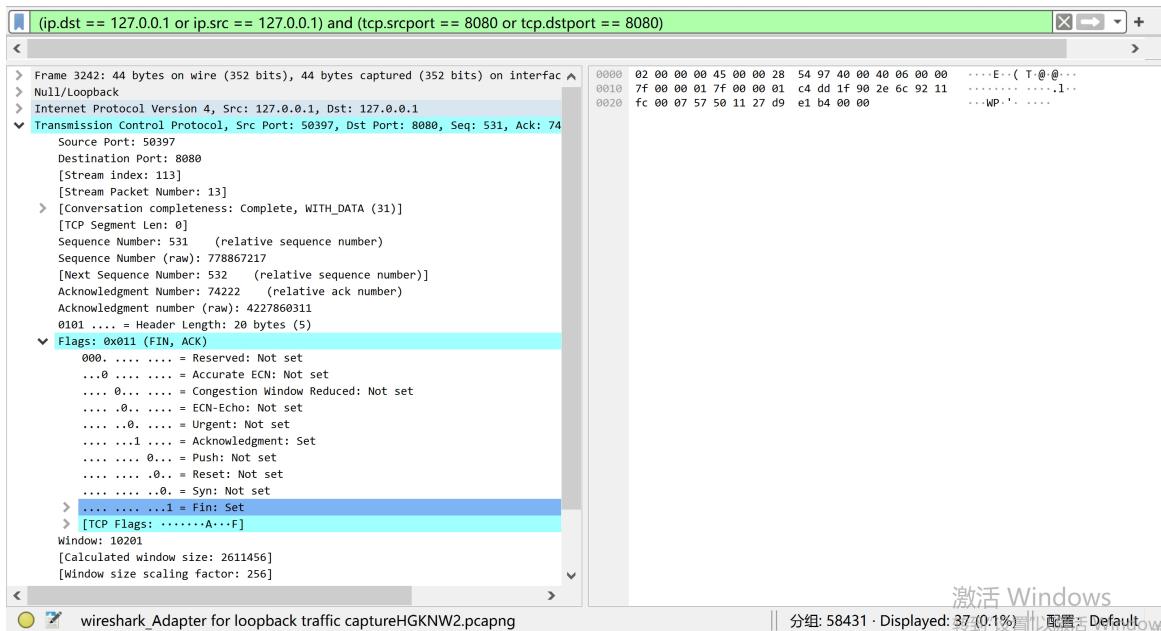
- 服务器端向客户端发送主动关闭连接报文
- 服务器端进入FIN\_WAIT1状态，等待客户端回复

- Source Port: 8080
- Destination Port: 50397
- Seq = 74221
- Ack = 531
- ACK、FIN：标志位
- 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



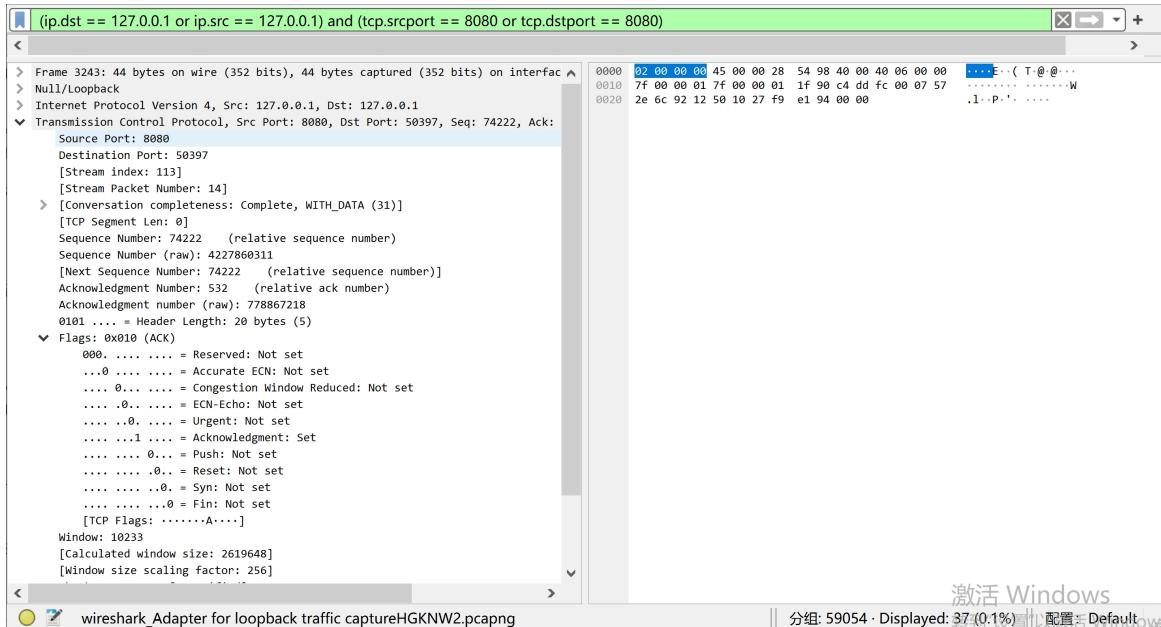
- 客户端接收报文，并回复确认报文
- 客户端进入CLOSE\_WAIT状态；服务器端收到客户端的确认报文后进入FIN\_WAIT2状态
- Source Port: 50397
- Destination Port: 8080
- Seq = 531
- Ack = 74222

- ACK : 标志位
- 第三次挥手：被动关闭方发送一个FIN并进入LAST\_ACK状态



- 客户端在确认数据传输完毕后向服务器端发送结束报文
- 客户端进入LAST\_ACK状态，等待服务器端最后确认

- Source Port: 8080
- Destination Port: 50397
- Seq = 531
- Ack = 74222
- ACK、FIN : 标志位
- 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



- 服务器端回复确认报文，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）
- 客户端接收到确认报文后关闭连接
  - Source Port: 50397

- Destination Port: 8080
- Seq = 74222
- Ack = 532
- ACK : 标志位