



南開大學  
Nankai University

计算机学院  
计算机网络实验报告

web 服务器与浏览器交互过程的抓包分析

姓名：边笛  
学号：2012668  
专业：计算机科学与技术

2022 年 10 月 28 日

目录

1	Web 服务器搭建	2
2	编写 Web 页面	2
3	Wireshark 捕获交互过程	3
4	交互过程分析	4
4.1	TCP 三次握手 . . . . .	4
4.2	HTTP 请求 . . . . .	4
4.3	HTTP 响应 . . . . .	6
4.4	TCP 四次挥手 . . . . .	7
4.5	其他过程分析 . . . . .	8
4.5.1	HTTP 报文 . . . . .	8
4.5.2	TCP 报文 . . . . .	8
5	问题与收获	10
5.1	实验中遇到的问题 . . . . .	10
5.2	收获 . . . . .	10



我自己的姓名全拼的 logo。我对网页背景颜色、字体颜色、图片显示格式等进行了一系列调整，同时还设置了网页的图标，让网页整体看上去更完整了一些，得到了最终的静态网页如下：



该 html 文件以及用到的图片、图标都存放在了建立服务器的路径下，这样才可以在浏览器中访问到我所编写的静态网页。

### 3 Wireshark 捕获交互过程

#### Step 1: 选择捕获选项开始捕获

由于交互过程中的动作都在本机发生，是一系列回环操作，在捕获时选择 *Adapter for loop back traffic capture* 选项开始捕获。

#### Step 2: 产生交互动作

在捕获开始前，我已经在目标路径命令行中输入 `python -m http.server --bind 127.98.9.22 2290` 搭建好了服务器

要想产生交互动作只需在浏览器中输入 `http://127.98.9.22:2290/lab2.html`，就可以加载出网页，实现交互，并由 Wireshark 捕获。

#### Step 2: 产生交互动作

在捕获开始前，我已经在目标路径命令行中输入 `python -m http.server --bind 127.98.9.22 2290` 搭建好了服务器

要想产生交互动作只需在浏览器中输入 `http://127.98.9.22:2290/lab2.html`，就可以加载出网页，实现交互，并由 Wireshark 捕获。

## Step 2: 过滤获取目标信息

停止捕获。由于捕获到的还有很多其他的交互信息，于是在 Wireshark 界面顶端的过滤器命令行输入 `ip.addr == 127.98.9.22` 来过滤得到要分析的目标信息。我是使用 IP 地址来进行过滤的，实际操作时使用端口号也可以。最后选择中所有目标分组，保存即可。

## 4 交互过程分析

### 4.1 TCP 三次握手

首先看前三条分组，是捕获到的 TCP 三次握手信息。

Time	Source	Destination	Protocol	Length	Info
1	2022-10-26 17:29:01.800844	127.0.0.1	127.98.9.22	TCP	56 54281 → 2290 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
2	2022-10-26 17:29:01.800917	127.98.9.22	127.0.0.1	TCP	56 2290 → 54281 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
3	2022-10-26 17:29:01.800928	127.0.0.1	127.98.9.22	TCP	44 54281 → 2290 [ACK] Seq=1 Ack=1 Win=327424 Len=0
4	2022-10-26 17:29:01.804557	127.0.0.1	127.98.9.22	HTTP	546 GET /lab2.html HTTP/1.1
5	2022-10-26 17:29:01.804567	127.98.9.22	127.0.0.1	TCP	44 2290 → 54281 [ACK] Seq=1 Ack=503 Win=2160640 Len=0
6	2022-10-26 17:29:02.155310	127.98.9.22	127.0.0.1	TCP	231 2290 → 54281 [PSH, ACK] Seq=1 Ack=503 Win=2160640 Len=187 [TCP segment of a reassembled PDU]
7	2022-10-26 17:29:02.155343	127.0.0.1	127.98.9.22	TCP	44 54281 → 2290 [ACK] Seq=503 Ack=188 Win=327168 Len=0
8	2022-10-26 17:29:02.155746	127.98.9.22	127.0.0.1	HTTP	1477 HTTP/1.0 200 OK (text/html)
9	2022-10-26 17:29:02.155761	127.0.0.1	127.98.9.22	TCP	44 54281 → 2290 [ACK] Seq=503 Ack=1621 Win=325632 Len=0
10	2022-10-26 17:29:02.155971	127.0.0.1	127.98.9.22	TCP	44 54281 → 2290 [FIN, ACK] Seq=503 Ack=1621 Win=325632 Len=0
11	2022-10-26 17:29:02.155982	127.98.9.22	127.0.0.1	TCP	44 2290 → 54281 [ACK] Seq=1621 Ack=504 Win=2160640 Len=0
12	2022-10-26 17:29:02.156340	127.98.9.22	127.0.0.1	TCP	44 2290 → 54281 [FIN, ACK] Seq=1621 Ack=504 Win=2160640 Len=0
13	2022-10-26 17:29:02.156372	127.0.0.1	127.98.9.22	TCP	44 54281 → 2290 [ACK] Seq=504 Ack=1622 Win=325632 Len=0
14	2022-10-26 17:29:02.202063	127.0.0.1	127.98.9.22	TCP	56 54282 → 2290 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
15	2022-10-26 17:29:02.202096	127.98.9.22	127.0.0.1	TCP	56 2290 → 54282 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
16	2022-10-26 17:29:02.202105	127.0.0.1	127.98.9.22	TCP	44 54282 → 2290 [ACK] Seq=1 Ack=1 Win=2161152 Len=0

→ 三次握手

**第一次握手**是浏览器端向 Web 服务器发送序列号 seq=0 的 SYN 数据包，标志位中 SYN=1，表明请求建立一个连接，浏览器端进入 SYN\_SENT 状态，等待 Web 服务的确认。

**第二次握手**是 Web 服务器向浏览器发送序列号 seq=0 的 SYN+ACK 响应数据包，标志位中 SYN=1，表明 Web 也希望建立 TCP 连接；确认报文段 ACK=1，用来告诉浏览器 Web 收到了来自它的 SYN 包。Web 端进入 SYN\_RECV 状态。

**第三次握手**是浏览器端向 Web 服务器发送序列号 seq=1 的 ACK 响应数据包，标志位中确认报文段 ACK=1，用来告诉 Web 端收到了 SYN 包。在浏览器发出 ACK 报文后，它进入 ESTABLISHED 状态，开始读写数据；而 Web 端收到来自浏览器的确认报文后也进入 ESTABLISHED 状态，可以进行数据的读写。

由此实现了 TCP 的三次握手，之后就可以进行数据的传输了。

### 4.2 HTTP 请求

第四条分组是浏览器端发给 Web 的 HTTP 请求，也就是第一个携带有效数据的包。标志位中 ACK 和 PUSH 两位为 1，ACK=1 是因为 TCP 规定在连接建立后所有传送的报文段都必须把 ACK 设置为 1；PUSH=1 表明该报文段高优先级。

下面以第四条分组为例，具体分析 HTTP 请求的内容。

HTTP 请求分为四个部分：请求行、请求头和请求体：

- 请求行

请求行由请求方法字段 (Request Method)、URL 字段 (Request URI) 和 HTTP 协议版本字段 (Request Version) 3 个字段组成，用空格分隔。

HTTP 协议的请求方法共八种，分别是 GET、POST、HEAD、PUT、DELETE、OPTIONS、TRACE、CONNECT。

- GET: 请求获取 Request-URI 所标识的资源。
- POST: 在 Request-URI 所标识的资源后附加新的数据。
- HEAD: 请求获取由 Request-URI 所标识的资源的响应消息报头。
- PUT: 请求服务器存储一个资源，并用 Request-URI 作为其标识。
- DELETE: 请求服务器删除 Request-URL 所标识的资源。
- OPTIONS: 请求查询服务器的性能，或者查询与资源相关的选项和需求。
- TRACE: 回显服务器收到的请求，主要用于测试或诊断。
- CONNECT: 保留将来使用。

- 请求头

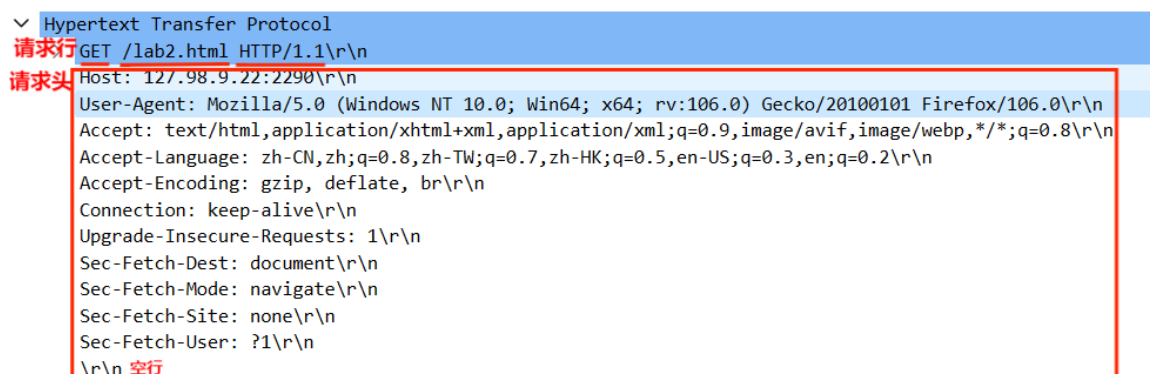
请求头由关键字/值对组成，每行一对，关键字和值用英文冒号“:”分隔。请求头通知服务器有关于客户端请求的信息，包括 Host、Connection 等。

请求头的最后会有一个空行，表示请求头结束。

- 请求体

请求体不在 GET 方法中使用，而在 POST 方法中使用。POST 方法适用于需要客户填写表单的场合，请求体就是用户填写的内容。

在本例只需要向 Web 端请求网页资源，因此是 GET 方式的请求，此处并没有请求体部分。



```
▼ Hypertext Transfer Protocol
请求行 GET /lab2.html HTTP/1.1\r\n
请求头 Host: 127.98.9.22:2290\r\n
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:106.0) Gecko/20100101 Firefox/106.0\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8\r\n
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2\r\n
Accept-Encoding: gzip, deflate, br\r\n
Connection: keep-alive\r\n
Upgrade-Insecure-Requests: 1\r\n
Sec-Fetch-Dest: document\r\n
Sec-Fetch-Mode: navigate\r\n
Sec-Fetch-Site: none\r\n
Sec-Fetch-User: ?1\r\n
\r\n 空行
```

可以在图中看到标注出的请求行、请求头和空行。请求行为 **GET /lab2.html HTTP/1.1**，其三部分用横线画出，表明请求方法是 GET、请求的是 lab2.html、HTTP 协议版本为 1.1。而请求头中包含了诸多信息，在这里做展开分析：

- Host: 主机，这里为 127.98.9.22:2290。
- User-Agent 用户代理，这里的表示使用的用户代理是火狐浏览器。
- Accept: 客户端能够接收的内容类型，这里的信息是浏览器支持的 MIME(多功能 Internet 邮件扩充服务) 类型分别是 HTML 格式,XHTML 格式,XML 数据格式,avif 图片格式,webp 图片格式和任意类型，优先顺序是它们从左到右的排列顺序。

- Accept-Language: 浏览器可接受的语言, 使用语言代码 + 区域代码表示, q 表示请求倾向于获得这种语言的, 这里涉及到的从左到右依次为中文 (中华人民共和国)、中文 (中国台湾)、中文 (中国香港特别行政区)、英语 (美国)、英语 (加勒比)。
- Accept-Encoding: 指定浏览器可以支持的 web 服务器返回内容压缩编码类型
- Connection: 设置当前的事务完成后, 是否会关闭网络连接。(HTTP 1.1 默认进行持久连接), 这里的 keep-alive 表示持久连接。
- Upgrade-Insecure-Requests: 设为 1, 服务器收到请求后会返回 “Content-Security-Policy: upgrade-insecure-requests” 头, 告诉浏览器, 可以把所属本站的所有 http 连接升级为 https 连接。
- Sec-Fetch-Dest: 表示请求的目的地, 即如何使用获取的数据; 这里的 document 表示页面导航算法。
- Sec-Fetch-Mode: 表明请求的模式; 这里的 navigate 表示这是一个浏览器的页面切换请求。navigate 请求仅在浏览器切换页面时创建, 该请求应该返回 HTML;
- Sec-Fetch-Site: 表示请求发起者的来源与目标资源来源之间的关系; 这里的 none 表示是用户直接触发页面导航的, 例如在浏览器地址栏中输入地址, 点击书签跳转等。
- Sec-Fetch-User: 取值是布尔值, true(?1) 表示导航请求由用户激活触发 (鼠标点击/键盘), false(?0) 表示导航请求由用户激活以外的原因触发; 这里的 ? 1 说明是用户激活触发的。

### 4.3 HTTP 响应

第八条分组是由 Web 发给浏览器的 HTTP 响应数据包, 标志位中同样也是 ACK 和 PUSH 两位为 1。接下来以第八条数据包为例分析 HTTP 响应的内容。

HTTP 响应分为响应行, 响应头和响应体三部分:

- 状态行

状态行由 HTTP 协议版本、状态码、状态码描述三部分组成, 用空格分隔。其中 HTTP 协议版本与 HTTP 请求一致。这里重点分析一下状态码。

状态码由三位数字构成的, 用来标识服务端对客户端这次请求的处理结果。状态码与状态码描述是一一对应的, 可以结合在一起来了解。常见的状态码及其状态描述有

- 200 ok, 表示访问成功。
- 404 NOT FOUND, 表示标识请求的 URL 路径没有对应的资源。比如在访问不存在路径时就会出现 404 状态码。
- 403 Forbidden, 表示禁止访问, 有可能时权限不够, 也有可能是没有登陆。
- 405 Method Not Allowed, 方法不支持, 由于前后端约定不同导致。
- 500 Internal Server Error, 表示服务器出现内部错误, 一般是服务器的代码执行过程中遇到了一些特殊情况 (服务器异常崩溃), 比较少见。
- 504 Gateway Timeout, 表示超时, 服务器负载比较大的时候, 就可能会导致出现超时的情况。
- 302 Move temporarily, 临时重定向。
- 301 Moved Permanently, 永久重定向。

- 响应头  
响应头向客户端提供一些额外信息，比如谁在发送响应、响应者的功能，甚至与响应相关的一些特殊指令，格式与请求报头一致。  
响应头的最后会有一个空行，表示响应头结束。
- 响应体  
响应体是服务器返回给客户端的文本信息。

```

Hypertext Transfer Protocol
响应行 HTTP/1.0 200 OK\r\n
响应头 Server: SimpleHTTP/0.6 Python/3.9.13\r\n
      Date: Wed, 26 Oct 2022 09:29:02 GMT\r\n
      Content-type: text/html\r\n
      Content-Length: 1433\r\n
      Last-Modified: Wed, 26 Oct 2022 08:00:19 GMT\r\n
      \r\n 空行
[HTTP response 1/1]
[Time since request: 0.351189000 seconds]
[Request in frame: 4]
[Request URI: http://127.98.9.22:2290/lab2.html]
响应体 File Data: 1433 bytes
> Line-based text data: text/html (31 lines)

```

在此例中，响应行、响应头与响应体都已在图片中标出。响应行为 **HTTP/1.0 200 OK**，表明 Web 的 HTTP 协议版本为 1.0，访问成功。响应头中的信息解析如下：

- Server: 服务器的信息，这里表示是基于 Python3.9.13 的 SimpleHTTP 建立的服务器。
- Date: 响应产生的时间。
- Content-type: 指定返回的数据类型是什么，这里 text/html 代表返回 HTML 文档。
- Content-length: 响应内容的长度 (字节数)。
- Last-Modified: 指定资源的最后修改时间。

而响应体就是我所编写的 html 文档的内容。

4.4 TCP 四次挥手

第 10 至 13 条分组是捕获到的 TCP 打开连接时的四次挥手过程。

Time	Source	Destination	Protocol	Length	Info
1	2022-10-26 17:29:01.800844	127.0.0.1	TCP	56	54281 → 2290 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
2	2022-10-26 17:29:01.800917	127.98.9.22	TCP	56	2290 → 54281 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
3	2022-10-26 17:29:01.800928	127.0.0.1	TCP	44	54281 → 2290 [ACK] Seq=1 Ack=1 Win=32768 Len=0
4	2022-10-26 17:29:01.804557	127.0.0.1	HTTP	546	GET /lab2.html HTTP/1.1 → HTTP请求
5	2022-10-26 17:29:01.804567	127.98.9.22	TCP	44	2290 → 54281 [ACK] Seq=1 Ack=503 Win=2160640 Len=0
6	2022-10-26 17:29:02.155310	127.98.9.22	TCP	231	2290 → 54281 [PSH, ACK] Seq=1 Ack=503 Win=2160640 Len=187 [TCP segment of a reassembled PDU]
7	2022-10-26 17:29:02.155343	127.0.0.1	TCP	44	54281 → 2290 [ACK] Seq=503 Ack=188 Win=327168 Len=0
8	2022-10-26 17:29:02.155746	127.98.9.22	HTTP	1477	HTTP/1.0 200 OK (text/html) → HTTP响应
9	2022-10-26 17:29:02.155761	127.0.0.1	TCP	44	54281 → 2290 [ACK] Seq=503 Ack=1621 Win=325632 Len=0
10	2022-10-26 17:29:02.155971	127.0.0.1	TCP	44	54281 → 2290 [FIN, ACK] Seq=503 Ack=1621 Win=325632 Len=0
11	2022-10-26 17:29:02.155982	127.98.9.22	TCP	44	2290 → 54281 [ACK] Seq=1621 Ack=504 Win=2160640 Len=0
12	2022-10-26 17:29:02.156340	127.0.0.1	TCP	44	2290 → 54281 [FIN, ACK] Seq=1621 Ack=504 Win=2160640 Len=0
13	2022-10-26 17:29:02.156372	127.0.0.1	TCP	44	54281 → 2290 [ACK] Seq=504 Ack=1622 Win=325632 Len=0
14	2022-10-26 17:29:02.202063	127.0.0.1	TCP	56	54282 → 2290 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
15	2022-10-26 17:29:02.202096	127.98.9.22	TCP	56	2290 → 54282 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
16	2022-10-26 17:29:02.202105	127.0.0.1	TCP	44	54282 → 2290 [ACK] Seq=1 Ack=1 Win=2161152 Len=0

第一次挥手是浏览器向 Web 服务器发送序列号 seq=503 的 FIN 数据包，标志位中 FIN 与 ACK 为 1，FIN=1 表示表示关闭连接，该数据包用来请求释放连接，并关闭了浏览器到 Web 服务器的数据传送，浏览器进入 FIN\_WAIT\_1 状态。



**第二次挥手**是 Web 服务器向浏览器发送序列号 seq=503,ACK=504（也就是上一条是浏览器向 Web 服务器的序列号 +1）的 ACK 数据包, 标志位中 ACK 为 1 表示确认，该数据包用来告诉浏览器 Web 收到了释放连接的请求，并对该条请求进行两确认，Web 服务器进入 CLOSE\_WAIT 状态。

**第三次挥手**是 Web 服务器向浏览器发送序列号 seq=1621,ACK=504 的 FIN 数据包, 标志位中 FIN 与 ACK 为 1。Web 服务器向浏览器发送 FIN 报文，表示释放连接，关闭 Web 服务器到浏览器的数据传送，Web 服务器进入 LAST\_ACK 状态

**第四次挥手**是浏览器发送序列号 seq=504,ACK=1622（上一数据包序列号 +1）的 ACK 数据包，标志位中 ACK 为 1 表示确认。该数据包为浏览器收到 Web 端的请求释放连接报文发送的确认收到信息，Web 服务器进入 CLOSED 状态。

四次挥手完成，Web 服务器与浏览器之间的连接断开，若想再进行数据传输，需要再次建立连接。

4.5 其他过程分析

4.5.1 HTTP 报文

在过滤器中输入 http, 筛选出所有的 HTTP 交互数据观察网页相关的数据传输。

	Time	Source	Destination	Protocol	Length	Info
4	2022-10-26 17:29:01.804557	127.0.0.1	127.98.9.22	HTTP	546	GET /lab2.html HTTP/1.1
8	2022-10-26 17:29:02.155746	127.98.9.22	127.0.0.1	HTTP	1477	HTTP/1.0 200 OK (text/html)
17	2022-10-26 17:29:02.202214	127.0.0.1	127.98.9.22	HTTP	481	GET /wcy.jpg HTTP/1.1
22	2022-10-26 17:29:02.202551	127.0.0.1	127.98.9.22	HTTP	483	GET /study.jpg HTTP/1.1
27	2022-10-26 17:29:02.202769	127.0.0.1	127.98.9.22	HTTP	482	GET /logo.png HTTP/1.1
40	2022-10-26 17:29:02.206998	127.98.9.22	127.0.0.1	HTTP	39283	HTTP/1.0 200 OK (JPEG JFIF image)
48	2022-10-26 17:29:02.207882	127.98.9.22	127.0.0.1	HTTP	55130	HTTP/1.0 200 OK (PNG)
121	2022-10-26 17:29:02.216671	127.98.9.22	127.0.0.1	HTTP	47792	HTTP/1.0 200 OK (JPEG JFIF image)
131	2022-10-26 17:29:02.232188	127.0.0.1	127.98.9.22	HTTP	485	GET /favicon.ico HTTP/1.1
135	2022-10-26 17:29:02.234459	127.98.9.22	127.0.0.1	HTTP	17002	HTTP/1.0 200 OK (image/x-icon)

可以看到在交互完 html 文档后，浏览器应该是对 html 文件进行了解析。由于文档中涉及到的还有一些图片和图标，浏览器也需要获取这些资源，于是向 Web 发送了获取图片的 GET 请求和获取图标的 GET 请求，Web 也做出了响应。

比较有趣的一点是这个请求与响应并不是一请求一响应的，而是浏览器依次发送完所有图片的 GET 请求，然后 Web 依次对图片请求进行响应；在所有完成所有对于 GET 图片的响应后，浏览器才会发出对于图标的 GET 请求，Web 再进行响应。

经查阅资料了解到，浏览器在解析代码时发现一个标签引用了一张图片，会向服务器发出请求，但并不会等到图片下载完，而是继续渲染后面的代码，待服务器返回图片文件后，浏览器会回过头来重新渲染这部分代码。所以请求与响应的顺序并不是说请求完才返回，只是恰巧第一个响应是在最后一张图片请求完之后才发出。

4.5.2 TCP 报文

可以看到在 HTTP 交互之外的 TCP 交互信息还是很多的。

1	127.0.0.1	127.98.9.22	TCP	56	54281 → 2290	[SYN]	Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
2	127.98.9.22	127.0.0.1	TCP	56	2290 → 54281	[SYN, ACK]	Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
3	127.0.0.1	127.98.9.22	TCP	44	54281 → 2290	[ACK]	Seq=1 Ack=1 Win=327424 Len=0
4	127.0.0.1	127.98.9.22	HTTP	546	GET /lab2.html HTTP/1.1		
5	127.98.9.22	127.0.0.1	TCP	44	2290 → 54281	[ACK]	Seq=1 Ack=503 Win=2160640 Len=0
6	127.98.9.22	127.0.0.1	TCP	231	2290 → 54281	[PSH, ACK]	Seq=1 Ack=503 Win=2160640 Len=187 [TCP segment of a reassembled PDU]
7	127.0.0.1	127.98.9.22	TCP	44	54281 → 2290	[ACK]	Seq=503 Ack=188 Win=327168 Len=0
8	127.98.9.22	127.0.0.1	HTTP	1477	HTTP/1.0 200 OK (text/html)		
9	127.0.0.1	127.98.9.22	TCP	44	54281 → 2290	[ACK]	Seq=503 Ack=1621 Win=325632 Len=0
10	127.0.0.1	127.98.9.22	TCP	44	54281 → 2290	[FIN, ACK]	Seq=503 Ack=1621 Win=325632 Len=0
11	127.98.9.22	127.0.0.1	TCP	44	2290 → 54281	[ACK]	Seq=1621 Ack=504 Win=2160640 Len=0
12	127.98.9.22	127.0.0.1	TCP	44	2290 → 54281	[FIN, ACK]	Seq=1621 Ack=504 Win=2160640 Len=0
13	127.0.0.1	127.98.9.22	TCP	44	54281 → 2290	[ACK]	Seq=504 Ack=1622 Win=325632 Len=0
14	127.0.0.1	127.98.9.22	TCP	56	54282 → 2290	[SYN]	Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
15	127.98.9.22	127.0.0.1	TCP	56	2290 → 54282	[SYN, ACK]	Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
16	127.0.0.1	127.98.9.22	TCP	44	54282 → 2290	[ACK]	Seq=1 Ack=1 Win=2161152 Len=0
17	127.0.0.1	127.98.9.22	HTTP	481	GET /wcy.jpg HTTP/1.1		
18	127.98.9.22	127.0.0.1	TCP	44	2290 → 54282	[ACK]	Seq=1 Ack=438 Win=2160896 Len=0
19	127.0.0.1	127.98.9.22	TCP	56	54283 → 2290	[SYN]	Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
20	127.98.9.22	127.0.0.1	TCP	56	2290 → 54283	[SYN, ACK]	Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
21	127.0.0.1	127.98.9.22	TCP	44	54283 → 2290	[ACK]	Seq=1 Ack=1 Win=2161152 Len=0
22	127.0.0.1	127.98.9.22	HTTP	483	GET /study.jpg HTTP/1.1		
23	127.98.9.22	127.0.0.1	TCP	44	2290 → 54283	[ACK]	Seq=1 Ack=440 Win=2160896 Len=0
24	127.0.0.1	127.98.9.22	TCP	56	54284 → 2290	[SYN]	Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
25	127.98.9.22	127.0.0.1	TCP	56	2290 → 54284	[SYN, ACK]	Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
26	127.0.0.1	127.98.9.22	TCP	44	54284 → 2290	[ACK]	Seq=1 Ack=1 Win=2161152 Len=0
27	127.0.0.1	127.98.9.22	HTTP	482	GET /Logo.png HTTP/1.1		
28	127.98.9.22	127.0.0.1	TCP	44	2290 → 54284	[ACK]	Seq=1 Ack=439 Win=2160896 Len=0
29	127.98.9.22	127.0.0.1	TCP	234	2290 → 54282	[PSH, ACK]	Seq=1 Ack=438 Win=2160896 Len=190 [TCP segment of a reassembled PDU]
30	127.0.0.1	127.98.9.22	TCP	44	54282 → 2290	[ACK]	Seq=438 Ack=191 Win=2160896 Len=0
31	127.98.9.22	127.0.0.1	TCP	235	2290 → 54283	[PSH, ACK]	Seq=1 Ack=440 Win=2160896 Len=191 [TCP segment of a reassembled PDU]
32	127.0.0.1	127.98.9.22	TCP	44	54283 → 2290	[ACK]	Seq=440 Ack=192 Win=2160896 Len=0
33	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54282	[ACK]	Seq=191 Ack=438 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
34	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54282	[ACK]	Seq=65686 Ack=438 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
35	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54282	[ACK]	Seq=131181 Ack=438 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
36	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54282	[ACK]	Seq=196676 Ack=438 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
37	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54282	[ACK]	Seq=262171 Ack=438 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
38	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54282	[ACK]	Seq=327666 Ack=438 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
39	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54282	[ACK]	Seq=393161 Ack=438 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
40	127.98.9.22	127.0.0.1	HTTP	39283	HTTP/1.0 200 OK (JPEG JFIF image)		
41	127.0.0.1	127.98.9.22	TCP	44	54282 → 2290	[ACK]	Seq=438 Ack=497895 Win=1860096 Len=0
42	127.0.0.1	127.98.9.22	TCP	44	[TCP Window Update] 54282 → 2290	[ACK]	Seq=438 Ack=497895 Win=1794560 Len=0
43	127.98.9.22	127.0.0.1	TCP	232	2290 → 54284	[PSH, ACK]	Seq=1 Ack=439 Win=2160896 Len=188 [TCP segment of a reassembled PDU]
44	127.0.0.1	127.98.9.22	TCP	44	54284 → 2290	[ACK]	Seq=439 Ack=189 Win=2160896 Len=0
45	127.0.0.1	127.98.9.22	TCP	44	[TCP Window Update] 54282 → 2290	[ACK]	Seq=438 Ack=497895 Win=2161152 Len=0
46	127.0.0.1	127.98.9.22	TCP	44	54282 → 2290	[FIN, ACK]	Seq=438 Ack=497895 Win=2161152 Len=0
47	127.98.9.22	127.0.0.1	TCP	44	2290 → 54282	[ACK]	Seq=497895 Ack=439 Win=2160896 Len=0
48	127.98.9.22	127.0.0.1	HTTP	55130	HTTP/1.0 200 OK (PNG)		
49	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=192 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
50	127.0.0.1	127.98.9.22	TCP	44	54284 → 2290	[ACK]	Seq=439 Ack=55275 Win=2105856 Len=0

首先发现在浏览器发出 HTTP 请求后，Web 服务器会发回一个 ACK 报文表示确认。

在浏览器发出请求到 Web 服务器做出响应的过程中，Web 服务器会向浏览器发送 PSH 报文传输数据。数据传输过程是发送方每一次 write，都会将这一次的数据打包成一个或多个 TCP 报文段)，并将最后一个 TCP 报文段标记为 PSH；只有当接收方收到包含了 PSH 标志的报文才会将接收缓冲区中的所有数据推送给应用进程，保证了数据的完整性。（如果发送缓冲区满了也会把缓冲区数据打包发送，同理接收缓冲区满了也会推送给应用进程）

79	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=1048768 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
80	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=1114263 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
81	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=1179758 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
82	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=1245253 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
83	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=1310748 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
84	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=1376243 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
85	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=1441738 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
86	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=1507233 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
87	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=1572728 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
88	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=1638223 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
89	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=1703718 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
90	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=1769213 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
91	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=1834708 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
92	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=1900203 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
93	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=1965698 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
94	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=2031193 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
95	127.98.9.22	127.0.0.1	TCP	700	2290 → 54283	[PSH, ACK]	Seq=2096688 Ack=440 Win=2160896 Len=656 [TCP segment of a reassembled PDU]
96	127.0.0.1	127.98.9.22	TCP	44	54283 → 2290	[ACK]	Seq=440 Ack=2097344 Win=1309184 Len=0
97	127.0.0.1	127.98.9.22	TCP	44	[TCP Window Update] 54283 → 2290	[ACK]	Seq=440 Ack=2097344 Win=1243648 Len=0
98	127.0.0.1	127.98.9.22	TCP	44	[TCP Window Update] 54283 → 2290	[ACK]	Seq=440 Ack=2097344 Win=2161152 Len=0
99	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=2097344 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
100	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=2162839 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
101	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=2228334 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
102	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=2293829 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
103	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=2359324 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
104	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=2424819 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
105	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=2490314 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
106	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=2555809 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
107	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=2621304 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
108	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=2686799 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
109	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=2752294 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
110	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=2817789 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
111	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=2883284 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
112	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=2948779 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
113	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=3014274 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
114	127.98.9.22	127.0.0.1	TCP	65539	2290 → 54283	[ACK]	Seq=3079769 Ack=440 Win=2160896 Len=65495 [TCP segment of a reassembled PDU]
115	127.98.9.22	127.0.0.1	TCP	700	2290 → 54283	[PSH, ACK]	Seq=3145264 Ack=440 Win=2160896 Len=656 [TCP segment of a reassembled PDU]
116	127.0.0.1	127.98.9.22	TCP	44	54283 → 2290	[ACK]	Seq=440 Ack=3145920 Win=1112576 Len=0

此外还会看到一些标有 [TCP window update] 的数据，这是当接收端接收窗口大小发生变化可以接收数据了，就会有该标志。例如，接收方消耗缓冲数据后，更新 TCP 窗口，可以看到从 win 逐渐变大，这时 wireshark 会打

上 [TCP window update] 标签。

5 问题与收获

5.1 实验中遇到的问题

1. 再进行抓包前如果已经通过 Web 服务器访问过了网页，在加载时会出现 GET 图片状态码为 304 的问题，查阅资料发现是浏览器缓存导致的，由于先前访问过该链接，浏览器将同一个链接认为是相同的请求，所以便没有往 Web 端发送该请求以致于设置失败。

我的解决方法是按 Ctrl+F5 强制刷新，这样就会出现恢复正常的结果了。

	Source	Destination	Protocol	Length	Info
1864	127.0.0.1	127.98.9.22	HTTP	596	GET /lab2.html HTTP/1.1
1874	127.98.9.22	127.0.0.1	HTTP	1477	HTTP/1.0 200 OK (text/html)
1970	127.0.0.1	127.98.9.22	HTTP	531	GET /wcy.jpg HTTP/1.1
1976	127.0.0.1	127.98.9.22	HTTP	533	GET /study.jpg HTTP/1.1
1997	127.98.9.22	127.0.0.1	HTTP	148	HTTP/1.0 304 Not Modified
2009	127.0.0.1	127.98.9.22	HTTP	532	GET /logo.png HTTP/1.1
2015	127.98.9.22	127.0.0.1	HTTP	148	HTTP/1.0 304 Not Modified
2025	127.98.9.22	127.0.0.1	HTTP	148	HTTP/1.0 304 Not Modified
2717	127.0.0.1	127.98.9.22	HTTP	589	GET /lab2.html HTTP/1.1
2725	127.98.9.22	127.0.0.1	HTTP	1477	HTTP/1.0 200 OK (text/html)
2766	127.0.0.1	127.98.9.22	HTTP	524	GET /wcy.jpg HTTP/1.1
2791	127.0.0.1	127.98.9.22	HTTP	526	GET /study.jpg HTTP/1.1
2808	127.0.0.1	127.98.9.22	HTTP	525	GET /logo.png HTTP/1.1
2823	127.98.9.22	127.0.0.1	HTTP	39283	HTTP/1.0 200 OK (JPEG JFIF image)
2851	127.98.9.22	127.0.0.1	HTTP	55130	HTTP/1.0 200 OK (PNG)
2926	127.98.9.22	127.0.0.1	HTTP	47792	HTTP/1.0 200 OK (JPEG JFIF image)
2999	127.0.0.1	127.98.9.22	HTTP	528	GET /favicon.ico HTTP/1.1
3007	127.98.9.22	127.0.0.1	HTTP	17002	HTTP/1.0 200 OK (image/x-icon)

刷新

2. html 的解析是从上到下进行的，但图标链接是在 head 里，在我的理解里应该先于图片被加载，但事实上它的请求最晚。查了一些资料但并没有找到答案，我猜测可能是因为图标在 head 里的链接调用和链接调用 CSS、JS 这种不一样，在最后渲染的时候才会起到作用。后面还会继续查一查。

5.2 收获

在本次实验中，首先了解到了如何使用 python 搭建一个简单的服务器，其次学习了 html 文件的简单编写，html 的编写过程还是很有趣的，在图书馆找到了几本通俗易懂的书，作业完成后还可以再继续读一读，深入了解一些。在 Wireshark 抓包过程中分析过程中，详细了解了 TCP 三次握手四次挥手已经 HTTP 请求与响应，简略了解了网页解析渲染的过程，但是这其中还有很多东西需要去学习。