



南开大学
Nankai University

南 开 大 学

网 络 空 间 安 全 学 院

计 算 机 网 络 实 验 报 告

建立简单的 Web 服务器与 Wireshark 分析

2014074 费泽锟

年级：2020 级

专业：信息安全

指导教师：徐敬东，张建忠

2022 年 10 月 26 日

摘要

在本次实验中，尝试编写了简单的 HTML 静态网页页面文档，初步了解认识了超文本标记语言的语法与语义。使用了 Python 中的 Flask 在本机上搭建了简单的 Web 服务器。（IP 地址为 127.0.0.1）（包含一个 HTML 文档，一个 CSS 格式文件以及 Logo 图片）

在搭建好的 Web 服务器的基础上，通过 Wireshark 对本机进行抓包，设置合适的过滤器可以捕获完整的 Web 服务器与浏览器的 Http 协议交互。

关键字：HTML, Flask, Wireshark, http/1.1

目录

一、 实验内容	1
(一) 实验要求	1
(二) Web 页面设计	1
(三) Web 服务器搭建	3
二、 Wireshark 捕获分析	4
(一) TCP 三次握手建立连接	5
(二) HTTP 请求交互	8
(三) TCP 四次挥手断开连接	9
三、 总结	11

一、 实验内容

(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）和自己的 Logo。
2. 通过浏览器获取自己编写的 Web 页面，使用 Wireshark 捕获浏览器与 Web 服务器的交互过程，并进行简单的分析说明。

(二) Web 页面设计

在本次实验中搭建简单的 Web 服务器，Web 网页不需要十分复杂。在本次设计中，主要设计了一个静态的 HTML 网页来展示个人信息，并没有添加网页的动态交互部分。为了一定的网页美观的效果，并没有将 HTML 中的 `img`、`div` 等部分的 `style` 格式保留在 HTML 文件中，而是单独设计了一个 CSS 格式文件对页面进行优化。通过 `link` 使得 HTML 文档应用该 CSS 文件，最终的网页展示效果包括动态效果和静态信息、Logo 展示。

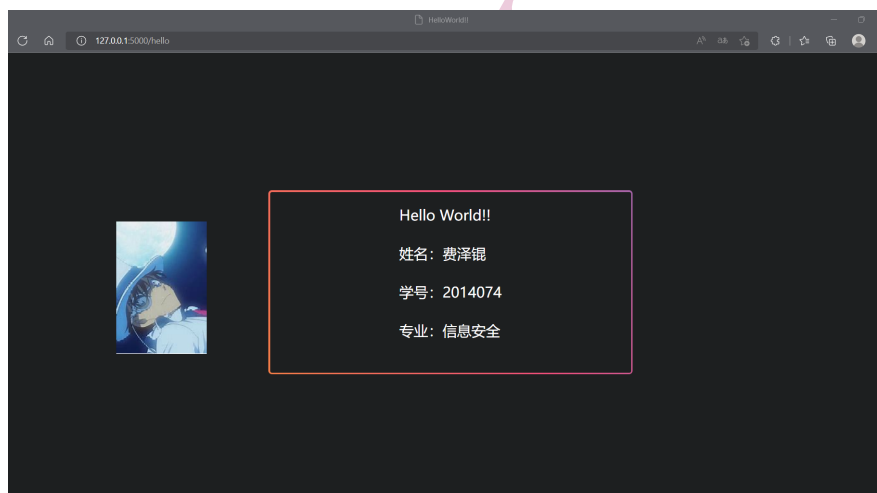


图 1: 个人 Web 网页展示

具体的 CSS 设计会将背景的底色设置为颜色可以变换的背景，再通过全为黑色的幕布进行遮盖，通过 `div` 窗口的边框设计实现动态的边框效果，具体的 HTML 文件以及 CSS 文件中的内容如下：

HTML 文件内容：

HTML 文件编写

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>HelloWorld!!</title>
6 </head>
7 <body>
8 
9 <style>
```

```
10     img{
11         position: absolute;
12         left: 180px;
13         top: 280px;
14     }
15 </style>
16 <div class="gradient-border" id="box">
17     Hello World!!<br/> <br/>
18     姓名: 费泽锟 <br/> <br/>
19     学号: 2014074 <br/> <br/>
20     专业: 信息安全 <br/> <br/>
21 </div>
22 <link rel="stylesheet" href="../../static/css/mystyle.css">
23 </body>
24 </html>
```

CSS 文件:

CSS 文件编写

```
1  html, body {
2      height: 100%;
3  }
4  body {
5      display: flex;
6      justify-content: center;
7      align-items: center;
8      height: 100%;
9      background: #1D1F20;
10 }
11 #box {
12     display: flex;
13     align-items: center;
14     justify-content: center;
15     width: 600px;
16     height: 300px;
17     color: white;
18     font-family: 'Raleway';
19     font-size: 1.5rem;
20 }
21 .gradient-border {
22     --borderWidth: 3px;
23     background: #1D1F20;
24     position: relative;
25     border-radius: var(--borderWidth);
26 }
27 .gradient-border:after {
28     content: '';
29     position: absolute;
30     top: calc(-1 * var(--borderWidth));
```

```

31 left: calc(-1 * var(--borderWidth));
32 height: calc(100% + var(--borderWidth) * 2);
33 width: calc(100% + var(--borderWidth) * 2);
34 background: linear-gradient(60deg, #f79533, #f37055, #ef4e7b, #a166ab,
    #5073b8, #1098ad, #07b39b, #6fba82);
35 border-radius: calc(2 * var(--borderWidth));
36 z-index: -1;
37 animation: animatedgradient 3s ease alternate infinite;
38 background-size: 300% 300%;
39 }
40
41
42 @keyframes animatedgradient {
43     0% {
44         background-position: 0% 50%;
45     }
46     50% {
47         background-position: 100% 50%;
48     }
49     100% {
50         background-position: 0% 50%;
51     }
52 }

```

(三) Web 服务器搭建

在本次设计之中并没有通过 Internet Information Services 互联网信息服务 (IIS 服务) 在本地搭建服务器, 而是通过 Python 中的 Flask 框架进行 Web 服务器的搭建。

Flask 是一个轻量级的可定制框架, 使用 Python 语言编写, 较其他同类型框架更为灵活、轻便、安全且容易上手。它可以很好地结合 MVC 模式进行开发, 开发人员分工合作, 小型团队在短时间内就可以完成功能丰富的中小型网站或 Web 服务的实现。

具体的 Python 代码如下:

HTML 文件编写

```

1 from flask import Flask, render_template
2 from gevent import pywsgi
3
4 app = Flask(__name__)
5 app.debug = True
6
7
8 @app.route('/hello', methods=['GET'])
9 def hello_world():
10     return render_template('hello.html')
11
12
13 if __name__ == '__main__':

```

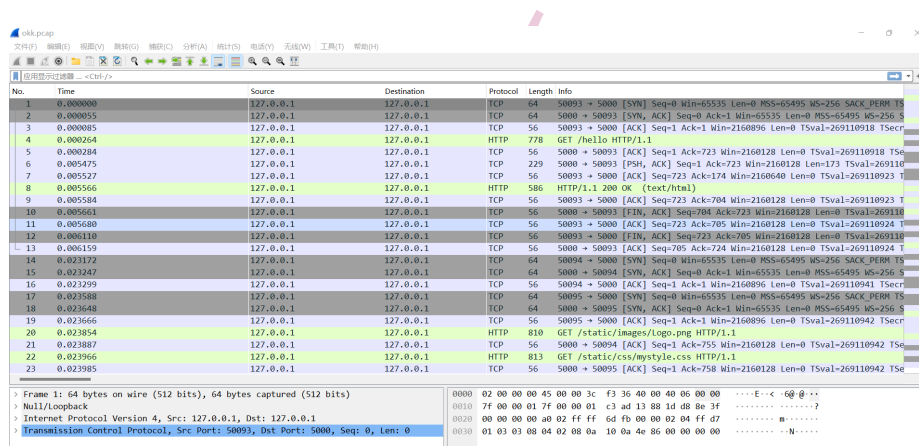
```
14 app.run()
15 # server = pywsgi.WSGIServer(('0.0.0.0', 5000), app)
16 # server.serve_forever()
```

编写完毕后，通过运行 main 函数，app.run() 即可成功开启 Web 服务器了，这时我们就可以打开浏览器输入网址：http://127.0.0.1:5000/Hello，即可成功访问到 HTML 资源了。

二、Wireshark 捕获分析

为实现本地回环地址抓包，需使用 Wireshark 的 Adapter for loopback traffic capture 功能，因此先安装 npcap。之后再打开 wireshark 开始捕获。

因为本地还有其余的服务，所以会有许多我们不希望看到的数据包，这时我们就需要通过设置合适的过滤器来捕获我们希望分析的数据包，因为在本次设计之中，设计的 TCP 端口为 5000，所以过滤器设置为 tcp.port == 5000 就可以成功只捕获自己想要的数据包了，用浏览器访问 http://127.0.0.1:5000/Hello 进行捕获，关闭网页之后得到的捕获结果如下：



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	64	50093 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
2	0.000055	127.0.0.1	127.0.0.1	TCP	64	5000 → 50093 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
3	0.000085	127.0.0.1	127.0.0.1	TCP	56	50093 → 5000 [ACK] Seq=1 Ack=1 Win=2160896 Len=0 TSval=269110918 TSecr=
4	0.000264	127.0.0.1	127.0.0.1	HTTP	778	GET /hello HTTP/1.1
5	0.000284	127.0.0.1	127.0.0.1	TCP	56	5000 → 50093 [ACK] Seq=1 Ack=723 Win=2160128 Len=0 TSval=269110918 TSecr=
6	0.005475	127.0.0.1	127.0.0.1	TCP	229	5000 → 50093 [PSH, ACK] Seq=1 Ack=723 Win=2160128 Len=173 TSval=269110918 TSecr=
7	0.005527	127.0.0.1	127.0.0.1	TCP	56	50093 → 5000 [ACK] Seq=723 Ack=174 Win=2160840 Len=0 TSval=269110923 TSecr=
8	0.005566	127.0.0.1	127.0.0.1	HTTP	586	HTTP/1.1 200 OK (text/html)
9	0.005584	127.0.0.1	127.0.0.1	TCP	56	50093 → 5000 [ACK] Seq=723 Ack=704 Win=2160128 Len=0 TSval=269110923 TSecr=
10	0.005661	127.0.0.1	127.0.0.1	TCP	56	5000 → 50093 [FIN, ACK] Seq=704 Ack=723 Win=2160128 Len=0 TSval=269110923 TSecr=
11	0.005680	127.0.0.1	127.0.0.1	TCP	56	50093 → 5000 [ACK] Seq=723 Ack=705 Win=2160128 Len=0 TSval=269110924 TSecr=
12	0.006110	127.0.0.1	127.0.0.1	TCP	56	50093 → 5000 [FIN, ACK] Seq=723 Ack=705 Win=2160128 Len=0 TSval=269110924 TSecr=
13	0.006159	127.0.0.1	127.0.0.1	TCP	56	5000 → 50093 [ACK] Seq=705 Ack=724 Win=2160128 Len=0 TSval=269110924 TSecr=
14	0.023172	127.0.0.1	127.0.0.1	TCP	64	50094 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
15	0.023247	127.0.0.1	127.0.0.1	TCP	64	5000 → 50094 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
16	0.023299	127.0.0.1	127.0.0.1	TCP	56	50094 → 5000 [ACK] Seq=1 Ack=1 Win=2160896 Len=0 TSval=269110941 TSecr=
17	0.023580	127.0.0.1	127.0.0.1	TCP	64	50095 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
18	0.023648	127.0.0.1	127.0.0.1	TCP	64	5000 → 50095 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
19	0.023666	127.0.0.1	127.0.0.1	TCP	56	50095 → 5000 [ACK] Seq=1 Ack=1 Win=2160896 Len=0 TSval=269110942 TSecr=
20	0.023854	127.0.0.1	127.0.0.1	HTTP	810	GET /static/images/logo.png HTTP/1.1
21	0.023887	127.0.0.1	127.0.0.1	TCP	56	5000 → 50094 [ACK] Seq=1 Ack=755 Win=2160128 Len=0 TSval=269110942 TSecr=
22	0.023966	127.0.0.1	127.0.0.1	HTTP	813	GET /static/css/mystyle.css HTTP/1.1
23	0.023985	127.0.0.1	127.0.0.1	TCP	56	5000 → 50095 [ACK] Seq=1 Ack=758 Win=2160128 Len=0 TSval=269110942 TSecr=

图 2: Wireshark 结果

HTTP 交互整体流程

1. 地址解析

从中分解出协议名、主机名、端口、对象路径等部分。同时需要域名系统 DNS 解析域名，得主机的 IP 地址。

2. 封装 HTTP 请求数据包

把主机名、端口号等信息结合本机自己的信息，封装成一个 HTTP 请求数据包。

3. 封装成 TCP 包，TCP 三次握手建立连接

在 HTTP 工作开始之前，客户机（Web 浏览器）首先要通过网络与服务器建立连接，该连接是通过 TCP 来完成的，该协议与 IP 协议共同构建 Internet，即著名的 TCP/IP 协议族。

4. 客户端向服务器发送请求命令

建立 TCP 连接后，客户机发送一个请求给服务器，请求方式的格式为：统一资源标识符（URL）、协议版本号，后边是 MIME 信息包括请求修饰符、客户机信息和可内容。

5. 服务器响应

服务器接到请求后，给予相应的响应信息，其格式为一个状态行，包括信息的协议版本号、一个成功或错误的代码，后边是 MIME 信息包括服务器信息、实体信息和可能的内容。

6. TCP 四次挥手断开连接

在断开连接之前客户端和服务端都处于 ESTABLISHED 状态，双方都可以主动断开连接，以客户端主动断开连接为优。

(一) TCP 三次握手建立连接

TCP 三次握手建立连接：

使用 TCP 协议进行通信的双方必须先建立连接，然后才能开始传输数据。为了确保连接双方可靠性，在双方建立连接时，TCP 协议采用了三次握手策略。

客户端和通讯端要进行连接，要确认双方的收发能力都是正常的

1. 第一次握手

客户端发送给服务器，服务器能收到，那么这个时候服务端能确定客户端的发送能力正常，服务端的接受能力正常。

2. 第二次握手

客户端确认服务端的接受能力正常，发送能力也是正常的，也知道了自己的发送能力是正常的。

3. 第三次握手

因为服务端只知道客户端的发送能力和自己的接受能力正常，通过第三次握手，服务端知道客户端的接收能力正常（客户端对服务端有了第二次的回应），服务端的发送能力正常。

下图引用自博客，展示的是具体的 TCP 三次握手建立连接的过程：

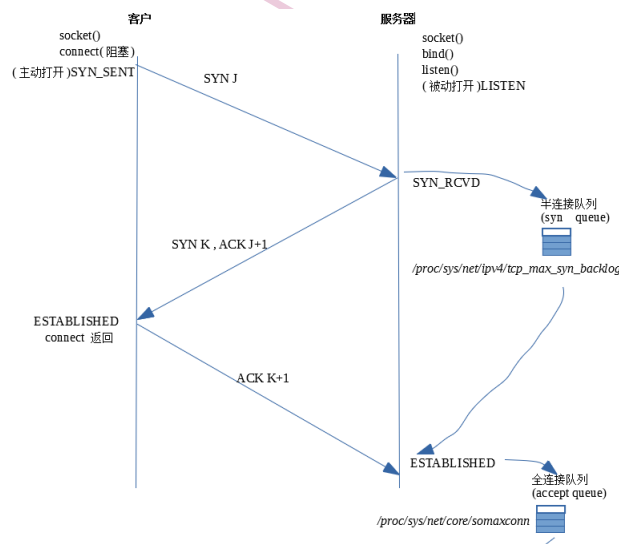


图 3: TCP 三次握手

在 Wireshark 捕获的过程之中，我们可以轻松地发现通过 5000 端口的前三个 TCP 数据包，正是 Web 服务器与浏览器建立连接的 TCP 三次握手数据包。

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	64	50093 → 50000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
2	0.000055	127.0.0.1	127.0.0.1	TCP	64	50000 → 50093 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
3	0.000085	127.0.0.1	127.0.0.1	TCP	56	50093 → 50000 [ACK] Seq=1 Ack=1 Win=2160896 Len=0 TSval=269110918 TSecr=0
4	0.000264	127.0.0.1	127.0.0.1	HTTP	778	GET /hello HTTP/1.1

图 4: Wireshark 三次握手

接下来讲点关于 TCP 三次握手的细节：

1. 客户端主动向服务器端发送 TCP 报文标记位为 SYN=1，表示请求建立新连接，发送初始序列号，随后客户端进入 SYN-SENT 阶段。
2. 服务器端接收到来自客户端的 TCP 报文之后，结束 LISTEN 阶段回复一段 TCP 报文
3. 标志位为 SYN=1，ACK=1，表示确认客户端的报文 Seq 序号有效，服务器能正常接收客户端发送的数据，并同意创建新连接，序号为 Seq=0，确认号为 Ack=0+1=1，即收到客户端的序号 Seq 并将其值加 1，随后服务器端进入 SYN-RCVD 阶段。
4. 客户端接收到来自服务器端的确认收到数据的 TCP 报文之后，明确了从客户端到服务器的数据传输是正常的，结束 SYN-SENT 阶段。

在 TCP 数据包之中我们能够轻松地看到相应的 Seq 序号的变化，但是如果想要找到具体的 SYN 以及 ACK 响应信息，我们还需要知道一点细节，那就是 TCP 数据包的字段布局分布。



图 5: TCP 头部结构

这时预备知识已经充分了，我们就可以仔细分析分析数据包内部的内容了，先看 Seq 序号的变化，首先就是三次握手之中的第一个数据包中的初始 Seq 序列号：

```

[Stream index: 0]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 0]
Sequence Number: 0 (relative sequence number)
Sequence Number (raw): 1915658151
[Next Sequence Number: 1 (relative sequence number)]
Acknowledgment Number: 0
Acknowledgment number (raw): 0
1010 .... = Header Length: 40 bytes (10)
> Flags: 0x002 (SYN)
Window: 65535
[Calculated window size: 65535]
0000 02 00 00 00 45 00 00 3c 95 da 40 00 40 06 00 00 .....E...<..@..
0010 7f 00 00 01 7f 00 00 01 fa 03 13 88 72 2e 9f a7 .....r...
0020 00 00 00 00 a0 02 ff ff bf a6 00 00 02 04 ff d7 .....
0030 01 03 03 08 04 02 08 0a 0f b3 61 1d 00 00 00 00 .....a.....

```

图 6: Seq 初始序列号

而在服务器收到信号，进行回复的数据报文之中（也就是三次握手之中的第二次），我们可以看到对应的 Acknowledgment number 确实为初始 Seq 序列号加一的结果：


```
Sequence Number: 0 (relative sequence number)
Sequence Number (raw): 1040032857
[Next Sequence Number: 1 (relative sequence number)]
Acknowledgment Number: 1 (relative ack number)
Acknowledgment number (raw): 1915658152
1010 .... = Header Length: 40 bytes (10)
> Flags: 0x012 (SYN, ACK)
Window: 65535
[Calculated window size: 65535]
Checksum: 0x6c6e [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0

0000 02 00 00 00 45 00 00 3c 95 db 40 00 40 06 00 00 ....E...<..@..@...
0010 7f 00 00 01 7f 00 00 01 13 88 fa 03 3d fd a4 59 .....=...Y
0020 72 2e 9f a8 a0 12 ff ff 6c 6e 00 00 02 04 ff d7 r.....ln.....
0030 01 03 03 08 04 02 08 0a 0f b3 61 1d 0f b3 61 1d .....a...a..
```

图 7: 第二次握手 ACK

可以发现第二次握手中的 ACK 为 1915658152,正好为第一次握手中的 Seq 序列号 1915658151 加一后的结果。

接下来我们来看看 Flags 中的 ACK 响应位的变化情况,在第一次握手的过程之中,打开具体的字段信息,可以发现此时的 ACK 响应位为 0,而 SYN 位则为 1。

```
[Stream index: 0]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 0]
Sequence Number: 0 (relative sequence number)
Sequence Number (raw): 1915658151
[Next Sequence Number: 1 (relative sequence number)]
Acknowledgment Number: 0
Acknowledgment number (raw): 0
1010 .... = Header Length: 40 bytes (10)
v Flags: 0x002 (SYN)
 000. .... = Reserved: Not set
...0 .... = Accurate ECN: Not set

0000 02 00 00 00 45 00 00 3c 95 da 40 00 40 06 00 00 ....E...<..@..@...
0010 7f 00 00 01 7f 00 00 01 fa 03 13 88 72 2e 9f a7 .....r.....
0020 00 00 00 00 a0 02 ff ff bf a6 00 00 02 04 ff d7 .....
0030 01 03 03 08 04 02 08 0a 0f b3 61 1d 00 00 00 00 .....a.....
```

图 8: 第一次握手字段信息

而在第二次握手的过程之中,就可以清晰地看到字段信息中的 ACK 位的变化了,具体变化展示如下:

```
Acknowledgment Number: 1 (relative ack number)
Acknowledgment number (raw): 1915658152
1010 .... = Header Length: 40 bytes (10)
v 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

0000 02 00 00 00 45 00 00 3c 95 db 40 00 40 06 00 00 ....E...<..@..@...
0010 7f 00 00 01 7f 00 00 01 13 88 fa 03 3d fd a4 59 .....=...Y
0020 72 2e 9f a8 a0 12 ff ff 6c 6e 00 00 02 04 ff d7 r.....ln.....
0030 01 03 03 08 04 02 08 0a 0f b3 61 1d 0f b3 61 1d .....a...a..
```

图 9: 第二次握手字段信息

可以看到字段信息中,第五位 ACK 应答位变化为了 1。

(二) HTTP 请求交互

HTTP 请求报文格式如下：

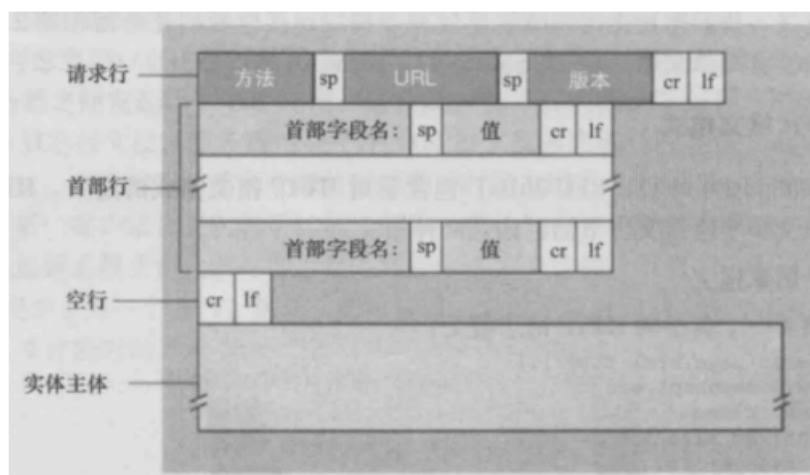


图 10: HTTP 请求格式

HTTP 请求是指从客户端到服务器端的请求消息。包括：消息首行中，对资源的请求方法、资源的标识符及使用的协议。

有 GET, POST, HEAD 等类型：GET 方法是获取 URL 指定资源。使用 GET 方法时，可以将请求参数和对应的值附加在 URL 后面 POST 请求一般是客户端提交给服务器的表单数据。当然，如果是上传文件，也应当使用 POST 请求。POST 请求还可以较 GET 请求更好的安全性。HEAD 方法与 GET 用法相同，但没有响应实体。

需要注意的是，我们在 Wireshark 捕获到的带有 Protocol 信息为 HTTP 的数据报文，通常都包含了其从 HTTP 报文转换为 TCP 数据包，再经过 IP 网络层封装的过程，因为在本次设计之中，使用的是本机的网卡即 127.0.0.1 的 IP 地址，所以 Wireshark 中捕获到的数据包的封装结构只会延续到 IP 层，我们不再需要 IP 层去寻找一个前往目标 IP 的路径，因为本机访问本机 IP。

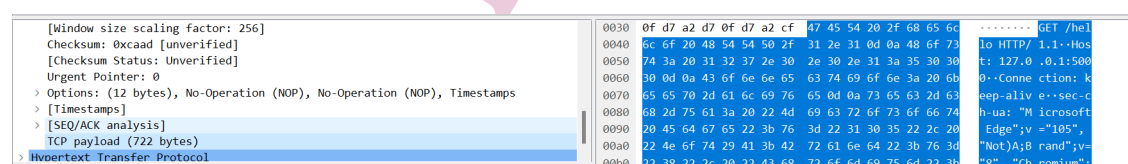


图 11: HTTP 报文

从中我们能够看到，TCP 数据包中对 HTTP 报文的信息封装到 TCP Data Segment 的过程，因为在本次设计之中的传输的信息数据量不是很大，所以看不到将 HTTP 报文切分为多个 TCP 数据包的过程。

我们不仅能够看到 HTTP 请求的发送，也能看到服务器对浏览器的响应，都会对应返回一个状态码，在本次实验之中第一次请求 HTML 文档成功时会显示状态码为 200，而在访问 Logo.png 和 CSS 文件资源时会显示 304 状态码，在访问网页的 icon 图标的时候，因为我没有设置/favicon.ico 所以会显示 404 状态码，表示未找到资源。

No.	Time	Source	Destination	Protocol	Length	Info
4	0.000049	127.0.0.1	127.0.0.1	TCP	64	8000 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
5	0.000314	127.0.0.1	127.0.0.1	TCP	64	5000 → 8000 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
6	0.000314	127.0.0.1	127.0.0.1	TCP	56	64004 → 5000 [ACK] Seq=1 Ack=1 Win=261888 Len=0 TSval=263414045 TSecr=
7	0.723980	127.0.0.1	127.0.0.1	HTTP	759	GET /hello HTTP/1.1
8	0.724024	127.0.0.1	127.0.0.1	TCP	56	5000 → 64003 [ACK] Seq=1 Ack=704 Win=2160128 Len=0 TSval=263414769 TSecr=
9	0.725461	127.0.0.1	127.0.0.1	TCP	229	5000 → 64003 [PSH, ACK] Seq=1 Ack=704 Win=2160128 Len=173 TSval=263414770 TSecr=
10	0.725502	127.0.0.1	127.0.0.1	TCP	56	64003 → 5000 [ACK] Seq=704 Ack=174 Win=327168 Len=0 TSval=263414770 TSecr=
11	0.725535	127.0.0.1	127.0.0.1	HTTP	586	HTTP/1.1 200 OK (text/html)
12	0.725535	127.0.0.1	127.0.0.1	TCP	56	64003 → 5000 [ACK] Seq=704 Ack=704 Win=326556 Len=0 TSval=263414770 TSecr=
13	0.725612	127.0.0.1	127.0.0.1	TCP	56	5000 → 64003 [FIN, ACK] Seq=704 Ack=704 Win=2160128 Len=0 TSval=263414770 TSecr=
14	0.725628	127.0.0.1	127.0.0.1	TCP	56	64003 → 5000 [ACK] Seq=704 Ack=705 Win=326556 Len=0 TSval=263414770 TSecr=

[TCP Segment Len: 173]	0000 02 00 00 00 45 00 00 e1 95 ec 40 00 40 06 00 00 ...E...@...Z
Sequence Number: 1 (relative sequence number)	0010 7f 00 00 01 7f 00 00 01 13 88 fa 03 3d fd a4 5aZ
Sequence Number (raw): 1040032858	0020 72 2e a2 67 80 18 20 f6 ac 42 00 00 01 01 08 0a r.g...B.....
[Next Sequence Number: 174 (relative sequence number)]	0030 0f b3 63 f2 0f b3 63 f1 48 54 54 50 2f 31 2e 31 ...C...HTTP/1.1
Acknowledgment Number: 704 (relative ack number)	0040 20 32 30 30 20 4f 4b 0d 0a 51 65 72 76 65 72 3a 200 OK -Server: Werkzeug/2.2.12
Acknowledgment number (raw): 191558855	0050 20 32 65 72 6b 7a 65 75 67 2f 32 3e 32 2e 32 20 Werkzeug/2.2.12
1000 = Header Length: 32 bytes (8)	0060 65 75 7a 6b 6f 6e 2f 33 2e 37 2e 30 0d 0a 44 63 python/3.7.2 (Ubuntu)
Flags: 0x018 (PSH, ACK)	0070 74 65 3a 20 51 75 6e 2c 20 32 31 20 4f 63 74 20 See: Sun, 23 Oct
Urgent Pointer: 0	0080 32 30 32 32 20 11 30 3a 30 30 3a 31 35 20 47 4d 2022 10: 08:15 GMT
Window: 8438	0090 54 0d 0a 43 6f 6e 74 65 6e 74 2d 54 79 70 65 3a Content-Type: text/html; charset=utf-8
[Calculated window size: 2160128]	00a0 20 74 65 70 74 2d 6f 74 6d 6e 30 20 63 6b 63 72 Content-Length: 530
[Window size scaling factor: 256]	00b0 23 65 74 3d 75 74 6e 20 30 0d 0a 43 6f 6e 74 65 Commit: mini closure
Checksum: 0xac42 [unverified]	00c0 6e 74 2d 4c 65 6e 67 74 68 3a 20 35 33 30 0d 0a
[Checksum Status: Unverified]	00d0 43 6f 6e 6e 65 63 74 69 6f 6e 3a 20 63 6e 6f 73
Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps	00e0 05 0d 0a 0d 0a
[Timestamps]	
[SEQ/ACK analysis]	
TCP payload (173 bytes)	
Reassembled PDU in frame: 111	
TCP segment data (173 bytes)	

图 12: TCP 段的 Data

实验中的三种不同 HTTP 响应状态码：

1. 状态码 200

OK：请求成功，一般用于 GET 与 POST 请求。

2. 状态码 304

Not Modified：未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。客户端通常会缓存访问过的资源，通过提供一个头信息指出客户端希望只返回在指定日期之后修改的资源。

3. 状态码 404

Not Found：服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置“您所请求的资源无法找到”的个性页面。

因为在本次实验使用 Wireshark 捕获数据包之前，已经访问过该网页的图片和 CSS 资源了，这些资源都已经被存储在了 Web 的缓存之中，所以 HTTP 请求的应答状态码为 304。

其余就是在本次实验之中，除了基本的 ACK 响应回答之外，还能看到 TCP 数据报文之中的 PSH 字段被设置，显示为 PSH 标志。它的英文单词是 PUSH，表示“推”的意思。TCP 模块什么时候将数据发送出去（从发送缓冲区中取数据），以及 read 函数什么时候将数据从接收缓冲区读取都是未知的。如果使用 PSH 标志，那么这件事就可以确定下来了。

如果接收方接收到了某个 TCP 报文段包含了 PSH 标志，则立即将缓冲区中的所有数据推送给应用进程（read 函数返回），当然有时候接收缓冲区满了，也会推送。

（三）TCP 四次挥手断开连接

服务器与客户端通过 TCP 四次挥手的过程进行断开连接，需要四次挥手的原因就是，服务端和客户端需要都断开各自的发送和接受功能，为了避免有一方有尚未发送的数据包，所以需要 TCP 进行四次挥手才能够断开连接。

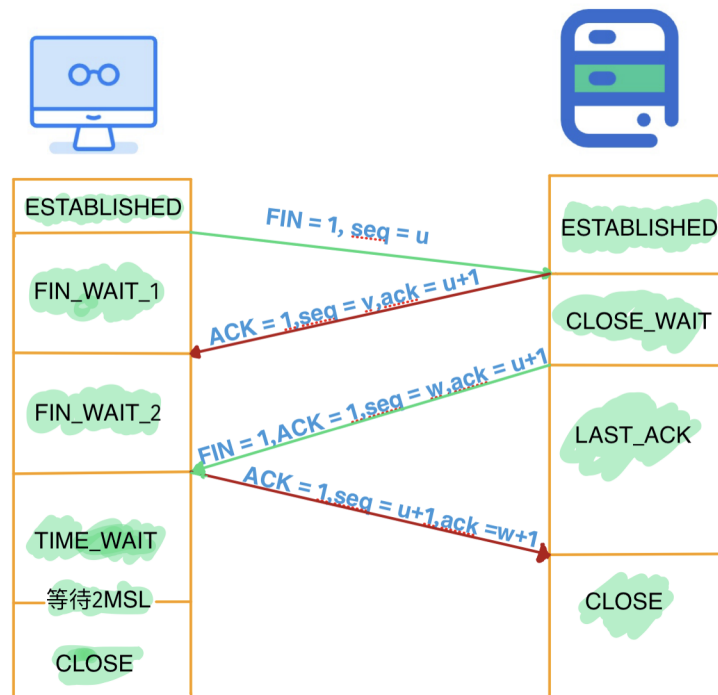


图 13: TCP 四次挥手

1. 第一次挥手

客户端打算断开连接, 向服务器发送 FIN 报文 (FIN 标记位被设置为 1, 1 表示为 FIN, 0 表示不是), FIN 报文中会指定一个序列号, 之后客户端进入 FIN_WAIT_1 状态。也就是客户端发出连接释放报文段 (FIN 报文), 指定序列号 $Seq = u$, 主动关闭 TCP 连接, 等待服务器的确认。

2. 第二次挥手

服务器收到连接释放报文段 (FIN 报文) 后, 就向客户端发送 ACK 应答报文, 以客户端的 FIN 报文的序列号 $Seq+1$ 作为 ACK 应答报文段的确认序列号 $ACK=Seq+1=u+1$ 。接着服务器进入 CLOSE_WAIT(等待关闭) 状态, 此时的 TCP 处于半关闭状态, 客户端到服务器的连接释放。客户端收到来自服务器的 ACK 应答报文段后, 进入 FIN_WAIT_2 状态。

3. 第三次挥手

服务器也打算断开连接, 向客户端发送连接释放 (FIN) 报文段, 之后服务器进入 LAST_ACK(最后确认) 状态, 等待客户端的确认。服务器的连接释放 (FIN) 报文段的 $FIN=1$, $ACK=1$, 序列号 $Seq=m$, 确认序列号 $ACK=u+1$ 。

4. 第四次挥手

客户端收到来自服务器的连接释放 (FIN) 报文段后, 会向服务器发送一个 ACK 应答报文段, 以连接释放 (FIN) 报文段的确认序号 ACK 作为 ACK 应答报文段的序列号 Seq, 以连接释放 (FIN) 报文段的序列号 $Seq+1$ 作为确认序号 ACK。

由客户端到服务器需要一个 FIN 和 ACK, 再由服务器到客户端需要一个 FIN 和 ACK, 因此通常被称为四次挥手。客户端和服务器都可以主动关闭连接, 只有率先请求关闭的一方才会进入 TIME_WAIT(时间等待状态)。具体 ACK 变化的细节这里就不赘述了, 其中细节的变化类似于 TCP 三次握手过程。

三、 总结

通过本次实验对服务器和 Wireshark 捕获的相关知识有了进一步的了解, 且对于 TCP 和 HTTP 的相关知识在基于课程讲解的基础上有了更深刻的理解。对于 TCP 和 HTTP 报文中的具体字段有了初步的了解, 学习了轻量级 Web 开发框架 Flask 的基础运用, 对整体的协议的封装与转换有了进一步的认识。

NIUB

参考文献

<https://blog.csdn.net/sunshine612/article/details/105774152>

<https://blog.csdn.net/LOOKTOMMER/article/details/121307137>

<https://blog.csdn.net/q1007729991/article/details/70154359>

NIKU