



南开大学
Nankai University

南 开 大 学

计 算 机 学 院

计算机网络实验报告

配置 Web 服务器，编写简单页面，分析交互过程

马永田

年级：2020 级

专业：计算机科学与技术

指导教师：张建忠 & 徐敬东

2022 年 10 月 28 日

摘要

搭建一个简单的 Web 服务器并使用 Wireshark 捕获浏览器与 Web 服务器的交互过程。

关键字：Web Wireshark TCP HTTP

目录

一、 实验要求	1
二、 实验流程	1
(一) Web 服务器搭建	1
(二) 编写 Web 页面	1
(三) Wireshark 捕获	2
(四) 交互过程分析	3
1. TCP 与 TCP 连接	3
2. TCP 连接建立	3
3. HTTP 请求	5
4. TCP 连接断开	5
(五) 特殊现象分析	6

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。
3. 提交实验报告。

二、实验流程

（一）Web 服务器搭建

Flask 是一个强大且简洁的 Python 开发的 Web 框架, 基于 WerkzeugWSGI 工具箱和 Jinja2 模板引擎, 可以快速搭建 web 服务器, 非常的简单, 高效。本次实验便是使用 Flask 搭建了一个简单的 Web 服务器, 代码如下:

```
1 from flask import Flask, request
2 #render_template, request, flash, get_flashed_messages
3 from flask import render_template #与html交互
4 from os import path
5 web = Flask(__name__)
6
7 @web.route('/', methods=["POST", "GET"]) #主页
8 def hello_world():
9     if request.method == "GET":
10         return render_template("index.html")
11
12 web.run(port=5000, debug=True)
```

（二）编写 Web 页面

编写简单的 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。

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5     <meta charset="UTF-8">
6     <meta http-equiv="X-UA-Compatible" content="IE=edge">
7     <meta name="viewport" content="width=device-width, initial-scale=1.0">
8     <title>Document</title>
9 </head>
10
11 <body>
12     
13     <ul>
14         <li>马永田</li>
```

```
15     <li>2012911</li>
16     <li>计算机科学与技术</li>
17 </ul>
18 </body>
19 <style>
20     body {
21         width: 600px;
22         margin: 50px 43%;
23         /* 5像素的黑色实线边框 */
24         background-color: #cccc1a2;
25     }
26
27     img {
28         width: 200px;
29         height: 200px;
30         display: block;
31     }
32 </style>
33
34 </html>
```

启动 Web 服务器并通过浏览器获取编写的 Web 页面，输入本机 IP 地址和相应端口号，界面如图1所示：

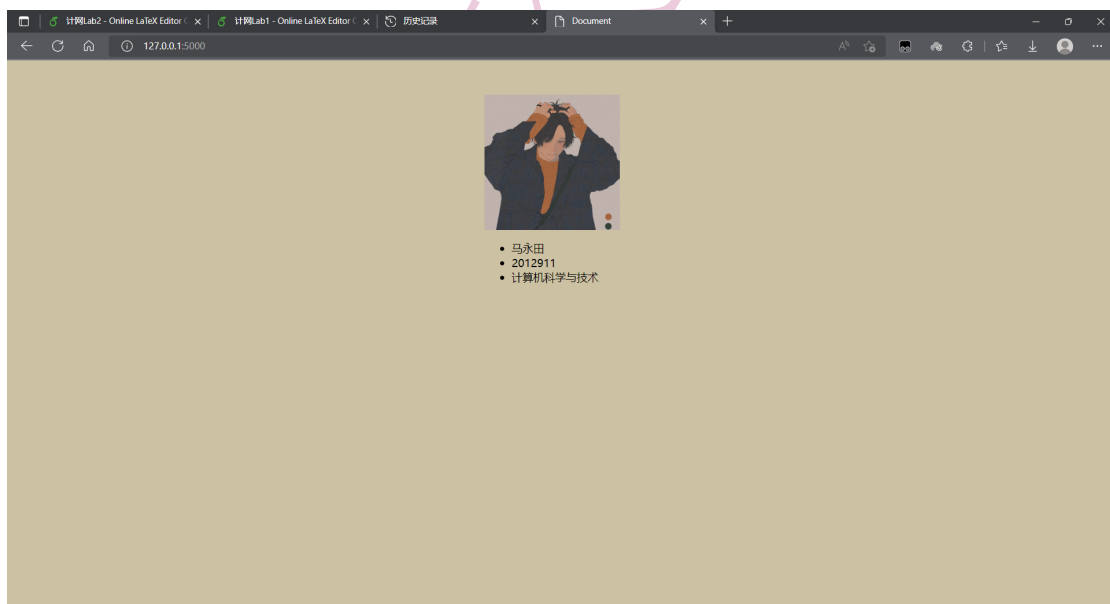


图 1: Web 网页

(三) Wireshark 捕获

打开 Wireshark，选择捕获 Adapter for loopback traffic capture 接口，应用显示过滤器设置为：`ip.addr == 127.0.0.1 and tcp.port == 5000`

之后使用浏览器访问 Web 服务器，Wireshark 捕获结果如图2所示

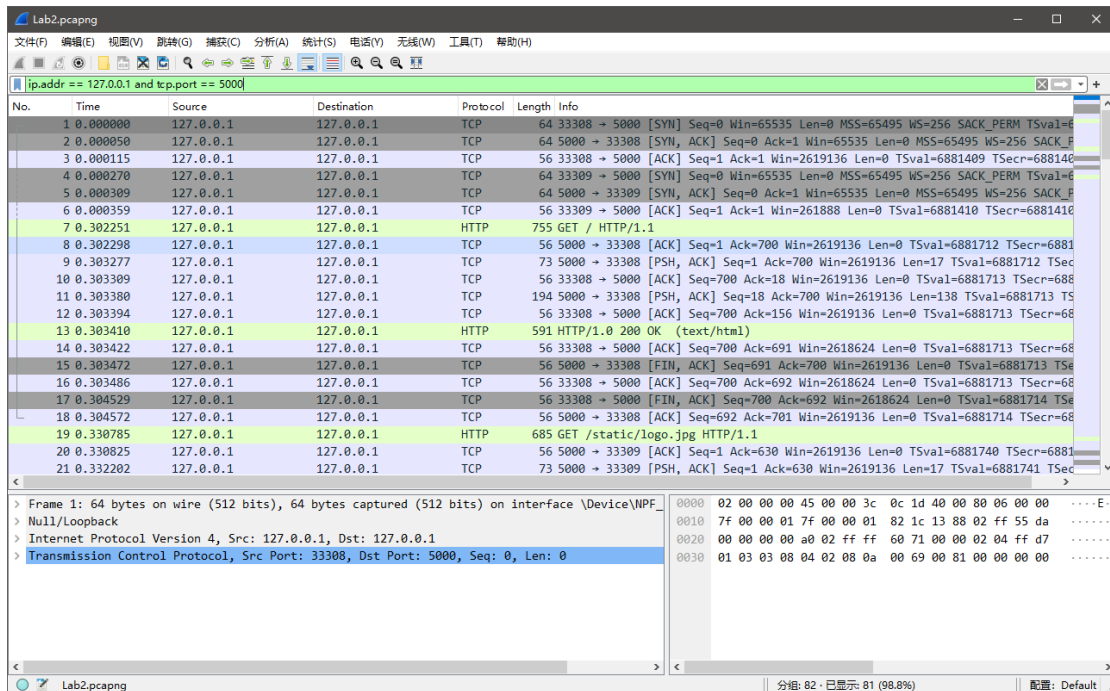


图 2: Wireshar 捕获的数据包

(四) 交互过程分析

1. TCP 与 TCP 连接

什么是 TCP? TCP 是面向连接的、可靠的、基于字节流的传输层通信协议。

- 面向连接：一定是「一对一」才能连接，不能像 UDP 协议可以一个主机同时向多个主机发送消息，也就是一对多是无法做到的；
- 可靠的：无论的网络链路中出现了怎样的链路变化，TCP 都可以保证一个报文一定能够到达接收端；
- 字节流：用户消息通过 TCP 协议传输时，消息可能会被操作系统「分组」成多个的 TCP 报文，如果接收方的程序如果不知道「消息的边界」，是无法读出一个有效的用户消息的。并且 TCP 报文是「有序的」，当「前一个」TCP 报文没有收到的时候，即使它先收到了后面的 TCP 报文，那么也不能扔给应用层去处理，同时对「重复」的 TCP 报文会自动丢弃。

什么是 TCP 连接? 简单来说就是，用于保证可靠性和流量控制维护的某些状态信息，这些信息的组合，包括 Socket、序列号和窗口大小称为连接。

所以，建立一个 TCP 连接是需要客户端与服务端端达成上述三个信息的共识。

- Socket：由 IP 地址和端口号组成
- 序列号：用来解决乱序问题等
- 窗口大小：用来做流量控制

2. TCP 连接建立

TCP 三次握手过程是怎样的? TCP 是面向连接的协议，所以使用 TCP 前必须先建立连接，而建立连接是通过三次握手来进行的。三次握手的过程如下图所示：

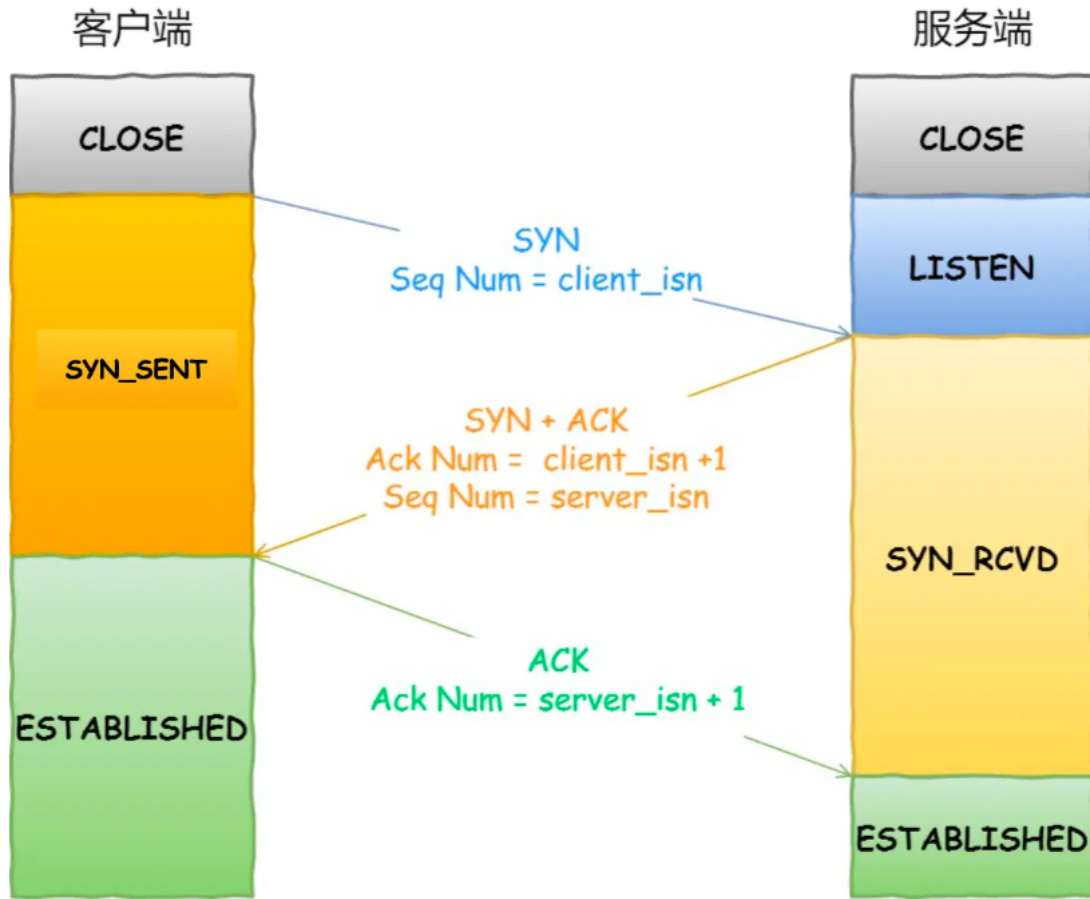


图 3: TCP 三次握手过程

- 一开始,客户端和服务端都处于 CLOSE 状态。先是服务端主动监听某个端口,处于 LISTEN 状态
- 客户端会随机初始化序号 (`client_isn`), 将此序号置于 TCP 首部的「序号」字段中, 同时把 SYN 标志位置为 1, 表示 SYN 报文。接着把第一个 SYN 报文发送给服务端, 表示向服务端发起连接, 该报文不包含应用层数据, 之后客户端处于 SYN-SENT 状态。
- 服务端收到客户端的 SYN 报文后, 首先服务端也随机初始化自己的序号 (`server_isn`), 将此序号填入 TCP 首部的「序号」字段中, 其次把 TCP 首部的「确认应答号」字段填入 `client_isn+1`, 接着把 SYN 和 ACK 标志位置为 1。最后把该报文发给客户端, 该报文也不包含应用层数据, 之后服务端处于 SYN-RCVD 状态。
- 客户端收到服务端报文后, 还要向服务端回应最后一个应答报文, 首先该应答报文 TCP 首部 ACK 标志位置为 1, 其次「确认应答号」字段填入 `server_isn + 1`, 最后把报文发送给服务端, 这次报文可以携带客户到服务端的数据, 之后客户端处于 ESTABLISHED 状态。
- 服务端收到客户端的应答报文后, 也进入 ESTABLISHED 状态。

整个过程中第三次握手是可以携带数据的, 前两次握手是不可以携带数据的。一旦完成三次握手, 双方都处于 ESTABLISHED 状态, 此时连接就已建立完成, 客户端和服务端就可以相互发送数据了。

3. HTTP 请求

HTTP 协议是因特网上应用最为广泛的一种网络传输协议，所有的 WWW 文件都必须遵守这个标准。HTTP 基于 TCP/IP 通信协议来传递数据（HTML 文件，图片文件，查询结果等）。

HTTP 协议工作于客户端-服务端架构上。浏览器作为 HTTP 客户端通过 URL 向 HTTP 服务端即 WEB 服务器发送所有请求。Web 服务器根据接收到的请求后，向客户端发送响应信息。

从捕获的数据包中可以看到在成功建立 TCP 连接后，客户端会发起 GET 请求获取 html 和 jpg 图片，即我们的前端页面及其中的 logo 图片，可以看到是采用的 HTTP1.1，支持流水线工作，因此在第一个 TCP 连接还未断开时就会创建第二个 TCP 连接。

4. TCP 连接断开

TCP 断开连接是通过四次挥手方式。双方都可以主动断开连接，断开连接后主机中的「资源」将被释放，四次挥手的过程如下图：

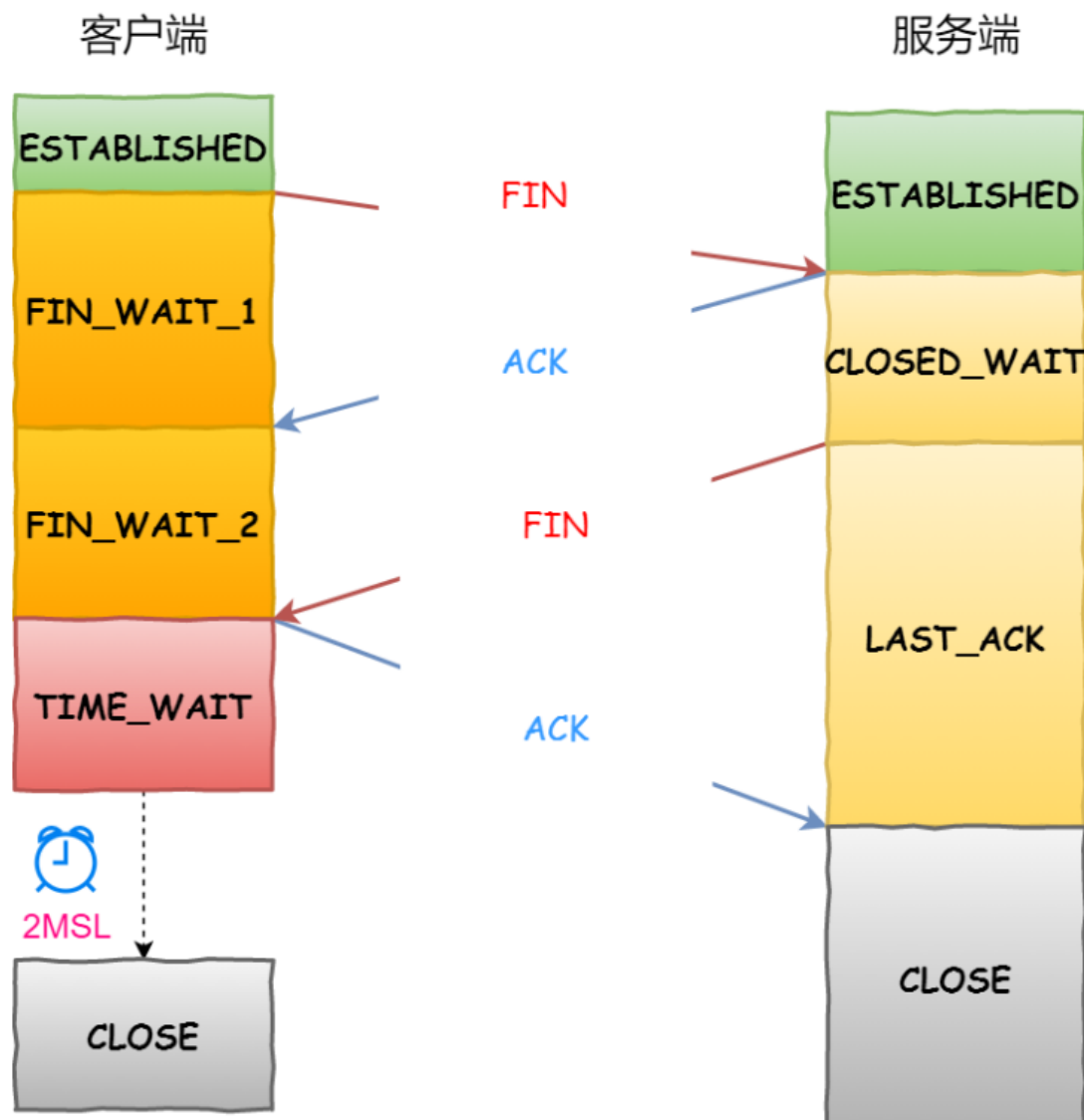


图 4: TCP 四次挥手过程

- 客户端打算关闭连接，此时会发送一个 TCP 首部 FIN 标志位被置为 1 的报文也即 FIN 报文，之后客户端进入 FIN_WAIT_1 状态。服务端收到该报文后，就向客户端发送 ACK 应答报文，接着服务端进入 CLOSE_WAIT 状态。
- 客户端收到服务端的 ACK 应答报文后，之后进入 FIN_WAIT_2 状态。等待服务端处理完数据后，也向客户端发送 FIN 报文，之后服务端进入 LAST_ACK 状态。
- 客户端收到服务端的 FIN 报文后，回一个 ACK 应答报文，之后进入 TIME_WAIT 状态。服务端收到了 ACK 应答报文后，就进入了 CLOSE 状态，至此服务端已经完成连接的关闭。

客户端在经过 2MSL 一段时间后，自动进入 CLOSE 状态，至此客户端也完成连接的关闭。由于每个方向都需要一个 FIN 和一个 ACK，因此通常被称为四次挥手。

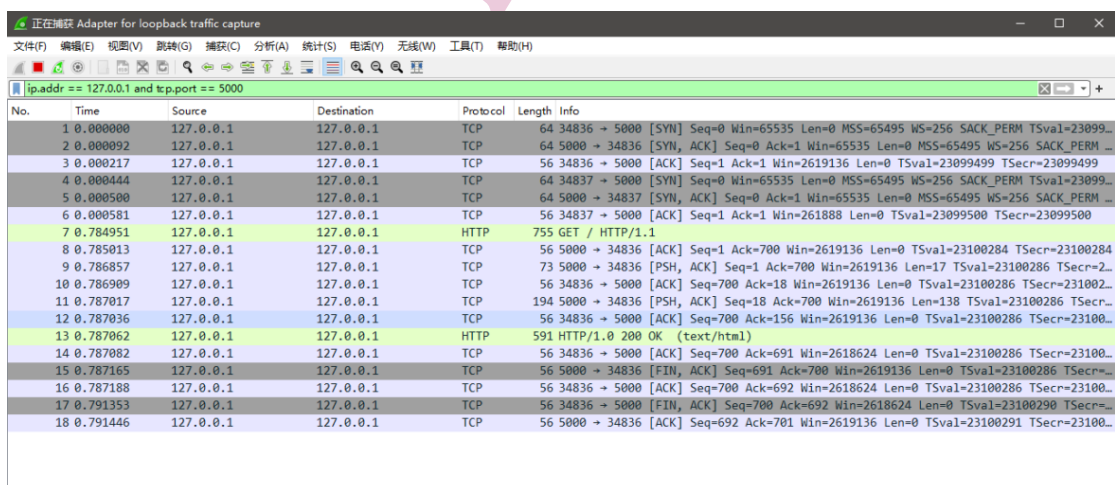
其中只有主动关闭连接的，才有 TIME_WAIT 状态；而 TIME_WAIT 会等待 2 倍的 MSL，这是由于网络中可能存在来自发送方的数据包，当这些发送方的数据包被接收方处理后又会对方向发送响应，所以一来一回需要等待 2 倍的时间。

例如，如果被动关闭方没有收到断开连接的最后的 ACK 报文，就会触发超时重发 FIN 报文，另一方接收到 FIN 后，会重发 ACK 给被动关闭方，一来一去正好 2 个 MSL。可以看到 2MSL 时长其实是相当于至少允许报文丢失一次。比如，若 ACK 在一个 MSL 内丢失，这样被动方重发的 FIN 会在第 2 个 MSL 内到达，TIME_WAIT 状态的连接可以应答，而实际上连续两次丢包的概率只有万分之一，这个概率实在是太小，因此只等待 2MSL 的时长。

(五) 特殊现象分析

实验中观察 Wireshark 捕获的数据包发现，在访问 Web 过程中 TCP 会建立两次连接，但只有第一个连接会发起 HTTP 请求来获取文件，上网查阅相关资料后得知是浏览器的特性，谷歌浏览器默认设置了预加载选项，建立多个连接用于加速访问。

此外还出现了如图 5 中建立连接后没有发起 GET 请求获取 logo 图片的情况，多次尝试后发现是由于浏览器对图片进行了缓存，在实验前先清除浏览器中的缓存图片，之后再访问与数据包的捕获即可得到我们想要的结果。



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	64	34836 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM TSval=23099...
2	0.000092	127.0.0.1	127.0.0.1	TCP	64	5000 → 34836 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM TS...
3	0.000217	127.0.0.1	127.0.0.1	TCP	56	34836 → 5000 [ACK] Seq=1 Ack=1 Win=2619136 Len=0 TSval=23099499 TSecr=23099499
4	0.000444	127.0.0.1	127.0.0.1	TCP	64	34837 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM TSval=23099...
5	0.000500	127.0.0.1	127.0.0.1	TCP	64	5000 → 34837 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM TS...
6	0.000581	127.0.0.1	127.0.0.1	TCP	56	34837 → 5000 [ACK] Seq=1 Ack=1 Win=261888 Len=0 TSval=23099500 TSecr=23099500
7	0.784951	127.0.0.1	127.0.0.1	HTTP	755	GET / HTTP/1.1
8	0.785013	127.0.0.1	127.0.0.1	TCP	56	5000 → 34836 [ACK] Seq=1 Ack=700 Win=2619136 Len=0 TSval=23100284 TSecr=23100284
9	0.786857	127.0.0.1	127.0.0.1	TCP	73	5000 → 34836 [PSH, ACK] Seq=1 Ack=700 Win=2619136 Len=17 TSval=23100286 TSecr=2...
10	0.786909	127.0.0.1	127.0.0.1	TCP	56	34836 → 5000 [ACK] Seq=700 Ack=18 Win=2619136 Len=0 TSval=23100286 TSecr=231002...
11	0.787017	127.0.0.1	127.0.0.1	TCP	194	5000 → 34836 [PSH, ACK] Seq=18 Ack=700 Win=2619136 Len=138 TSval=23100286 TSecr=...
12	0.787036	127.0.0.1	127.0.0.1	TCP	56	34836 → 5000 [ACK] Seq=700 Ack=156 Win=2619136 Len=0 TSval=23100286 TSecr=23100...
13	0.787062	127.0.0.1	127.0.0.1	HTTP	591	HTTP/1.0 200 OK (text/html)
14	0.787082	127.0.0.1	127.0.0.1	TCP	56	34836 → 5000 [ACK] Seq=700 Ack=691 Win=2618624 Len=0 TSval=23100286 TSecr=23100...
15	0.787165	127.0.0.1	127.0.0.1	TCP	56	5000 → 34836 [FIN, ACK] Seq=691 Ack=700 Win=2619136 Len=0 TSval=23100286 TSecr=...
16	0.787188	127.0.0.1	127.0.0.1	TCP	56	34836 → 5000 [ACK] Seq=700 Ack=692 Win=2618624 Len=0 TSval=23100286 TSecr=23100...
17	0.791353	127.0.0.1	127.0.0.1	TCP	56	34836 → 5000 [FIN, ACK] Seq=700 Ack=692 Win=2618624 Len=0 TSval=23100290 TSecr=...
18	0.791446	127.0.0.1	127.0.0.1	TCP	56	5000 → 34836 [ACK] Seq=692 Ack=701 Win=2619136 Len=0 TSval=23100291 TSecr=23100...

图 5: Wireshark 捕获数据包