南開大學
Nankai University

计算机学院

计算机网络实验报告

web 服务器搭建与报文捕获

姓名：赵一名

学号：2013922

专业：计算机科学与技术

2022 年 11 月 5 日

目录

1 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。

2 检查作业中遇到的问题

TCP 断开连接时为何是四次挥手?

当服务端收到客户端发送过来的 FIN 断开请求时，回复 ACK 后只是断开了 client -> server 方向的连接，服务端还可以继续向客户端发送数据（若数据没有发送完），待服务端数据发送完后，服务端也发送一个 FIN，客户端回复 ACK，此时断开了 server->client 方向的连接，之后双方都可以释放 TCP 连接所占用的资源。

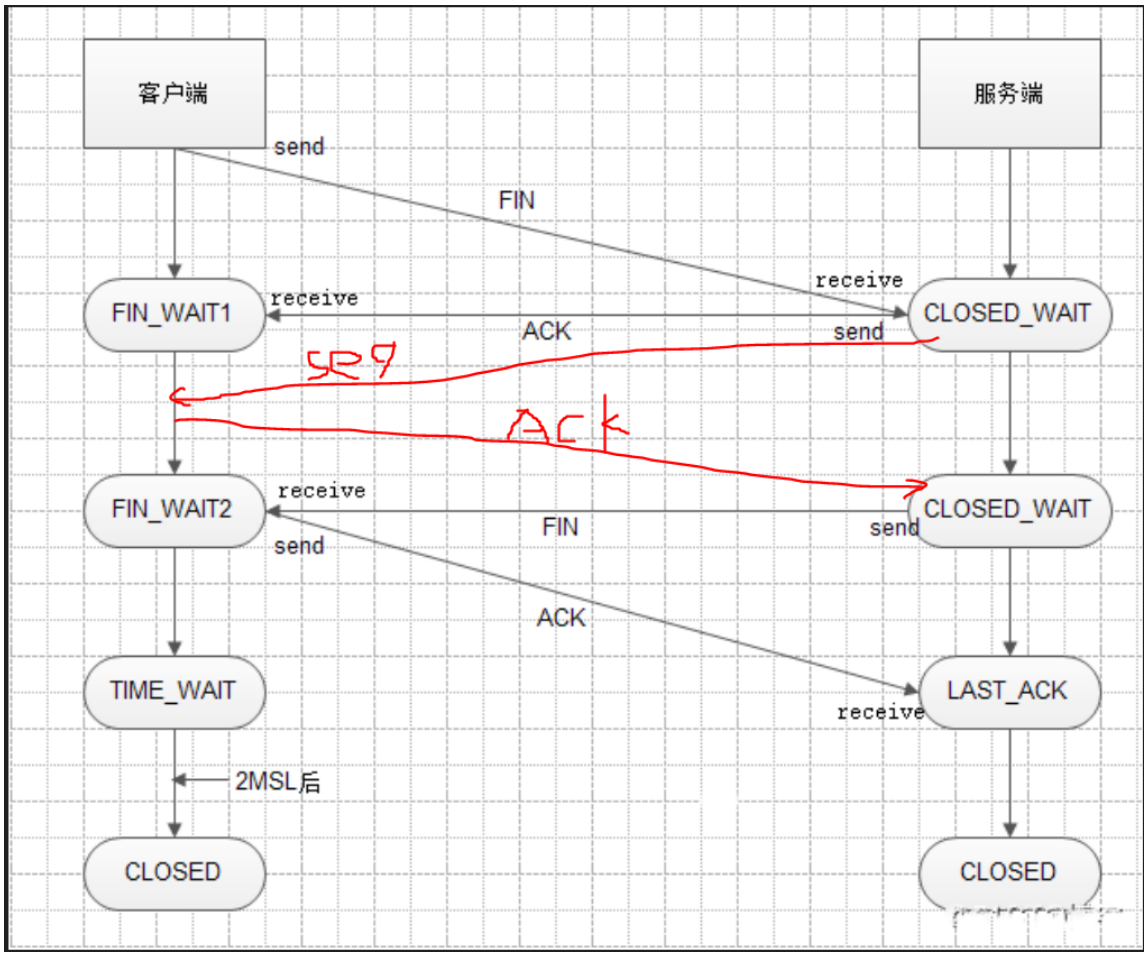


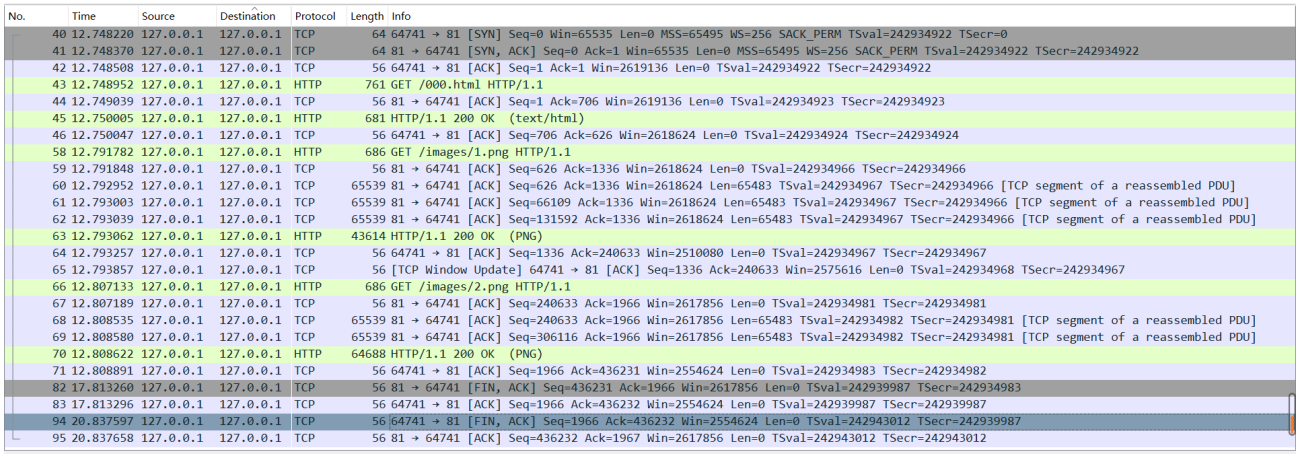
图 2.1: 四次挥手

上述情况体现在四次握手的示意图中就是上图的两条红线所示意的发送过程（实际中可能不止一次发送），当服务端发送完所有残留数据之后向客户端发出 FIN 示意服务端也可结束连接，当服务端收到 ACK 回复后便关闭 TCP 并释放掉相关资源。

除此之外，客户端在收到服务器端的 FIN 后还要继续等待 2MSL 的时间，这是因为客户端发出的 ACK 可能中途丢失，这时候就要继续等待服务器端因超时而发出的 FIN 并再次回复 ACK。其中 MSL 是 TCP 报文的最大生命周期，等待时间达到 2MSL 可以保证在两个传输方向上的尚未接收到或者迟到的报文段已经消失

3 最终捕获结果

一个捕获网页的过程如下图所示：



No.	Time	Source	Destination	Protocol	Length	Info
40	12.748220	127.0.0.1	127.0.0.1	TCP	64	64741 → 81 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM TSval=242934922 TSecr=0
41	12.748370	127.0.0.1	127.0.0.1	TCP	64	81 → 64741 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM TSval=242934922 TSecr=242934922
42	12.748508	127.0.0.1	127.0.0.1	TCP	56	64741 → 81 [ACK] Seq=1 Ack=1 Win=2619136 Len=0 TSval=242934922 TSecr=242934922
43	12.748952	127.0.0.1	127.0.0.1	HTTP	761	GET /000.html HTTP/1.1
44	12.749039	127.0.0.1	127.0.0.1	TCP	56	81 → 64741 [ACK] Seq=1 Ack=706 Win=2619136 Len=0 TSval=242934923 TSecr=242934923
45	12.750005	127.0.0.1	127.0.0.1	HTTP	681	HTTP/1.1 200 OK (text/html)
46	12.750047	127.0.0.1	127.0.0.1	TCP	56	64741 → 81 [ACK] Seq=706 Ack=626 Win=2618624 Len=0 TSval=242934924 TSecr=242934924
58	12.791782	127.0.0.1	127.0.0.1	HTTP	686	GET /images/1.png HTTP/1.1
59	12.791848	127.0.0.1	127.0.0.1	TCP	56	81 → 64741 [ACK] Seq=626 Ack=1336 Win=2618624 Len=0 TSval=242934966 TSecr=242934966
60	12.792952	127.0.0.1	127.0.0.1	TCP	65539	81 → 64741 [ACK] Seq=626 Ack=1336 Win=2618624 Len=65483 TSval=242934967 TSecr=242934966 [TCP segment of a reassembled PDU]
61	12.793003	127.0.0.1	127.0.0.1	TCP	65539	81 → 64741 [ACK] Seq=66109 Ack=1336 Win=2618624 Len=65483 TSval=242934967 TSecr=242934966 [TCP segment of a reassembled PDU]
62	12.793039	127.0.0.1	127.0.0.1	TCP	65539	81 → 64741 [ACK] Seq=131592 Ack=1336 Win=2618624 Len=65483 TSval=242934967 TSecr=242934966 [TCP segment of a reassembled PDU]
63	12.793062	127.0.0.1	127.0.0.1	HTTP	43614	HTTP/1.1 200 OK (PNG)
64	12.793257	127.0.0.1	127.0.0.1	TCP	56	64741 → 81 [ACK] Seq=1336 Ack=240633 Win=2510080 Len=0 TSval=242934967 TSecr=242934967
65	12.793857	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 64741 → 81 [ACK] Seq=1336 Ack=240633 Win=2575616 Len=0 TSval=242934968 TSecr=242934967
66	12.807133	127.0.0.1	127.0.0.1	HTTP	686	GET /images/2.png HTTP/1.1
67	12.807189	127.0.0.1	127.0.0.1	TCP	56	81 → 64741 [ACK] Seq=240633 Ack=1966 Win=2617856 Len=0 TSval=242934981 TSecr=242934981
68	12.808535	127.0.0.1	127.0.0.1	TCP	65539	81 → 64741 [ACK] Seq=240633 Ack=1966 Win=2617856 Len=65483 TSval=242934982 TSecr=242934981 [TCP segment of a reassembled PDU]
69	12.808580	127.0.0.1	127.0.0.1	TCP	65539	81 → 64741 [ACK] Seq=306116 Ack=1966 Win=2617856 Len=65483 TSval=242934982 TSecr=242934981 [TCP segment of a reassembled PDU]
70	12.808622	127.0.0.1	127.0.0.1	HTTP	64688	HTTP/1.1 200 OK (PNG)
71	12.808891	127.0.0.1	127.0.0.1	TCP	56	64741 → 81 [ACK] Seq=1966 Ack=436231 Win=2554624 Len=0 TSval=242934983 TSecr=242934982
82	17.813260	127.0.0.1	127.0.0.1	TCP	56	81 → 64741 [FIN, ACK] Seq=436231 Ack=1966 Win=2617856 Len=0 TSval=242939987 TSecr=242934983
83	17.813296	127.0.0.1	127.0.0.1	TCP	56	64741 → 81 [ACK] Seq=1966 Ack=436232 Win=2554624 Len=0 TSval=242939987 TSecr=242939987
94	20.837597	127.0.0.1	127.0.0.1	TCP	56	64741 → 81 [FIN, ACK] Seq=1966 Ack=436232 Win=2554624 Len=0 TSval=242943012 TSecr=242939987
95	20.837658	127.0.0.1	127.0.0.1	TCP	56	81 → 64741 [ACK] Seq=436232 Ack=1967 Win=2617856 Len=0 TSval=242943012 TSecr=242943012

图 3.2: 捕获过程

从图中可以看出，前三个数据包对应于 tcp 连接的三次握手，用于确保两台主机都能够发送和接收消息。

确认可以通信后（三次握手后），客户端发送 GET 请求 HTTP 包，在本次捕获的包中可以看到，HTTP 报文大小为 761-56=705。

第四个 TCP 包的作用是确认收到客户端的 GET 请求。

序号为 45 的 HTTP 包是对 GET 请求的应答，其附带响应码 200 表示成功接受，已经 html 代码的内容。

序号为 60、61、62 的三个 TCP 包用来传输图片 1.png，故包的大小远大于其他 TCP 包，达到了 65539 字节，序号为 68，69 的两个 TCP 包用来传输图片 2.png。

最后四个 TCP 包表示四次挥手断开连接。

4 实验过程

4.1 服务器搭建

使用 Apache 搭建服务器，在官网下载 Apache 软件的压缩包，解压后在命令行中输入 $httpd-tstart$ 开启 Apache 服务，之后将编写好的 html 文件放入 htdocs 目录下，使用浏览器访问本机 ip 地址 127.0.0.1 即可看到编写好的 html 页面。

4.2 wireshark 捕获界面

之后打开 wireshark 并监视对应网卡即可看到捕获的包的相关信息。

4.3 封包详细信息

首先可以看到每个包均具有以下几种信息

1.Frame: 物理层的数据帧概况

2.Ethernet II: 数据链路层以太网帧头部信息

3.Internet Protocol Version 4: 互联网层 IP 包头部信息

4.Transmission Control Protocol: 传输层的数据段头部信息，本实验中是 TCP

5.Hypertext Transfer Protocol: 应用层的信息，本实验中是 HTTP 协议

具体对应关系如下图所示：

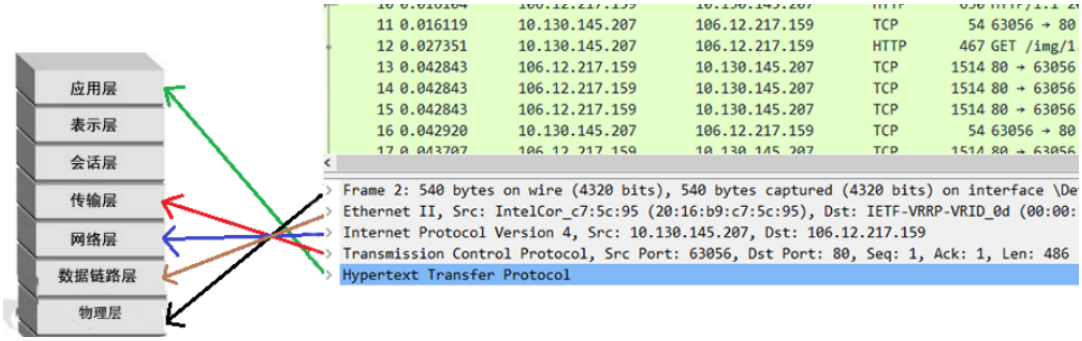


图 4.3: 对应关系

在本实验中由于直接访问本机的服务器，所以不存在 Ethernet II 所对应的数据链路层

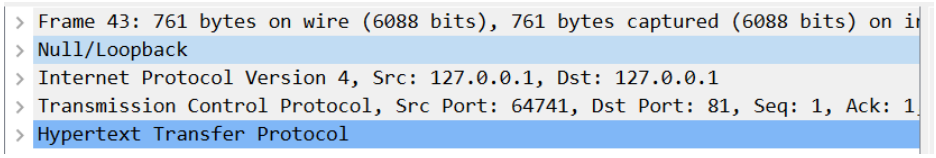


图 4.4: 缺失 Ethernet II 层

4.4 HTTP 请求与 HTTP 协议相关

浏览器向服务器发送 HTTP 的请求的过程大概分为：

1. 浏览器发送一个 HTTP 的 GET 请求，这个请求包含了 cookie 和其他的 head 头信息。

2. 服务器发送一个包含了 ACK 的 TCP 报文，表示服务器收到了浏览器的请求。

3. 服务器响应浏览器的 HTTP 请求，将请求的内容发给浏览器。

4. 服务器发送一个包含 ACK 的 TCP 报文，表示请求的内容发送完毕。

请求报文的结构如下：

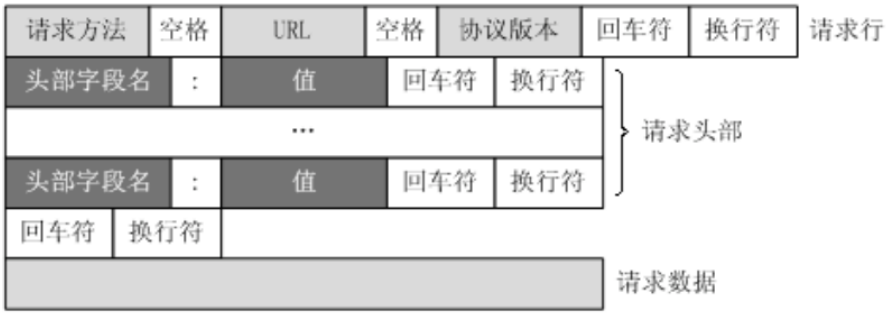


图 4.5: HTTP 请求报文结构

本实验中浏览器向服务器发送 GET 请求的 HTTP 包如下图所示。

```
Hypertext Transfer Protocol
  GET /000.html HTTP/1.1\r\n
  Host: 127.0.0.1:81\r\n
  Connection: keep-alive\r\n
  sec-ch-ua: "Chromium";v="106", "Microsoft Edge";v="106", "Not;A=Brand";v="99"\r\n
  sec-ch-ua-mobile: ?0\r\n
  sec-ch-ua-platform: "Windows"\r\n
  Upgrade-Insecure-Requests: 1\r\n
  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36 Edg/106.0.1370.52\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9\r\n
  Sec-Fetch-Site: none\r\n
  Sec-Fetch-Mode: navigate\r\n
  Sec-Fetch-User: ?1\r\n
  Sec-Fetch-Dest: document\r\n
  Accept-Encoding: gzip, deflate, br\r\n
  Accept-Language: en,zh-CN;q=0.9,zh;q=0.8,en-GB;q=0.7,en-US;q=0.6\r\n
  \r\n
  [Full request URI: http://127.0.0.1:81/000.html]
  [HTTP request 1/3]
  [Response in frame: 45]
  [Next request in frame: 58]
```

图 4.6: HTTP 包

各对应段的含义如下：

GET 为请求方式，后面跟请求的内容（这个地方可以看作是一个网页），协议版本 http 1.1

Host 为请求的主机名

Connection 客户端与服务端指定的请求，响应有关选项

User-Agent 为发送请求的操作系统、及浏览器信息

Accept 为客户端可识别的内容类型列表，用于指定客户端接受哪些类型的信息

Accept-Encoding 为客户端可识别的数据编码

Accept-language 为浏览器所支持的语言类型

可以看到，上面这个 HTTP 包的第一行 host 表明了即将建立连接的 IP 地址为 127.0.0.1，user-agent 段可以看到浏览器为 edge 浏览器。

相应报文的结构如下：

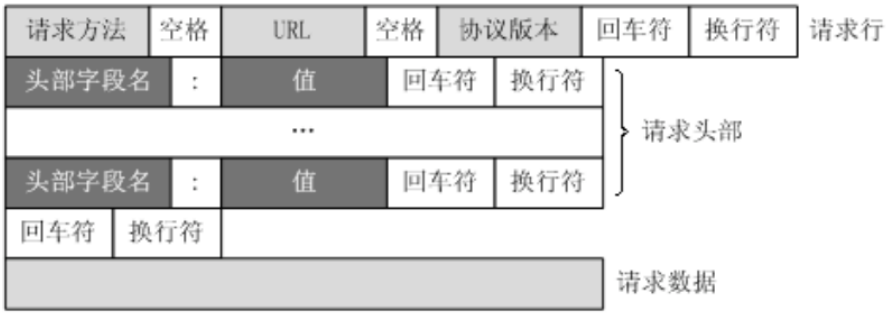


图 4.7: HTTP 发送报文结构

服务器响应的 HTTP 包如下:

```
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
  Date: Fri, 28 Oct 2022 01:08:11 GMT\r\n
  Server: Apache/2.4.54 (Win64) OpenSSL/1.1.1p\r\n
  Upgrade: h2,h2c\r\n
  Connection: Upgrade, Keep-Alive\r\n
  Last-Modified: Tue, 25 Oct 2022 02:13:13 GMT\r\n
  ETag: "12c-5ebd27349f0f1"\r\n
  Accept-Ranges: bytes\r\n
  Content-Length: 300\r\n
  Keep-Alive: timeout=5, max=100\r\n
  Content-Type: text/html\r\n
  \r\n
  [HTTP response 1/3]
  [Time since request: 0.001053000 seconds]
  [Request in frame: 43]
  [Next request in frame: 58]
  [Next response in frame: 63]
  [Request URI: http://127.0.0.1:81/000.html]
  File Data: 300 bytes
```

图 4.8: HTTP 包

各对应段的含义如下:

HTTP/1.1 200: 状态行，200 表示客户端请求成功

Server: 表示服务器信息

Content-length: 消息主体的大小

ETag: 资源的特定版本的标识符

Last-Modified: 请求资源的最后修改时间

Accept-Ranges: 用于标识下载中断时，可以尝试中断了的下载，值一般是 0，或 byte,0 表示不支持

Content-Type: 告诉客户端实际返回的内容类型

File Data: 响应报文大小

Line-based text data: 响应报文的主体，即 http 传送的内容

从上图中可以看到上述 HTTP 的状态码为 200，表示接收成功。

相应报文的结构如下:

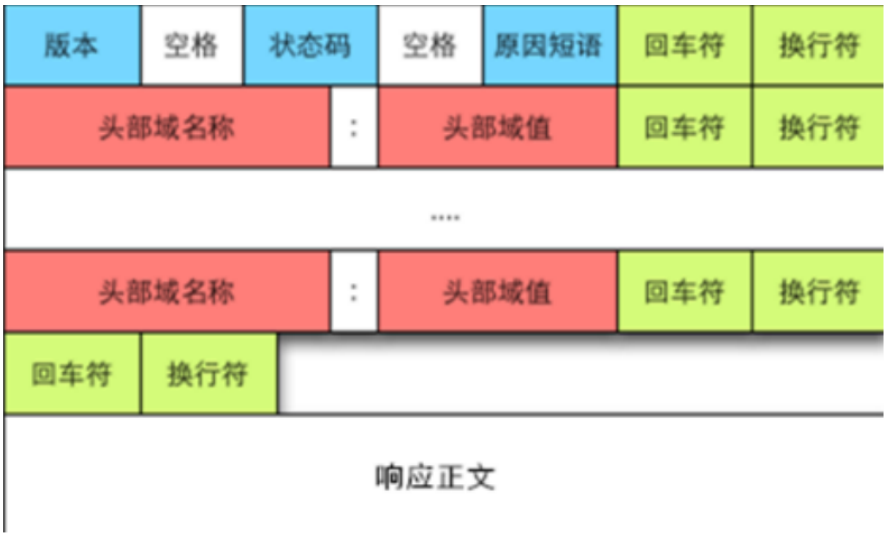


图 4.9: HTTP 响应报文结构

4.5 TCP 报文

实验中一个 TCP 包的内容如下图所示:

```
    46 12.750047 127.0.0.1   127.0.0.1   TCP        56 64741 → 81 [ACK] Seq=706 Ack=626 Win=2618624 Len=0 TSval=242934924
> Frame 46: 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
v Transmission Control Protocol, Src Port: 64741, Dst Port: 81, Seq: 706, Ack: 626, Len: 0
    Source Port: 64741
    Destination Port: 81
    [Stream index: 2]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 706    (relative sequence number)
    Sequence Number (raw): 2357219449
    [Next Sequence Number: 706    (relative sequence number)]
    Acknowledgment Number: 626    (relative ack number)
    Acknowledgment number (raw): 1240329278
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x010 (ACK)
    Window: 10229
    [Calculated window size: 2618624]
    [Window size scaling factor: 256]
    Checksum: 0x5a5b [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  > [Timestamps]
  > [SEQ/ACK analysis]
```

图 4.10: TCP 包

各个字段表示的意义如下:

端口号: 数据传输的 16 位源端口号和 16 位目标端口号 (用于寻找发端和收端应用进程);

相对序列号: 该数据包的相对序列号为 1461(此序列号用来确定传送数据的正确位置, 且序列号用来侦测丢失的包); 下一个数据包的序列号是 2921;

Acknowledgment number: 是 32 位确认序列号，值等于 1 表示数据包收到，确认有效；手动的数据包的头字节长度是 20 字节；

Flags: 含 6 种标志；ACK ： 确认序号有效；SYN ： 同步序号用来发起一个连接；FIN ： 发端完成发送任务；RST ： 重新连接；PSH ： 接收方应该尽快将这个报文段交给应用层；URG ： 紧急指针有效；

window size:TCP 的流量控制由连接的每一端通过声明的窗口大小来提供。窗口大小为字节数，起始于确认序号字段指明的值，这个值是接收端正期望接收的字节。窗口大小是一个 16bit 字段，因此窗口大小最大为 65536 字节；

Checksum:16 位校验和，检验和覆盖了整个的 TCP 报文段，由发端计算和存储，并由收端进行验证；

TCP 报文的结构如下：

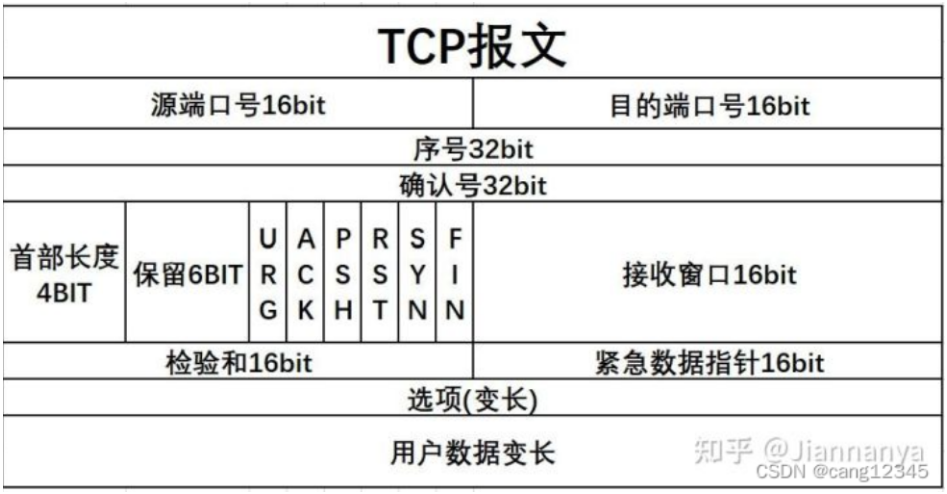


图 4.11: TCP 报文结构

4.6 三次握手

浏览器与服务器交互时的前三个报文表示三次握手的过程，具体过程如下：

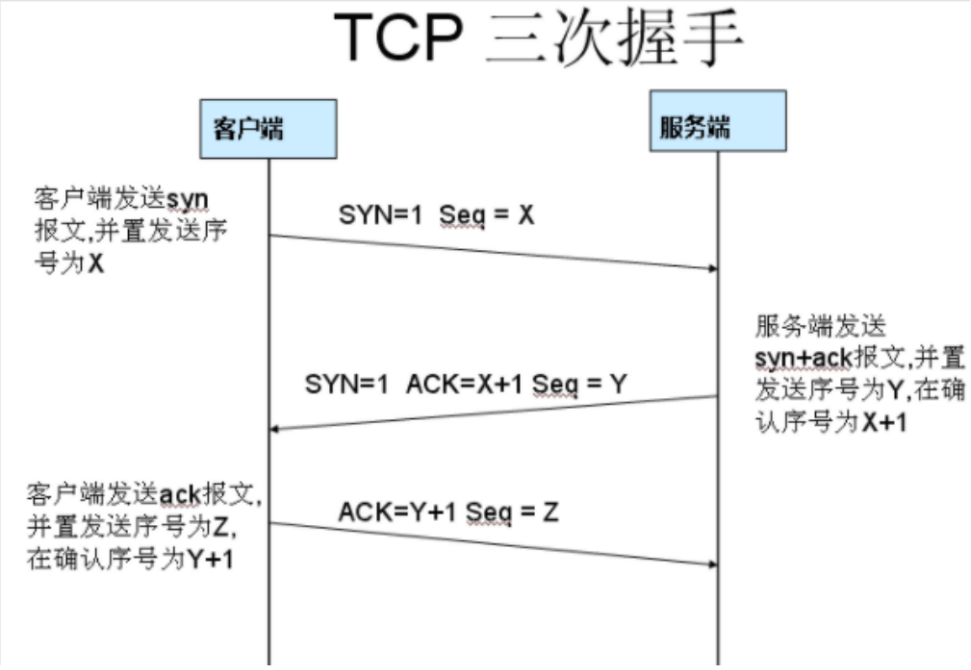


图 4.12: 三次握手

本实验中的三次握手的数据包如下图所示

1. 第一次握手数据包：客户端发送一个 TCP，标志位为 SYN，序列号为 0，代表客户端请求建立连接。

```
Source Port: 64741
Destination Port: 81
[Stream index: 2]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 0]
Sequence Number: 0    (relative sequence number)
Sequence Number (raw): 2357218743
[Next Sequence Number: 1    (relative sequence number)]
Acknowledgment Number: 0
Acknowledgment number (raw): 0
1010 .... = Header Length: 40 bytes (10)
Flags: 0x002 (SYN)
```

图 4.13: 第一次握手

2. 第二次握手数据包：服务器发回确认包，标志位为 SYN,ACK . 将确认序号设置为客户的 I S N 加 1 以. 即 0+1=1

```
Source Port: 81
Destination Port: 64741
[Stream index: 2]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 0]
Sequence Number: 0    (relative sequence number)
Sequence Number (raw): 1240328652
[Next Sequence Number: 1    (relative sequence number)]
Acknowledgment Number: 1    (relative ack number)
Acknowledgment number (raw): 2357218744
1010 .... = Header Length: 40 bytes (10)
Flags: 0x012 (SYN, ACK)
```

图 4.14: 第二次握手

3. 第三次握手数据包：客户端再次发送确认包 (ACK) SYN 标志位为 0,ACK 标志位为 1. 并且把服务器发来 ACK 的序号字段 +1, 放在确定字段中发送给对方. 并且在数据段放写 ISN 的 +1

```
Source Port: 64741
Destination Port: 81
[Stream index: 2]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 0]
Sequence Number: 1    (relative sequence number)
Sequence Number (raw): 2357218744
[Next Sequence Number: 1    (relative sequence number)]
Acknowledgment Number: 1    (relative ack number)
Acknowledgment number (raw): 1240328653
1000 .... = Header Length: 32 bytes (8)
Flags: 0x010 (ACK)
```

图 4.15: 第三次握手

4.7 四次挥手

浏览器与服务器交互时的后四个报文表示三次握手的过程，具体过程如下：

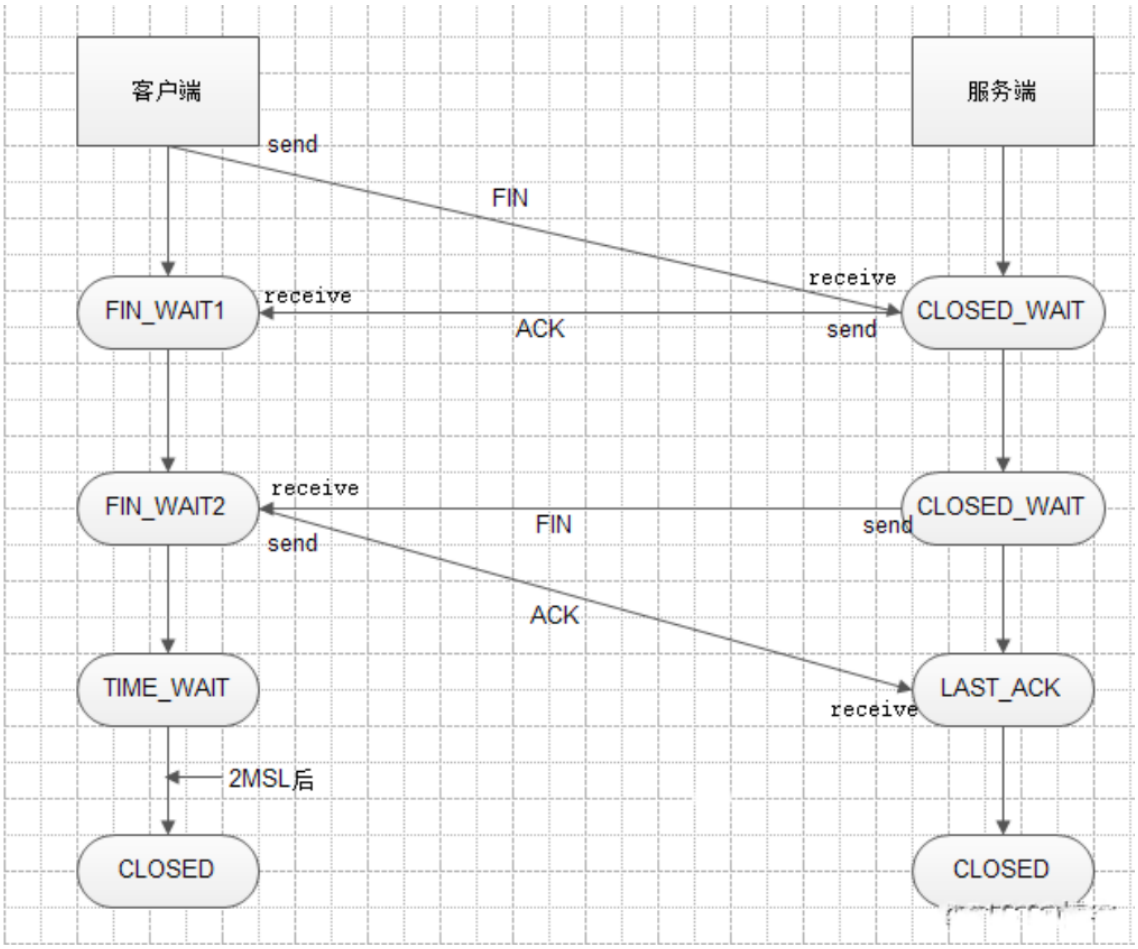


图 4.16: 四次挥手

状态转换的过程如下：

1、客户端先向服务器发送 FIN 报文，请求断开连接，其状态变为 FIN_WAIT1

2、服务器收到 FIN 后向客户端发送 ACK ，服务器的状态围边 CLOSE_WAIT

3、客户端收到 ACK 后就进入 FIN_WAIT2 状态，此时连接已经断开了一半了。如果服务器还有数据要发送给客户端，就会继续发送

4、直到发完数据，就会发送 FIN 报文，此时服务器进入 LAST_ACK 状态

5、客户端收到服务器的 FIN 后，马上发送 ACK 给服务器，此时客户端进入 TIME_WAIT 状态

6、再过了 2MSL 长的时间后进入 CLOSED 状态。服务器收到客户端的 ACK 就进入 CLOSED 状态。

7. 至此，还有一个状态没有出来：CLOSING 状态。CLOSING 状态表示：客户端发送了 FIN ，但是没有收到服务器的 ACK ，却收到了服务器的 FIN ，这种情况发生在服务器发送的 ACK 丢包的时候。

但这里似乎有一点问题：我捕获到的四次挥手 TCP 包是服务器首先挥手，客户端再挥手，关闭 TCP 连接

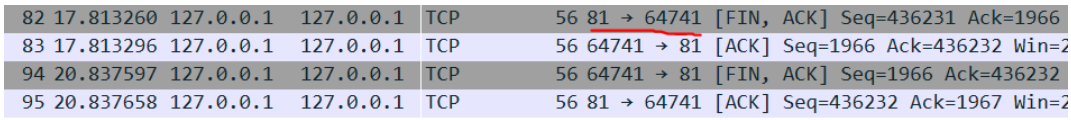


图 4.17: 四次挥手

我猜想应该是，Apache 优化了这方面的算法，当服务器给客户端传输完所有数据后，会直接进行第一次挥手，来表示内容已经发送完毕。

至此，我们就可以成功访问自己编写的 html 网页了。