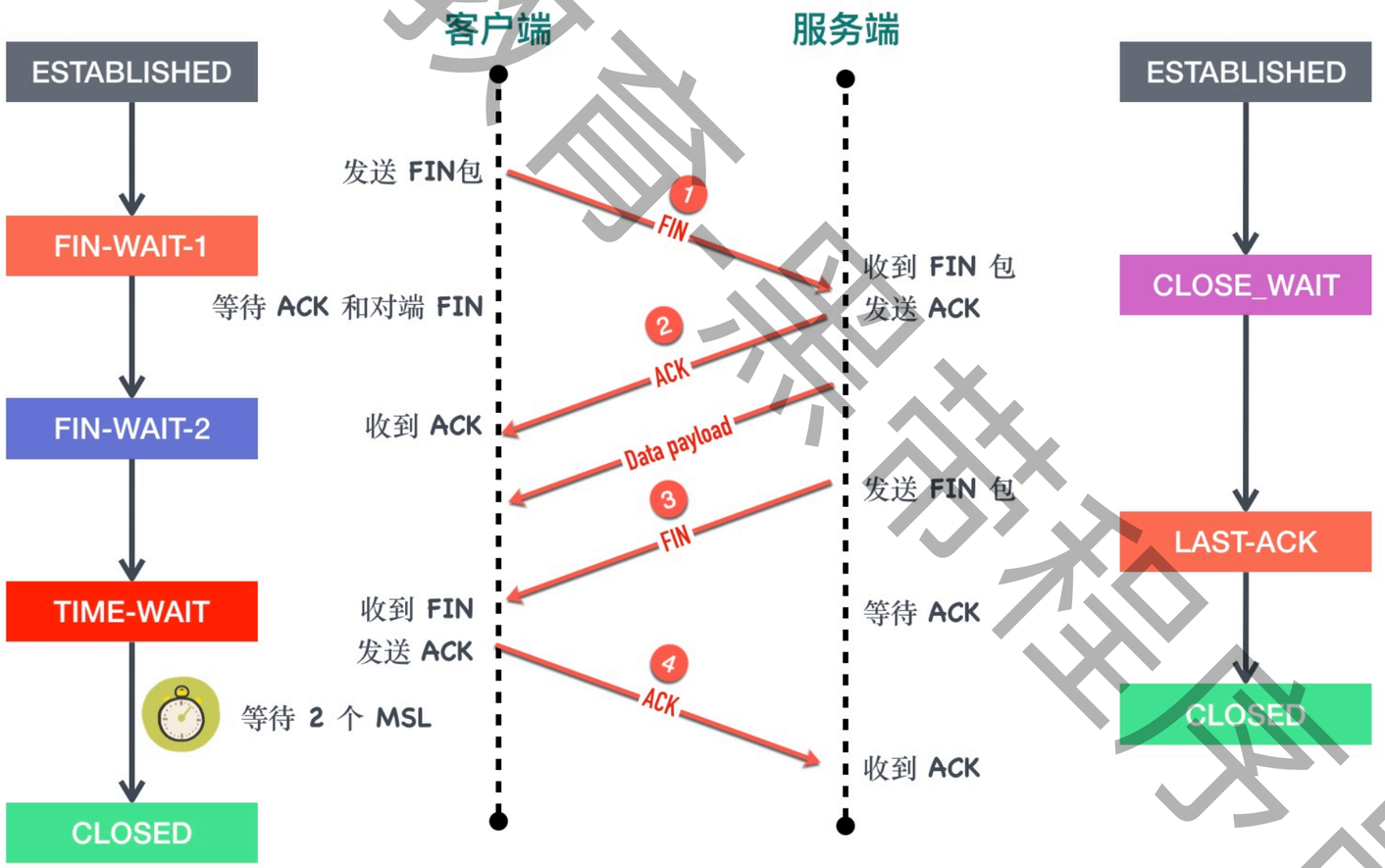
聊聊四次挥手

四次挥手



四次挥手第一步

客户端调用 close 方法，执行「主动关闭」，会发送一个 FIN 报文给服务端，从这以后客户端不能再发送数据给服务端了，客户端进入FIN-WAIT-1状态。FIN 报文其实就是将 FIN 标志位设置为 1。

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td colspan="2">源端口（Source port）</td><td colspan="2">目标端口（Destination port）</td></tr>
<tr><td colspan="4">序列号（Sequence number）</td></tr>
<tr><td colspan="4">确认号（Acknowledgment number）</td></tr>
<tr><td>头部长度</td><td>保留</td><td>NS</td><td>CWR</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td colspan="2">窗口大小（Window Size）</td></tr>
<tr><td colspan="3">校验和（Checksum）</td><td colspan="3">紧急指针（Urgent pointer）</td></tr>
<tr><td colspan="4">选项（Options）、填充（Padding）</td></tr>
</table>

关于 FIN 报文的一些疑问

FIN 报文可以携带数据吗?

FIN 段是可以携带数据的，比如可以在它最后要发送的数据块中“捎带” FIN 段。

FIN 报文如果不携带数据，需要消耗序列号吗?

不管 FIN 是否携带数据，都需要消耗一个序列号

一方发送 FIN 报文以后，还能发送数据或者接收对方的数据吗?

一方发送 FIN 包以后不能再发送数据给对方，但是还可以接受对方端发送的数据。

什么情况会触发 FIN 报文

- 程序调用 close
- 程序异常退出（内核帮忙善后）

程序调用 close

```
int main() {
    int fd = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in server_addr;
    bzero(&server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    server_addr.sin_port = htons(9090);
    int ret = connect(fd, (struct sockaddr *) &server_addr, sizeof(server_addr));
    std::cout << "connect done, ret: " << ret << std::endl;
    std::cin.get();
    std::cout << "closing socket" << std::endl;
    close(fd); // 关闭 socket, 触发发送 FIN
    std::cin.get();
}
```

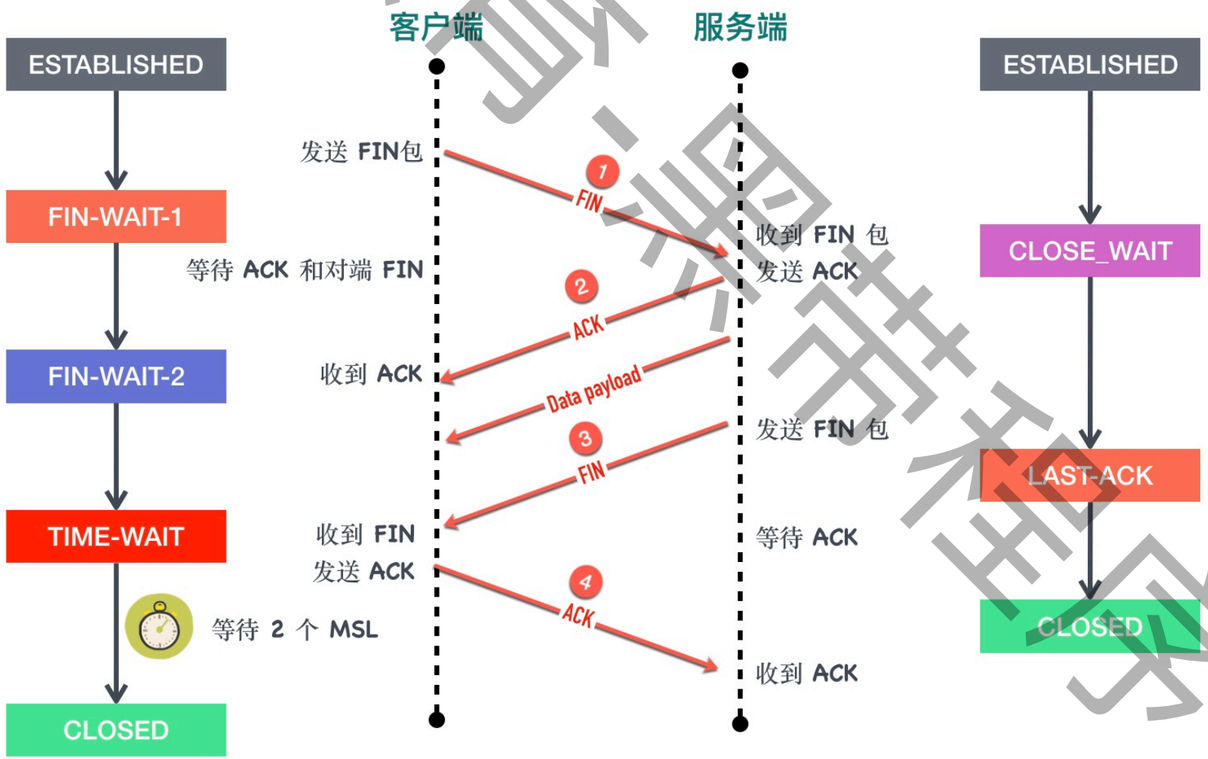
程序异常退出（内核帮忙善后）

kill 一个建立 establish 状态的进程

```
int main() {
    int fd = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in server_addr;
    bzero(&server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    server_addr.sin_port = htons(9090);
    int ret = connect(fd, (struct sockaddr *) &server_addr, sizeof(server_addr));
    std::cout << "connect done, ret: " << ret << std::endl;
    std::cin.get();
}
```

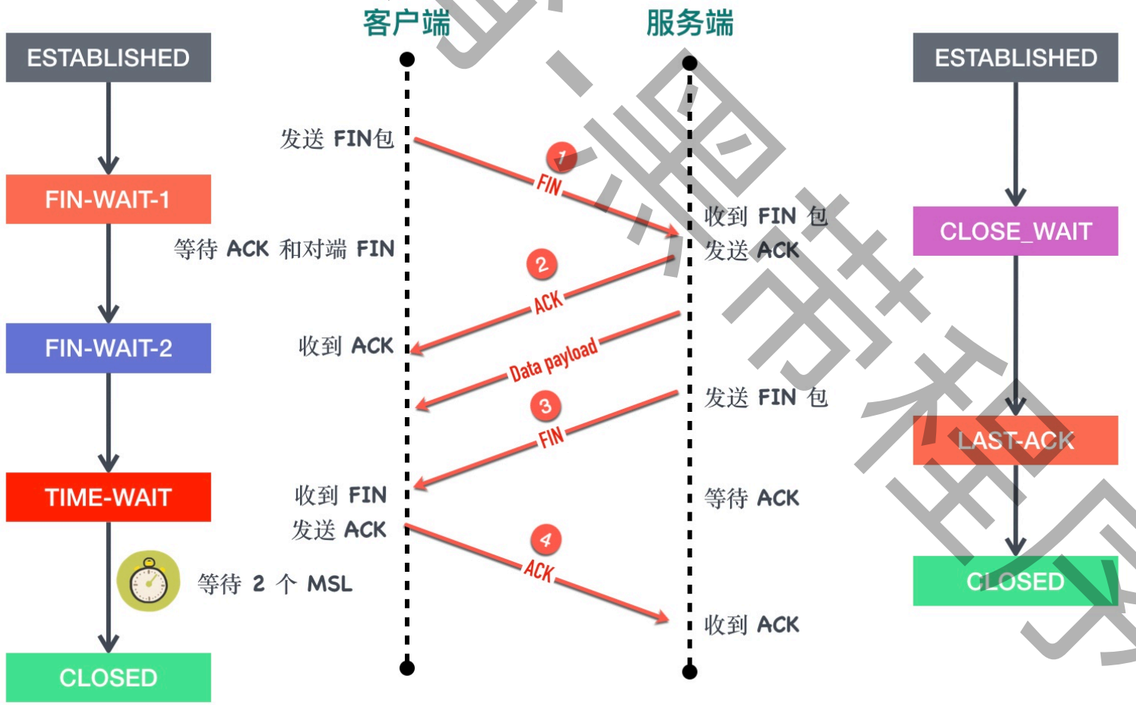
四次挥手第二步

服务端收到 FIN 包以后回复确认 ACK 报文给客户端，服务端进入 CLOSE_WAIT，客户端收到 ACK 以后进入FIN-WAIT-2状态。



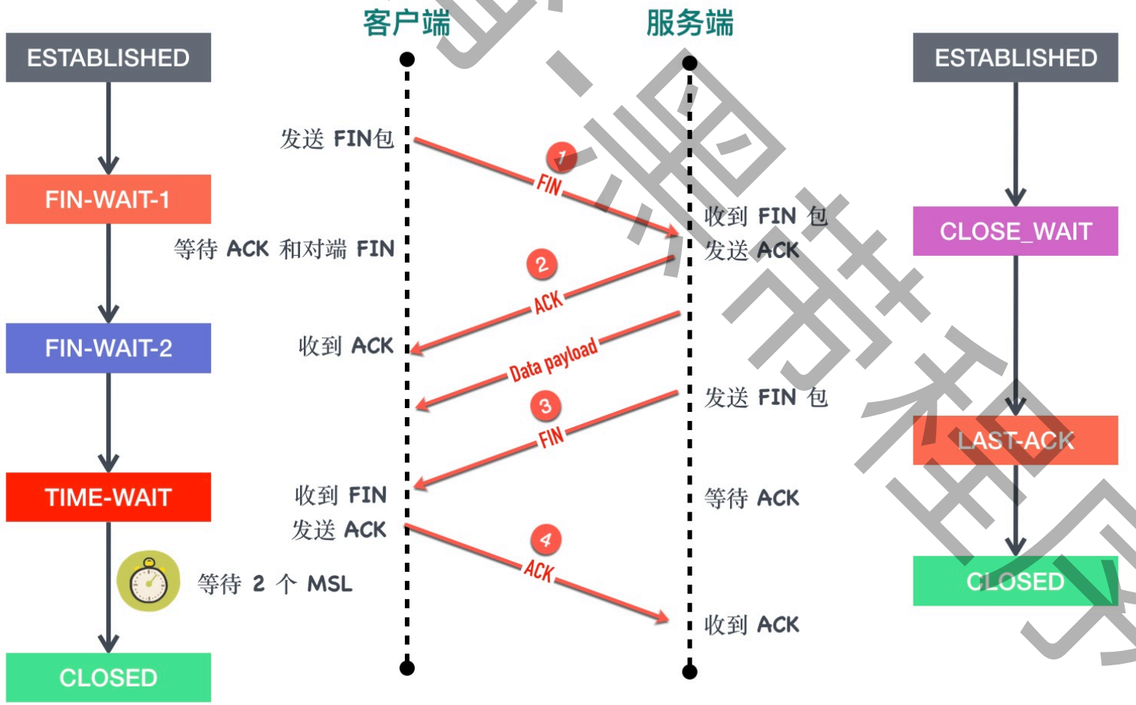
四次挥手第三步

服务端也没有数据要发送了，发送 FIN 报文给客户端，然后进入LAST-ACK 状态，等待客户端的 ACK。同前面一样如果 FIN 段没有携带数据，也需要消耗一个序列号。



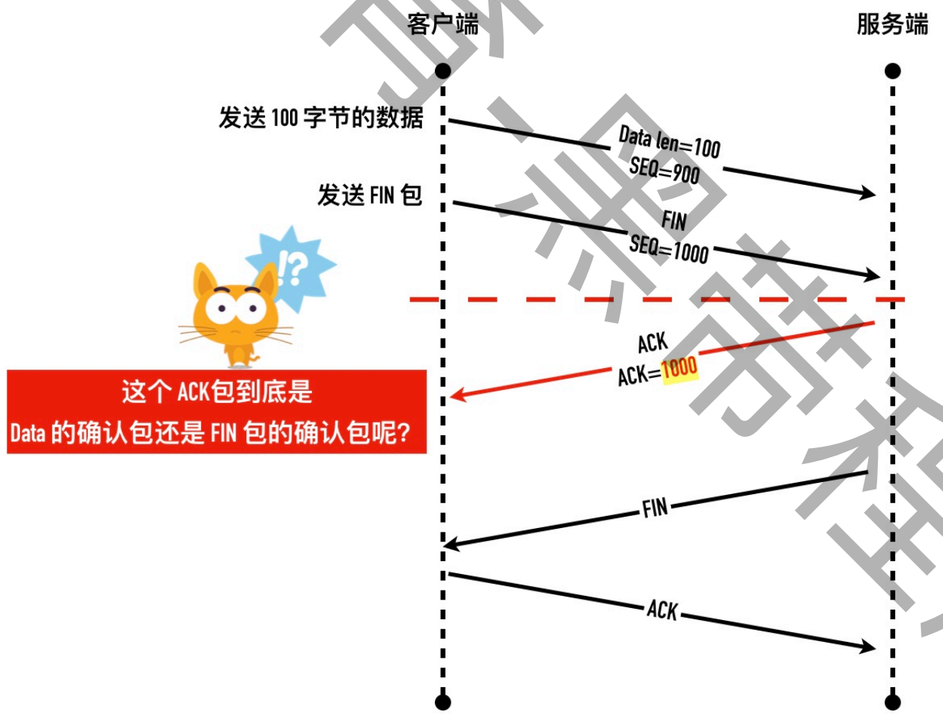
四次挥手第四步

客户端收到服务端的 FIN 报文以后，回复 ACK 报文用来确认第三步里的 FIN 报文，进入TIME_WAIT状态，等待 2 个 MSL 以后进入CLOSED状态。服务端收到 ACK 以后进入CLOSED状态。



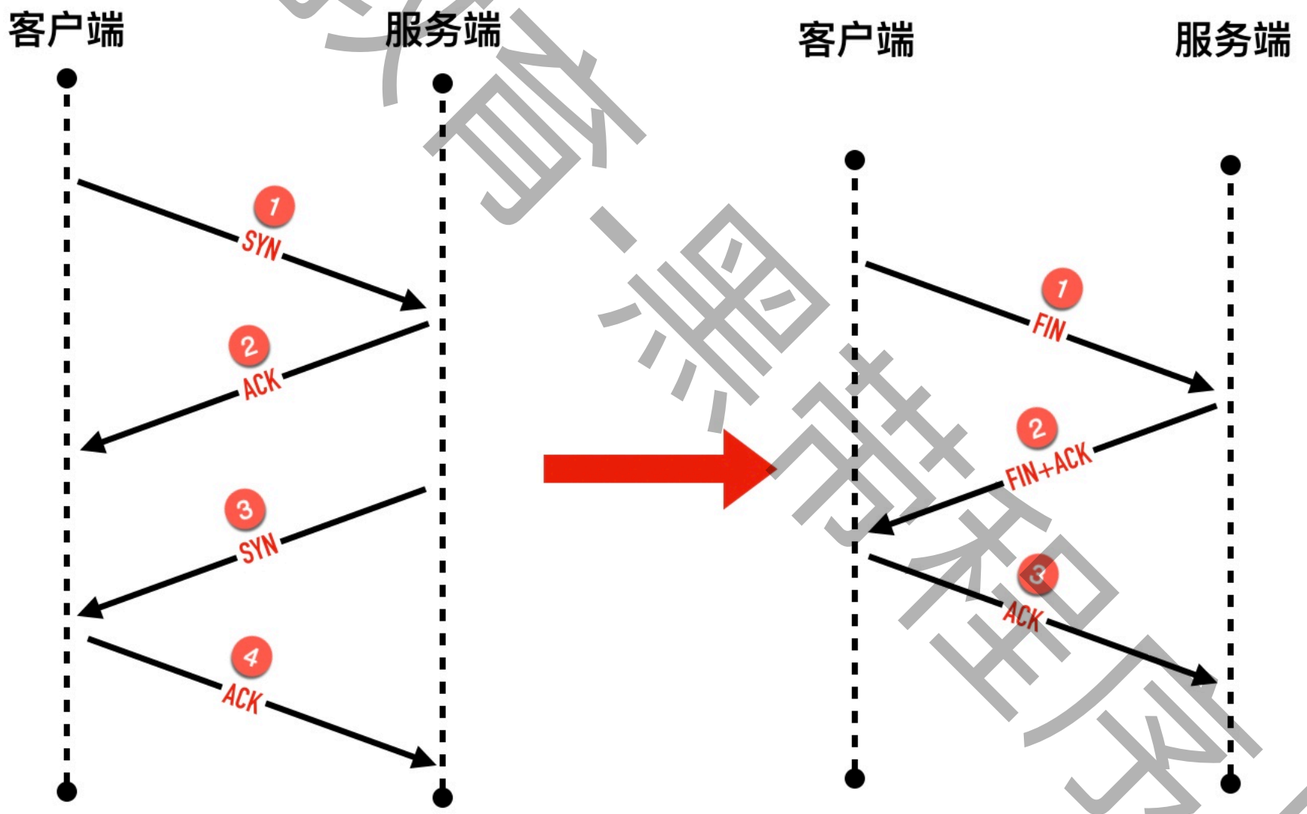
为什么 FIN 报文要消耗一个序列号?

如三次握手的 SYN 报文一样，如果不携带数据，FIN 段也需要消耗一个序列号。



为什么挥手要四次，变为三次可以吗?

可以，因为有**延迟确认**的存在，把第二步的 ACK 经常会跟随第三步的 FIN 包一起捎带会对端。延迟确认后面有一节专门介绍。



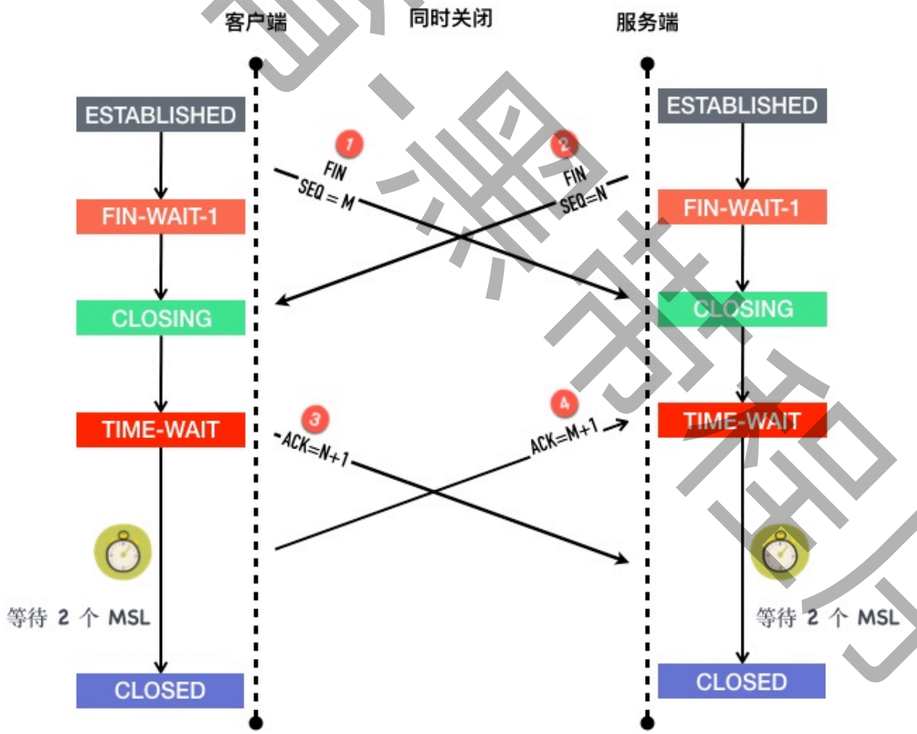
一个真实的 wireshark 抓包如下图所示

Apply a display filter ... <⌘/>

No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.211.55.5	10.211.55.10	TCP	49730 → 9999 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM
2	0.000048	10.211.55.10	10.211.55.5	TCP	9999 → 49730 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=146
3	0.000206	10.211.55.5	10.211.55.10	TCP	49730 → 9999 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=111507
4	4.558550	10.211.55.5	10.211.55.10	TCP	49730 → 9999 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=6 TSval=1
5	4.558702	10.211.55.10	10.211.55.5	TCP	9999 → 49730 [ACK] Seq=1 Ack=7 Win=29056 Len=0 TSval=533961
6	6.468853	10.211.55.5	10.211.55.10	TCP	49730 → 9999 [FIN, ACK] Seq=7 Ack=1 Win=29312 Len=0 TSval=1
7	6.468936	10.211.55.10	10.211.55.5	TCP	9999 → 49730 [FIN, ACK] Seq=1 Ack=8 Win=29056 Len=0 TSval=5
8	6.469088	10.211.55.5	10.211.55.10	TCP	49730 → 9999 [ACK] Seq=8 Ack=2 Win=29312 Len=0 TSval=111508

同时关闭

前面介绍的都是一端收到了对端的 FIN，然后回复 ACK，随后发送自己的 FIN，等待对端的 ACK。TCP 是全双工的，当然可以两端同时发起 FIN 包。如下图所示



```
--tolerance_usecs=10000
0.000 socket(..., SOCK_STREAM, IPPROTO_TCP) = 3
0.000 setsockopt(3, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0
0.000 bind(3, ..., ...) = 0
0.000 listen(3, 1) = 0

// 三次握手
+0 < S 0:0(0) win 65535 <mss 1460>
+0 > S. 0:0(0) ack 1 <...>
+0.1 < . 1:1(0) ack 1 win 65535
+0.010 accept(3, ..., ...) = 4

0.150 close(4) = 0                    // [服务端]关闭socket，触发服务端发送 FIN
0.150 > F. 1:1(0) ack 1 <...>         // 这里只是一个断言，断言服务端会发送 FIN

0.150 < F. 1:1(0) ack 2 win 65535 // [客户端]模拟 FIN 包，发送给服务端
0.150 > .  2:2(0) ack 2 <...>         // 这里只是一个断言，断言服务端会发送 FIN

0.150 < . 2:2(0) ack 2 win 65535 // [客户端]模拟 ACK 包，发送给服务端
```

抓包演示

下面的脚本并不能每次模拟出两端都进入TIME_WAIT的状态，取决于在发送 FIN包之前有没有提前收到对端的 FIN 包。如果在发送 FIN 之前收到了对端的 FIN，只会有一端进入TIME_WAIT

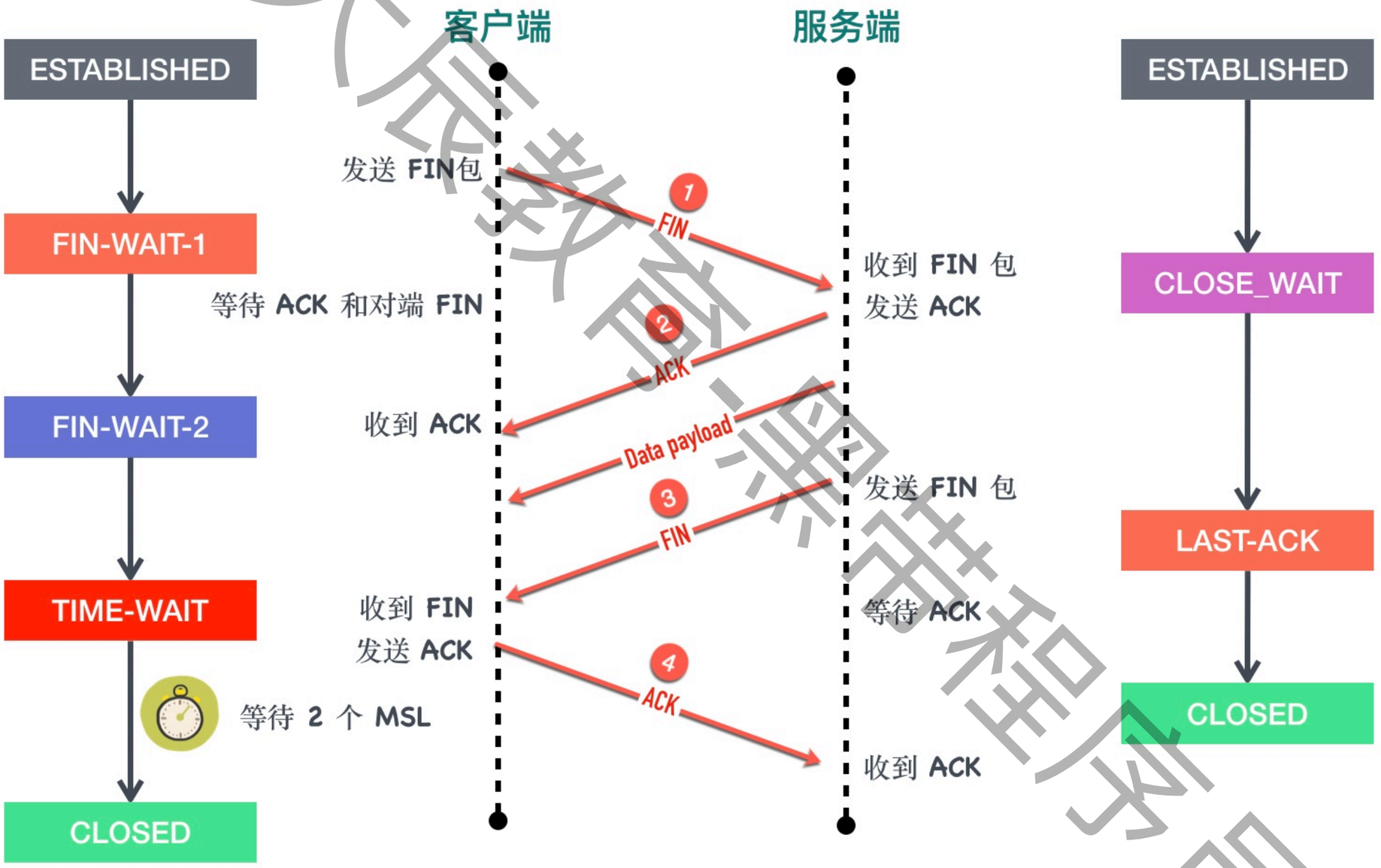
Apply a display filter ... <⌘/>

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.0.2.1	192.168.198.228	TCP	35769 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=1460
2	0.000032	192.168.198.228	192.0.2.1	TCP	8080 → 35769 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
3	0.103171	192.0.2.1	192.168.198.228	TCP	35769 → 8080 [ACK] Seq=1 Ack=1 Win=65535 Len=0
4	0.152619	192.168.198.228	192.0.2.1	TCP	8080 → 35769 [FIN, ACK] Seq=1 Ack=1 Win=29200 Len=0
5	0.152790	192.0.2.1	192.168.198.228	TCP	35769 → 8080 [FIN, ACK] Seq=1 Ack=2 Win=65535 Len=0
6	0.152815	192.168.198.228	192.0.2.1	TCP	8080 → 35769 [ACK] Seq=2 Ack=2 Win=29200 Len=0
7	0.152904	192.0.2.1	192.168.198.228	TCP	[TCP Dup ACK 5#1] 35769 → 8080 [ACK] Seq=2 Ack=2 Win=65535 Len=0

面试题讲解

HTTP传输完成，断开进行四次挥手，第二次挥手的时候客户端所处的状态是：

- A、CLOSE_WAIT
- B、LAST_ACK
- C、FIN_WAIT
- D、TIME_WAIT


客户端
服务端
ESTABLISHED
FIN-WAIT-1
FIN-WAIT-2
TIME-WAIT
CLOSED
发送 FIN包
1
FIN
收到 FIN 包
发送 ACK
等待 ACK 和对端 FIN
2
ACK
收到 ACK
Data payload
发送 FIN 包
3
FIN
收到 FIN
发送 ACK
等待 ACK
等待 2 个 MSL
4
ACK
收到 ACK
ESTABLISHED
CLOSE_WAIT
LAST-ACK
CLOSED

TCP三次握手和四次挥手过程中，以下状态分别处于服务端和客户端描述正确的是：

- A、服务端：SYN-SENT，TIME-WAIT 客户端：SYN-RCVD，CLOSE-WAIT
- B、服务端：SYN-SENT，CLOSE-WAIT 客户端：SYN-RCVD，TIME-WAIT
- C、服务端：SYN-RCVD，CLOSE-WAIT 客户端：SYN-SENT，TIME-WAIT
- D、服务端：SYN-RCVD，TIME-WAIT 客户端：SYN-SENT，CLOSE-WAIT