为什么我的 TCP 挥手只有 3 次

聊聊我遇到过的一些网络问题

自我介绍

运维组 王泓智

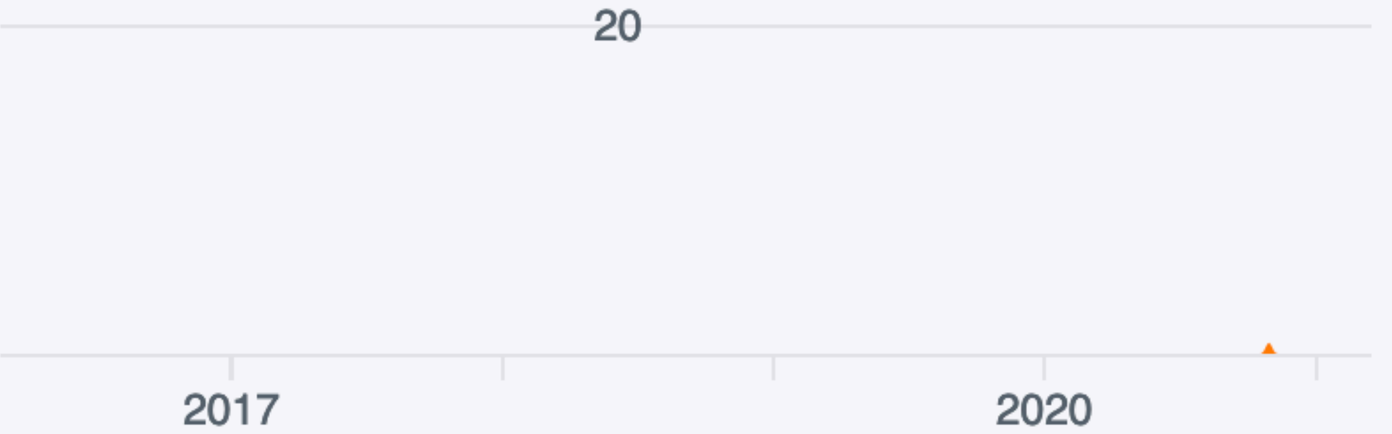
- 终身学习者一名，最近在看 Google SRE 工作手册
- 一直关注云原生技术栈，参与过相关文档的翻译，往 kubernetes 源码提过一个 PR
- ctripcorp/apollo 第 69 位 “contributor”

https://github.com/ctripcorp/apollo/graphs/contributors

WisWang

#69

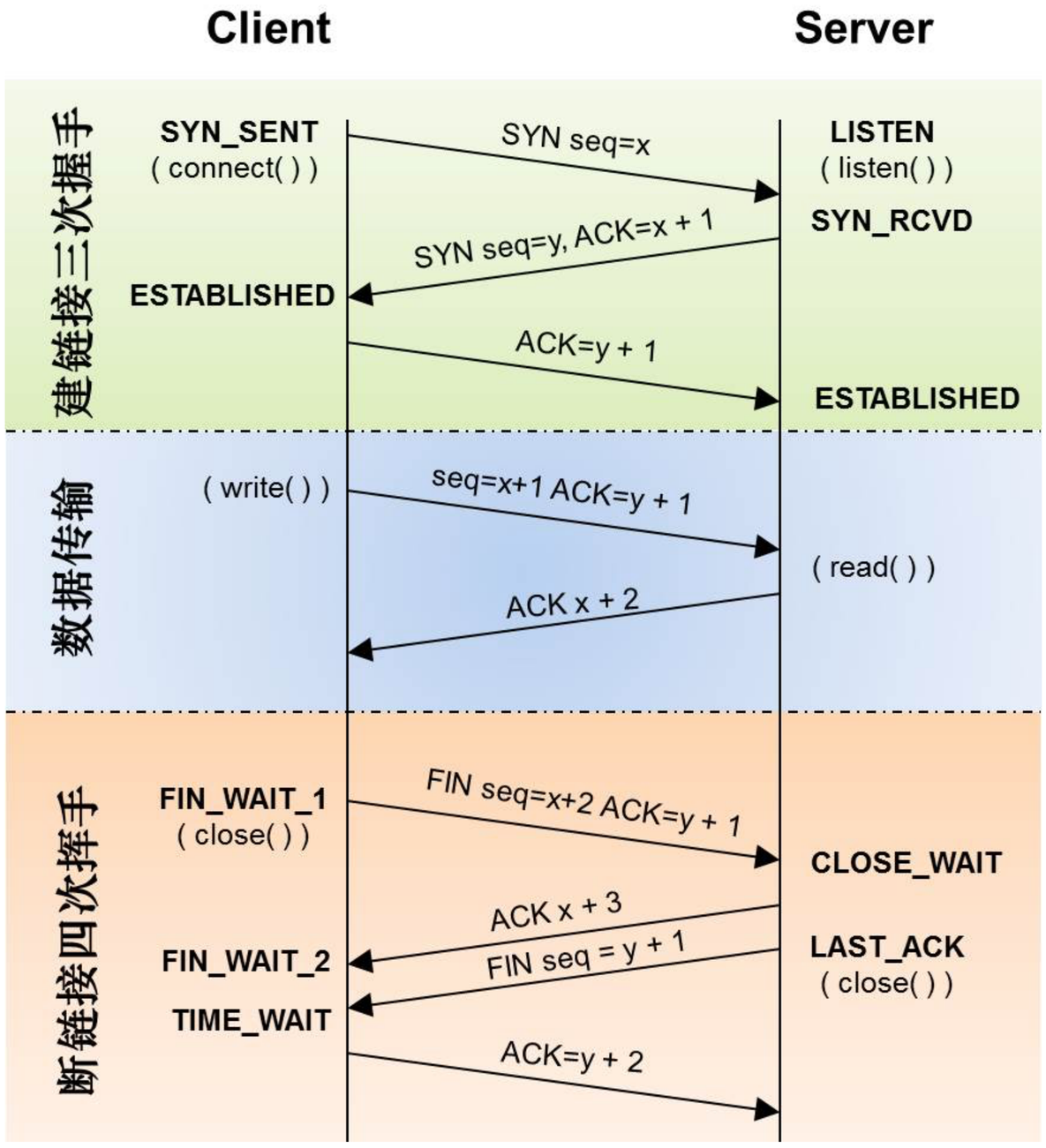
1 commit 2 ++ 2 --



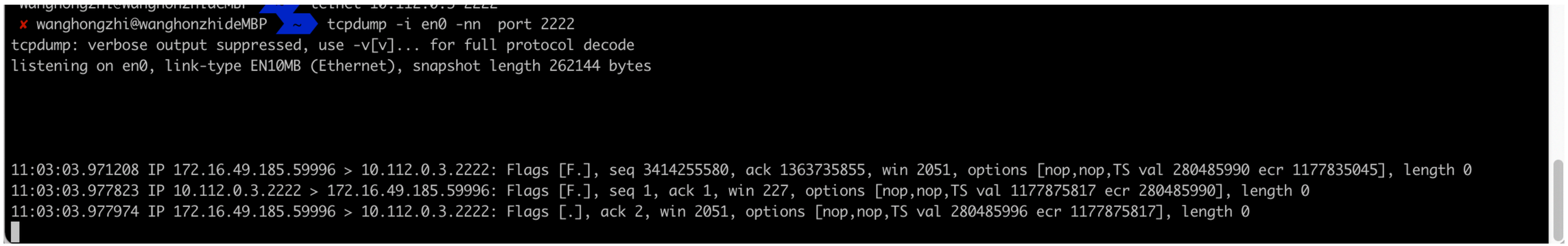
目录

我遇到的一些网络问题

- 为什么我的 TCP 挥手只有 3 次
- ESTABLISHED 的连接只在一端有，另一端却没有
- CLOSE_WAIT 状态连接过多怎么办


Client
Server
建链接三次握手
SYN_SENT
(connect( ))
SYN seq=x
LISTEN
(listen( ))
SYN_RCVD
SYN seq=y, ACK=x + 1
ESTABLISHED
ACK=y + 1
ESTABLISHED
数据传输
(write( ))
seq=x+1 ACK=y + 1
(read( ))
ACK x + 2
断链接四次挥手
FIN_WAIT_1
(close( ))
FIN seq=x+2 ACK=y + 1
CLOSE_WAIT
ACK x + 3
FIN_WAIT_2
FIN seq = y + 1
LAST_ACK
(close( ))
TIME_WAIT
ACK=y + 2

为什么我的 TCP 挥手只有 3 次

- 

ESTABLISHED 的连接只在一端有，另一端却没有

```
tcp        0    108 10.89.150.15:22         10.89.90.52:51686       ESTABLISHED -
tcp        0      0 10.89.150.15:2379       10.89.150.15:55774      ESTABLISHED -
tcp        0      0 10.89.150.15:2380       10.89.150.14:47028      ESTABLISHED -
tcp        0      0 10.89.150.15:55666      10.89.150.15:2379       ESTABLISHED -
tcp        0      0 10.89.150.15:55732      10.89.150.15:2379       ESTABLISHED -
tcp        0      0 10.89.150.15:55750      10.89.150.15:2379       ESTABLISHED -
tcp        0      0 10.89.150.15:55792      10.89.150.15:2379       ESTABLISHED -
tcp        0      0 10.89.150.15:55766      10.89.150.15:2379       ESTABLISHED -
tcp        0      0 10.89.150.15:2379       10.89.150.15:55766      ESTABLISHED -
tcp        0      0 10.89.150.15:55660      10.89.150.15:2379       ESTABLISHED -
tcp        0      0 10.89.150.15:2379       10.89.150.15:55730      ESTABLISHED -
tcp        0      0 10.89.150.15:2379       10.89.150.15:55714      ESTABLISHED -
tcp        0      0 10.89.150.15:55748      10.89.150.15:2379       ESTABLISHED -
tcp        0      0 10.89.150.15:55768      10.89.150.15:2379       ESTABLISHED -
```

CLOSE_WAIT 状态连接过多怎么办

-
```
[root@VM-0-3-centos ~]# netstat -anpt|grep -i close_wait
tcp      738      0 10.112.0.3:8080         172.17.0.2:49746        CLOSE_WAIT  27983/python3.6
tcp     1493      0 127.0.0.1:8080          127.0.0.1:46464         CLOSE_WAIT  27983/python3.6
tcp        1      0 127.0.0.1:8080          127.0.0.1:46212         CLOSE_WAIT  1189/python3.6
tcp      294      0 10.112.0.3:8080         172.17.0.2:49472        CLOSE_WAIT  27983/python3.6
tcp        1      0 127.0.0.1:8080          127.0.0.1:45908         CLOSE_WAIT  1189/python3.6
tcp      738      0 10.112.0.3:8080         172.17.0.2:49642        CLOSE_WAIT  27983/python3.6
tcp     1629      0 127.0.0.1:8080          127.0.0.1:46350         CLOSE_WAIT  27983/python3.6
tcp     1670      0 127.0.0.1:8080          127.0.0.1:46696         CLOSE_WAIT  27983/python3.6
tcp      294      0 10.112.0.3:8080         172.17.0.2:49702        CLOSE_WAIT  27983/python3.6
tcp      738      0 10.112.0.3:8080         172.17.0.2:49414        CLOSE_WAIT  27983/python3.6
tcp        1      0 127.0.0.1:8080          127.0.0.1:45922         CLOSE_WAIT  1189/python3.6
```

TCP 知识扩展

- TCP 为每个连接建立 7 个定时器
- nagle’s 算法
- TCP 流量控制和拥塞控制
- QUIC 简介

TCP 为每个连接建立 7 个定时器

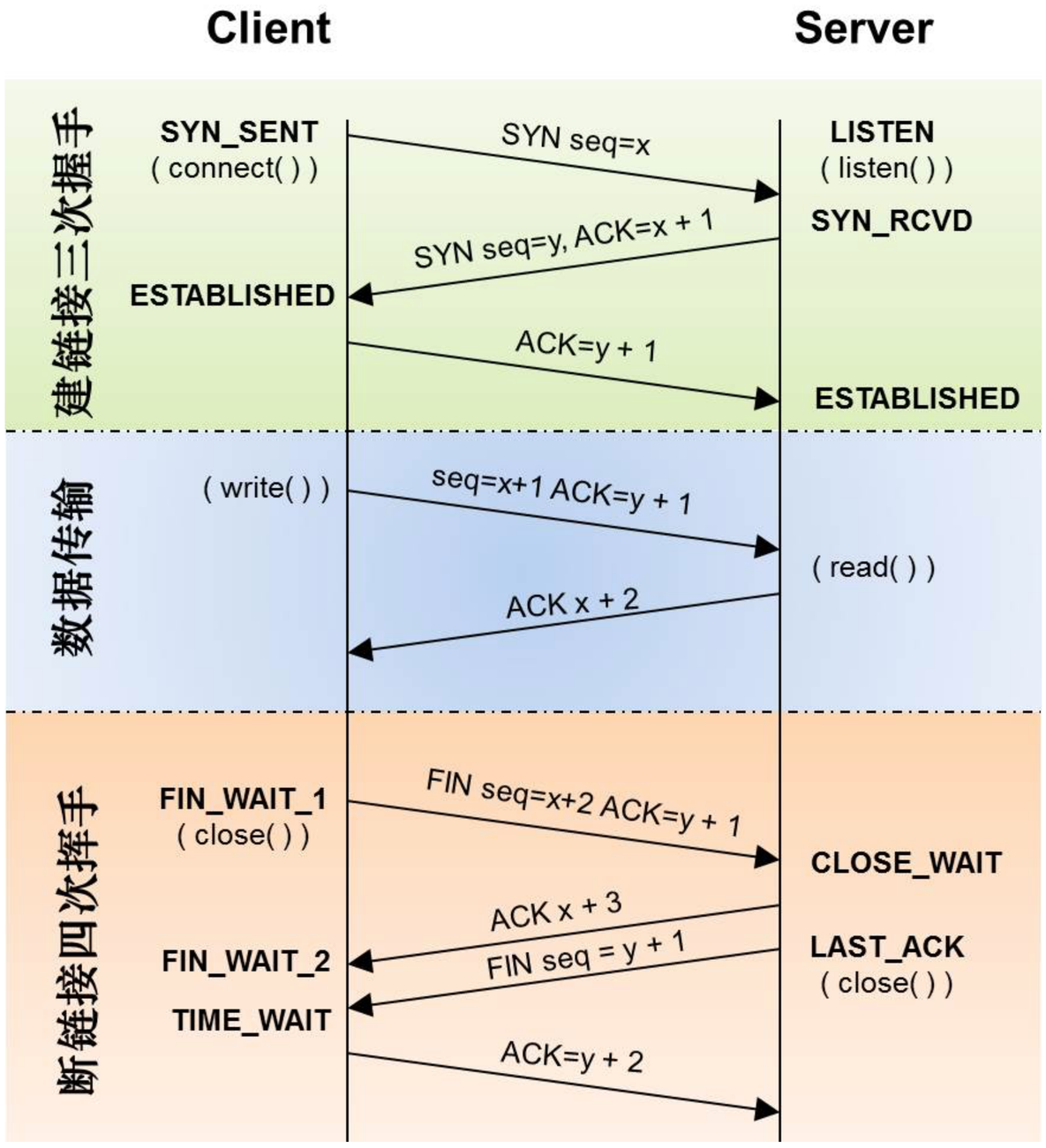
TCPIP 卷二 25 章中提到 TCP 为每个连接建立 7 个定时器

- 连接建立(connection establishment) 定时器在发送 SYN 报文段建立一条新连接时启动。如果没有在 75 秒内收到响应，连接建立将中止。
- 重传(retransmission)定时器在 TCP 发送数据时设定。
- 延迟 ACK(delayed ACK) 定时器在 TCP 收到必须被确认但无需马上发出确认的数据时设定。也称为捎带确认。

TCP 为每个连接建立 7 个定时器

TCPIP 卷二 25 章中提到 TCP 为每个连接建立 7 个定时器

- 持续(persist)定时器在连接对端通告接收窗口为 0，阻止 TCP 继续发送数据时设定。
- 保活(keepalive)定时器在应用进程选取了插口的 SO_KEEPALIVE 选项时生效。
- FIN_WAIT_2 定时器。
- TIME_WAIT 定时器，一般也称为 2MSL 定时器。


Client
Server
建链接三次握手
SYN_SENT
(connect( ))
SYN seq=x
LISTEN
(listen( ))
SYN_RCVD
SYN seq=y, ACK=x + 1
ESTABLISHED
ACK=y + 1
ESTABLISHED
数据传输
(write( ))
seq=x+1 ACK=y + 1
(read( ))
ACK x + 2
断链接四次挥手
FIN_WAIT_1
(close( ))
FIN seq=x+2 ACK=y + 1
CLOSE_WAIT
ACK x + 3
FIN_WAIT_2
FIN seq = y + 1
LAST_ACK
(close( ))
TIME_WAIT
ACK=y + 2

nagle's 算法

```
if there is new data to send
  if the window size >= MSS and available data is >= MSS
    send complete MSS segment now
  else
    if there is unconfirmed data still in the pipe
      enqueue data in the buffer until an acknowledge is received
    else
      send data immediately
    end if
  end if
end if
```

TCP 流量控制和拥塞控制

- 滑动窗口。接收方把它的处理能力告诉发送方，使其限制发送速度即可，这就是滑动窗口的由来。
- swnd = min(cwnd, rwnd) ， 这个公式是拥塞窗口和滑动窗口共同控制发送的速度。
- 拥塞控制（慢启动、拥塞避免、快速重传、快速恢复）。

QUIC 简介

- 应用程序层面就能实现不同的拥塞控制算法。
- 集成了 TLS 1.3 加密
- 连接迁移（ QUIC 让客户端生成一个 Connection ID （64 位）来区别不同连接）

参考资料

- nagle’s 算法（https://en.wikipedia.org/wiki/Nagle%27s_algorithm）
- 云网络丢包故障定位，看这一篇就够了（https://mp.weixin.qq.com/s/-Q1AkxUr9xzGKwUMV-FQhQ）
- 那些你不知道的 TCP 冷门知识！（https://mp.weixin.qq.com/s/6lop61UtnQ-vfWJy17V87w）

参考资料

- 极客时间：趣谈网络协议，系统性能调优必知必会，Linux 性能优化实战
- TCP/IP 卷一，卷二
- 腾讯技术工程精华文章电子书 2020（https://tm.huohua.cn/2890222485043855 38/articles/3032543800013946 89）