# Nagle 算法那些事

## TCP 发包的 hold 住哥


# TCP发包的Hold住哥

## Nagle 算法

```
Socket socket = new Socket();
socket.connect(new InetSocketAddress("localhost", 9999));
OutputStream output = socket.getOutputStream();
byte[] request = new byte[10];
for (int i = 0; i < 5; i++) {
    output.write(request);
}
```

说法正确的是：

- A. TCP 把 5 个包合并，一次发送 50 个字节
- B. TCP 分 5 次发送，一次发送 10 个字节
- C. 以上都不对

# 抓包结果

| No. | Time | Source | Destination | Info |
|---|---|---|---|---|
| 1 | 08:36:04.887321 | 10.211.55.10 | 10.211.55.5 | 48406 → 9999 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1880814 TSecr=0 W |
| 2 | 08:36:04.887389 | 10.211.55.5 | 10.211.55.10 | 9999 → 48406 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=188088 |
| 3 | 08:36:04.887439 | 10.211.55.10 | 10.211.55.5 | 48406 → 9999 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1880814 TSecr=1880889 |
| 4 | 08:36:04.887804 | 10.211.55.10 | 10.211.55.5 | 48406 → 9999 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=10 TSval=1880814 TSecr=1880889 |
| 5 | 08:36:04.888012 | 10.211.55.5 | 10.211.55.10 | 9999 → 48406 [ACK] Seq=1 Ack=11 Win=29056 Len=0 TSval=1880890 TSecr=1880814 |
| 6 | 08:36:04.888135 | 10.211.55.10 | 10.211.55.5 | 48406 → 9999 [PSH, ACK] Seq=11 Ack=1 Win=29312 Len=40 TSval=1880815 TSecr=1880890 |
| 7 | 08:36:04.888333 | 10.211.55.5 | 10.211.55.10 | 9999 → 48406 [ACK] Seq=1 Ack=51 Win=29056 Len=0 TSval=1880890 TSecr=1880815 |
| 8 | 08:36:05.889022 | 10.211.55.10 | 10.211.55.5 | 48406 → 9999 [FIN, ACK] Seq=51 Ack=1 Win=29312 Len=0 TSval=1881816 TSecr=1880890 |
| 9 | 08:36:05.889183 | 10.211.55.5 | 10.211.55.10 | 9999 → 48406 [FIN, ACK] Seq=1 Ack=52 Win=29056 Len=0 TSval=1881891 TSecr=1881816 |
| 10 | 08:36:05.889225 | 10.211.55.10 | 10.211.55.5 | 48406 → 9999 [ACK] Seq=52 Ack=2 Win=29312 Len=0 TSval=1881816 TSecr=1881891 |

可以看到除了第一个包是单独发送，后面的四个包合并到了一起，所以答案是 C


# Nagle 算法

nagle 算法讲的是减少发送端频繁的发送小包给对方。

Nagle 算法要求：

当一个 TCP 连接中有在传数据（已经发出但还未确认的数据）时，小于 MSS 的报文段就不能被发送，直到所有的在传数据都收到了 ACK。

同时收到 ACK 后，TCP 还不会马上就发送数据，会收集小包合并一起发送

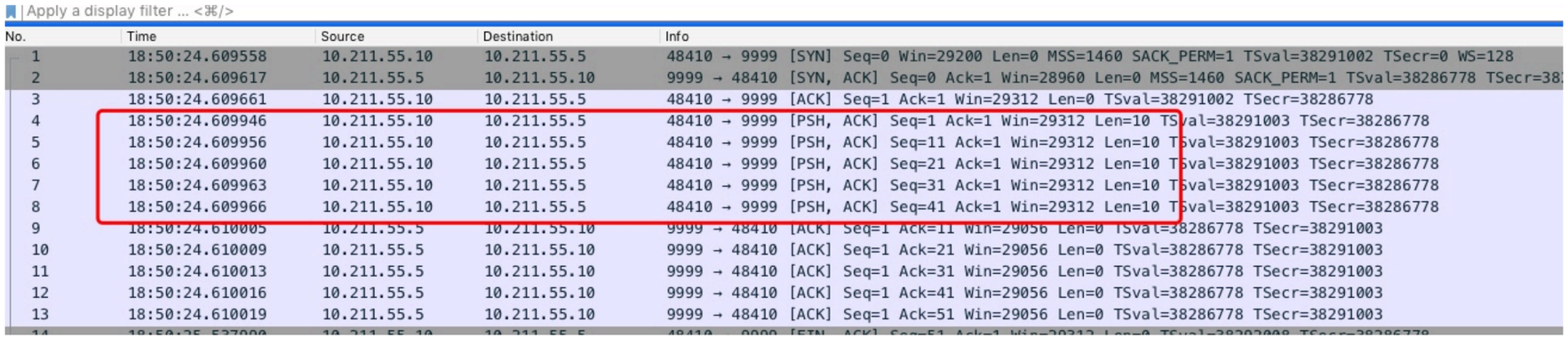
```
if there is new data to send
  if the window size >= MSS and available data is >= MSS
    send complete MSS segment now
  else
    if there is unconfirmed data still in the pipe
      enqueue data in the buffer until an acknowledge is received
    else
      send data immediately
    end if
  end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

```
socket.setTcpNoDelay(true);
```


# 再次实验

## Nagle 算法

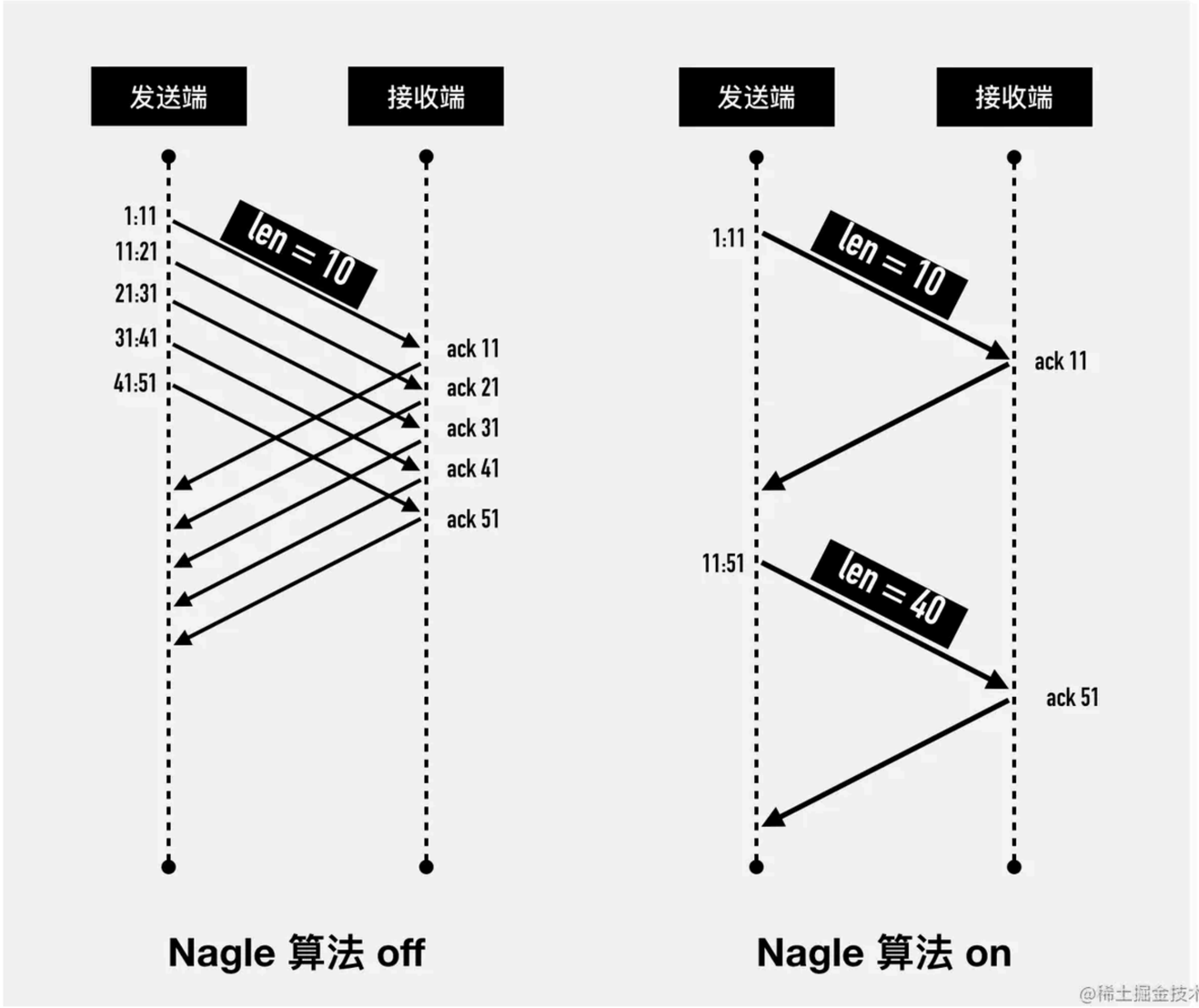
```
Socket socket = new Socket();
socket.setTcpNoDelay(true);
socket.connect(new InetSocketAddress("localhost", 9999));
OutputStream output = socket.getOutputStream();
byte[] request = new byte[10];
for (int i = 0; i < 5; i++) {
    output.write(request);
}
```



Apply a display filter ... <⌘/>

| No. | Time | Source | Destination | Info |
|---|---|---|---|---|
| 1 | 18:50:24.609558 | 10.211.55.10 | 10.211.55.5 | 48410 → 9999 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=38291002 TSecr=0 WS=128 |
| 2 | 18:50:24.609617 | 10.211.55.5 | 10.211.55.10 | 9999 → 48410 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=38286778 TSecr=38 |
| 3 | 18:50:24.609661 | 10.211.55.10 | 10.211.55.5 | 48410 → 9999 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=38291002 TSecr=38286778 |
| 4 | 18:50:24.609946 | 10.211.55.10 | 10.211.55.5 | 48410 → 9999 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=10 TSval=38291003 TSecr=38286778 |
| 5 | 18:50:24.609956 | 10.211.55.10 | 10.211.55.5 | 48410 → 9999 [PSH, ACK] Seq=11 Ack=1 Win=29312 Len=10 TSval=38291003 TSecr=38286778 |
| 6 | 18:50:24.609960 | 10.211.55.10 | 10.211.55.5 | 48410 → 9999 [PSH, ACK] Seq=21 Ack=1 Win=29312 Len=10 TSval=38291003 TSecr=38286778 |
| 7 | 18:50:24.609963 | 10.211.55.10 | 10.211.55.5 | 48410 → 9999 [PSH, ACK] Seq=31 Ack=1 Win=29312 Len=10 TSval=38291003 TSecr=38286778 |
| 8 | 18:50:24.609966 | 10.211.55.10 | 10.211.55.5 | 48410 → 9999 [PSH, ACK] Seq=41 Ack=1 Win=29312 Len=10 TSval=38291003 TSecr=38286778 |
| 9 | 18:50:24.610005 | 10.211.55.5 | 10.211.55.10 | 9999 → 48410 [ACK] Seq=1 Ack=11 Win=29056 Len=0 TSval=38286778 TSecr=38291003 |
| 10 | 18:50:24.610009 | 10.211.55.5 | 10.211.55.10 | 9999 → 48410 [ACK] Seq=1 Ack=21 Win=29056 Len=0 TSval=38286778 TSecr=38291003 |
| 11 | 18:50:24.610013 | 10.211.55.5 | 10.211.55.10 | 9999 → 48410 [ACK] Seq=1 Ack=31 Win=29056 Len=0 TSval=38286778 TSecr=38291003 |
| 12 | 18:50:24.610016 | 10.211.55.5 | 10.211.55.10 | 9999 → 48410 [ACK] Seq=1 Ack=41 Win=29056 Len=0 TSval=38286778 TSecr=38291003 |
| 13 | 18:50:24.610019 | 10.211.55.5 | 10.211.55.10 | 9999 → 48410 [ACK] Seq=1 Ack=51 Win=29056 Len=0 TSval=38286778 TSecr=38291003 |

# 前后对比



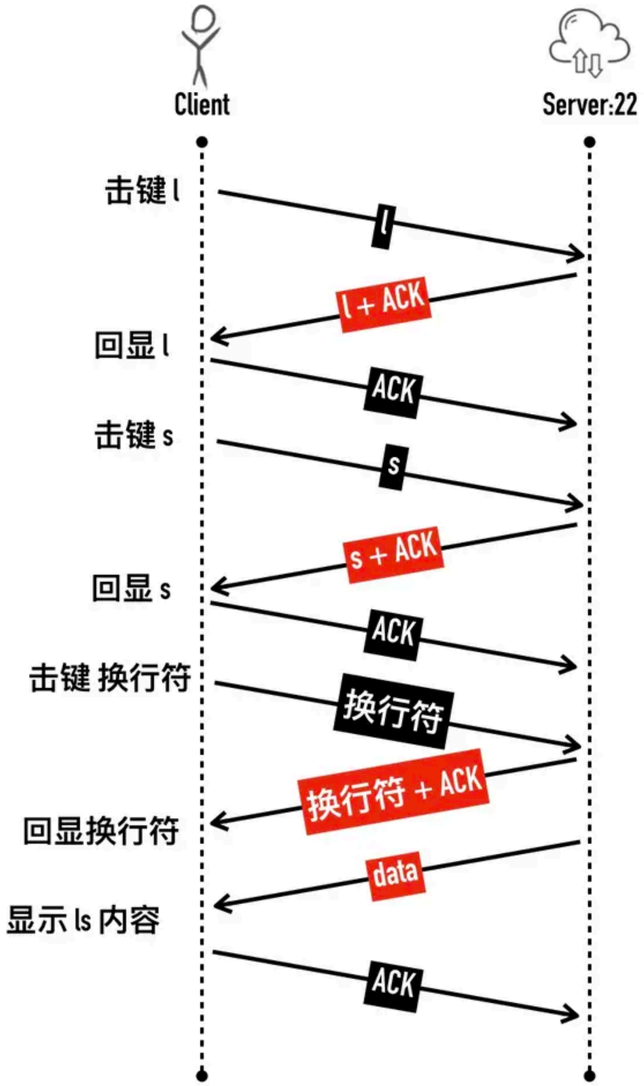
# packetdrill 演示

# packetdrill 脚本

```
--tolerance_usecs=100000
0.000 socket(..., SOCK_STREAM, IPPROTO_TCP) = 3
// 0.010 setsockopt(3, SOL_TCP, TCP_NODELAY, [1], 4) = 0

0.100...0.200 connect(3, ..., ...) = 0

// Establish a connection.
0.100 > S 0:0(0) <mss 1460,sackOK,TS val 100 ecr 0,nop,wscale 7>
0.200 < S. 0:0(0) ack 1 win 32792 <mss 1100,nop,wscale 7>
0.200 > . 1:1(0) ack 1

+0 write(3, ..., 10) = 10
+0 write(3, ..., 10) = 10
+0 write(3, ..., 10) = 10
+0 write(3, ..., 10) = 10
+0 write(3, ..., 10) = 10

+0.030 < . 1:1(0) ack 11 win 257
+0.030 < . 1:1(0) ack 21 win 257
+0.030 < . 1:1(0) ack 31 win 257
+0.030 < . 1:1(0) ack 41 win 257
+0.030 < . 1:1(0) ack 51 win 257

+0 `sleep 1000000`
```

# 典型的小包场景

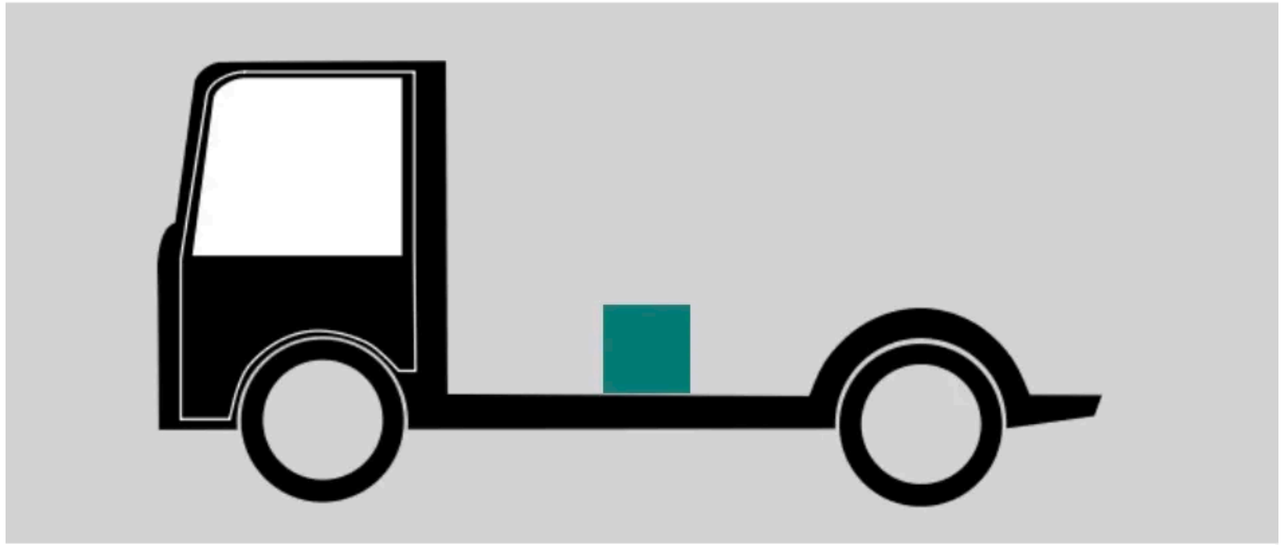
每输入一个字符，服务端也随即进行回应，客户端收到了以后才会把输入的字符和响应的内容显示在自己这边




1. 客户端输入`l`，字符 `l` 被加密后传输给服务器
2. 服务器收到`l`包，回复被加密的 `l` 及 ACK
3. 客户端输入`s`，字符 `s` 被加密后传输给服务器
4. 服务器收到`s`包，回复被加密的 `s` 及 ACK
5. 客户端输入 enter 换行符，换行符被加密后传输给服务器
6. 服务器收到换行符，回复被加密的换行符及 ACK
7. 服务端返回执行 ls 的结果
8. 客户端回复 ACK

# Nagle 算法的意义

Nagle 算法的作用是减少小包在客户端和服务端直接传输，一个包的 TCP 头和 IP 头加起来至少都有 40 个字节，如果携带的数据比较小的话，那就非常浪费了。就好比开着一辆大货车运一箱苹果一样。



# Nagle 算法是时代的产物

Nagle 算法在通信时延较低的场景下意义不大。在 Nagle 算法中 ACK 返回越快，下次数据传输就越早。

假设 RTT 为 10ms 且没有延迟确认，那么你敲击键盘的间隔大于 10ms 的话就不会触发 Nagle 的条件。如果你想触发 Nagle 的停等（stop-wait）机制，1s 内要输入超过 100 个字符。

因此如果在局域网内，Nagle 算法基本上没有什么效果。

如果客户端到服务器的 RTT 较大，比如多达 200ms，这个时候你只要1s 内输入超过 5 个字符，就有可能触发 Nagle 算法了。

Nagle 算法出现的时候网络带宽都很小，当有大量小包传输时，很容易将带宽占满，出现丢包重传等现象。因此对 ssh 这种交互式的应用场景，选择开启 Nagle 算法可以使得不再那么频繁的发送小包，而是合并到一起，代价是稍微有一些延迟。现在的 ssh 客户端已经默认关闭了 Nagle 算法。

# Nagle 小结

Nagle 算法是应用在发送端的，简而言之就是，对发送端而言：

- 当第一次发送数据时不用等待，就算是 1byte 的小包也立即发送
- 后面发送数据时需要累积数据包直到满足下面的条件之一才会继续发送数据：
    1. 数据包达到最大段大小MSS
    2. 接收端收到之前数据包的确认 ACK