40. 案例篇：網路請求延遲變大了，我該怎麼辦？

10/8　Freddy Fan

除了DDOS外，其他網路延遲的原因

1. 網路傳輸慢
2. Linux 内核协议栈报文处理慢 <=俗稱電腦很忙
3. 应用程序数据处理慢 <=跑資料跑太久

hping3

-c 指定資料包的次數

-S syn 使用SNY標記

```
[root@localhost ~]# hping3 -c 3 -S -p 80 baidu.com
HPING baidu.com (enp0s3 220.181.38.148): S set, 40 headers + 0 data bytes
len=46 ip=220.181.38.148 ttl=45 id=35216 sport=80 flags=SA seq=0 win=8192 rtt=67.1 ms
len=46 ip=220.181.38.148 ttl=45 id=58584 sport=80 flags=SA seq=1 win=8192 rtt=65.8 ms
len=46 ip=220.181.38.148 ttl=45 id=42170 sport=80 flags=SA seq=2 win=8192 rtt=80.9 ms

--- baidu.com hping statistic ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 65.8/71.3/80.9 ms
```

RTT : **Round Trip Time**（來回通訊延遲）

```
[root@localhost ~]# traceroute --tcp -p 80 -n baidu.com
traceroute to baidu.com (39.156.69.79), 30 hops max, 60 byte packets
 1  192.168.0.1  3.016 ms  2.918 ms  2.890 ms
 2  10.48.192.1  14.518 ms  15.071 ms  14.843 ms
 3  10.4.18.113  14.460 ms  15.291 ms  15.203 ms
 4  211.76.114.254  16.252 ms  16.185 ms  16.008 ms
 5  219.80.240.185  15.125 ms 219.80.241.237  14.704 ms 219.80.240.185  14.901 ms
 6  60.199.4.217  14.866 ms 60.199.4.225  15.558 ms  21.235 ms
 7  60.199.3.122  39.972 ms 60.199.3.130  13.446 ms 60.199.3.122  27.661 ms
 8  60.199.14.241  12.723 ms 60.199.14.245  12.057 ms 60.199.14.241  12.433 ms
 9  175.41.60.169  140.970 ms  146.942 ms  146.787 ms
10  175.41.60.222  143.975 ms 175.41.60.18  167.445 ms 175.41.60.222  142.639 ms
11  175.41.60.58  174.997 ms  173.807 ms 175.41.60.90  164.445 ms
12  * * *
13  * * *
14  * * *
15  221.183.55.86  38.813 ms  39.717 ms  43.458 ms
16  * * *
17  * 221.176.22.161  41.336 ms  40.912 ms
18  * * 221.176.15.209  77.113 ms
19  * * *
20  111.24.14.14  85.347 ms 111.24.14.22  77.766 ms 111.24.14.6  97.875 ms
21  39.156.27.1  90.045 ms 111.13.0.174  82.393 ms 111.13.188.38  79.342 ms
22  * 39.156.27.5  88.427 ms  80.218 ms
23  * * *
24  * * *
25  * * *
26  * * *
27  39.156.69.79  81.358 ms  89.887 ms  78.932 ms
```

traceroute

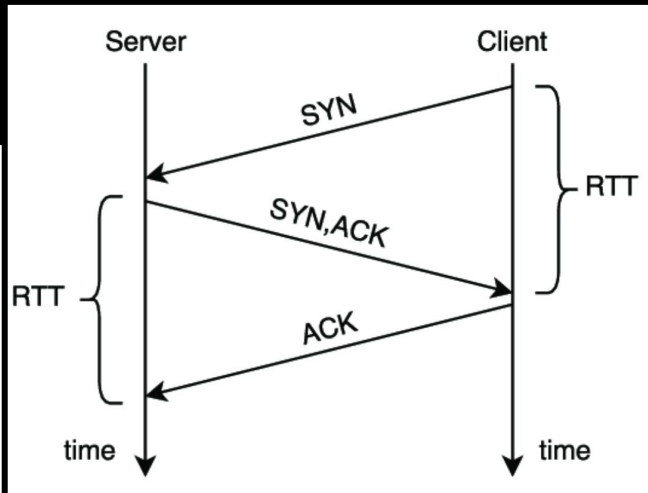
可顯示封包在IP網路經過的路由器的IP位址

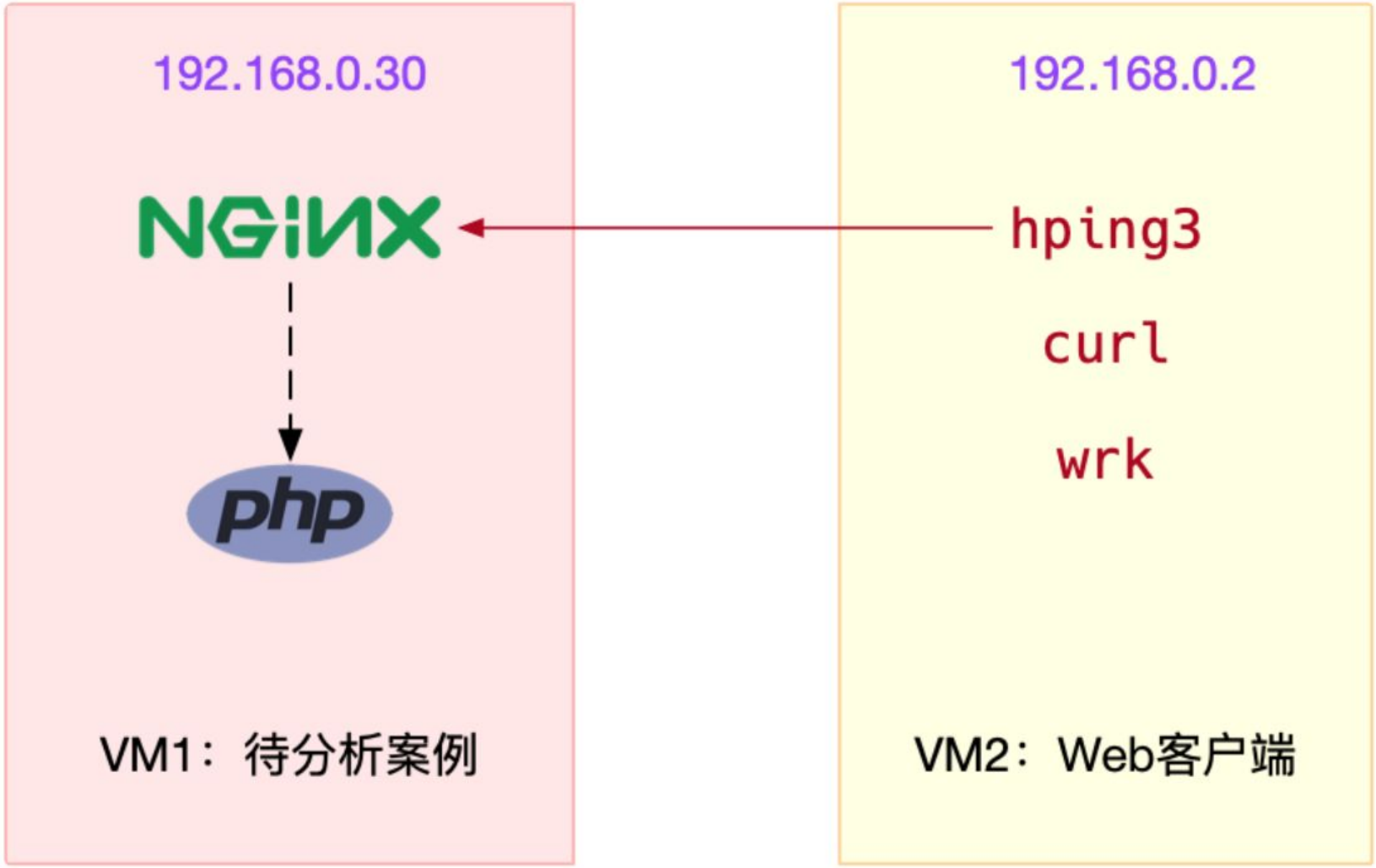
預設3個資料包的次數

PS:自己常用的方式 curl

```
[root@localhost ~]# curl -o /dev/null -s -w "time_namelookup:  %{time_namelookup}\n time_connect:  %{time_connect}
(TCP handshake)\n time_appconnect:  %{time_appconnect} (SSL handshake)\n time_pretransfer:  %{time_pretransfer}\n
time_redirect:  %{time_redirect}\n time_starttransfer:  %{time_starttransfer}\n  ----------\n time_total:  %{time_t
otal}\n" http://www.baidu.com/
time_namelookup:  0.030
 time_connect:  0.071 (TCP handshake)
 time_appconnect:  0.000 (SSL handshake)
 time_pretransfer:  0.071
 time_redirect:  0.000
 time_starttransfer:  0.137
  ----------
 time_total:  0.140
```

time_connect
接近RTT數據




192.168.0.30
NGINX
php
VM1：待分析案例
192.168.0.2
hping3
curl
wrk
VM2：Web客户端

good

```
[root@localhost ~]# hping3 -c 3 -S -p 80 192.168.0.17
HPING 192.168.0.17 (enp0s3 192.168.0.17): S set, 40 headers + 0 data bytes
len=46 ip=192.168.0.17 ttl=64 DF id=0 sport=80 flags=SA seq=0 win=29200 rtt=1.7 ms
len=46 ip=192.168.0.17 ttl=64 DF id=0 sport=80 flags=SA seq=1 win=29200 rtt=1.1 ms
len=46 ip=192.168.0.17 ttl=64 DF id=0 sport=80 flags=SA seq=2 win=29200 rtt=1.6 ms

--- 192.168.0.17 hping statistic ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 1.1/1.5/1.7 ms
```

Nginx

```
[root@localhost ~]# hping3 -c 3 -S -p 8080 192.168.0.17
HPING 192.168.0.17 (enp0s3 192.168.0.17): S set, 40 headers + 0 data bytes
len=46 ip=192.168.0.17 ttl=64 DF id=0 sport=8080 flags=SA seq=0 win=29200 rtt=1.3 ms
len=46 ip=192.168.0.17 ttl=64 DF id=0 sport=8080 flags=SA seq=1 win=29200 rtt=1.7 ms
len=46 ip=192.168.0.17 ttl=64 DF id=0 sport=8080 flags=SA seq=2 win=29200 rtt=1.3 ms

--- 192.168.0.17 hping statistic ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 1.3/1.4/1.7 ms
```

good

```
[root@localhost ~]# wrk --latency -c 100 -t 2 --timeout 2 http://192.168.0.18/
Running 10s test @ http://192.168.0.18/
  2 threads and 100 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    48.55ms   87.97ms   1.01s    94.31%
    Req/Sec     1.59k   264.84     2.27k    72.86%
  Latency Distribution
     50%   28.91ms
     75%   33.44ms
     90%   42.99ms
     99%  527.83ms
  31802 requests in 10.08s, 25.78MB read
Requests/sec:   3154.42
Transfer/sec:      2.56MB
```

```
Thread Stats   Avg      Stdev
  Latency     9.19ms   12.32ms
  Req/Sec     6.20k   426.80
Latency Distribution
   50%    7.78ms
   75%    8.22ms
   90%    9.14ms
   99%   50.53ms
```

Nginx

```
[root@localhost ~]# wrk --latency -c 100 -t 2 --timeout 2 http://192.168.0.18:8080/
Running 10s test @ http://192.168.0.18:8080/
  2 threads and 100 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    45.42ms    6.24ms  74.85ms   88.08%
    Req/Sec     1.10k   108.66     1.52k    83.50%
  Latency Distribution
     50%   44.74ms
     75%   47.72ms
     90%   51.61ms
     99%   62.58ms
  21904 requests in 10.06s, 17.78MB read
Requests/sec:   2176.46
Transfer/sec:      1.77MB
```

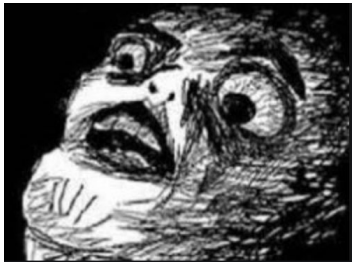
```
Thread Stats   Avg      Stdev
  Latency    43.60ms    6.41ms
  Req/Sec     1.15k   120.29
Latency Distribution
   50%   44.02ms
   75%   44.33ms
   90%   47.62ms
   99%   48.88ms
```

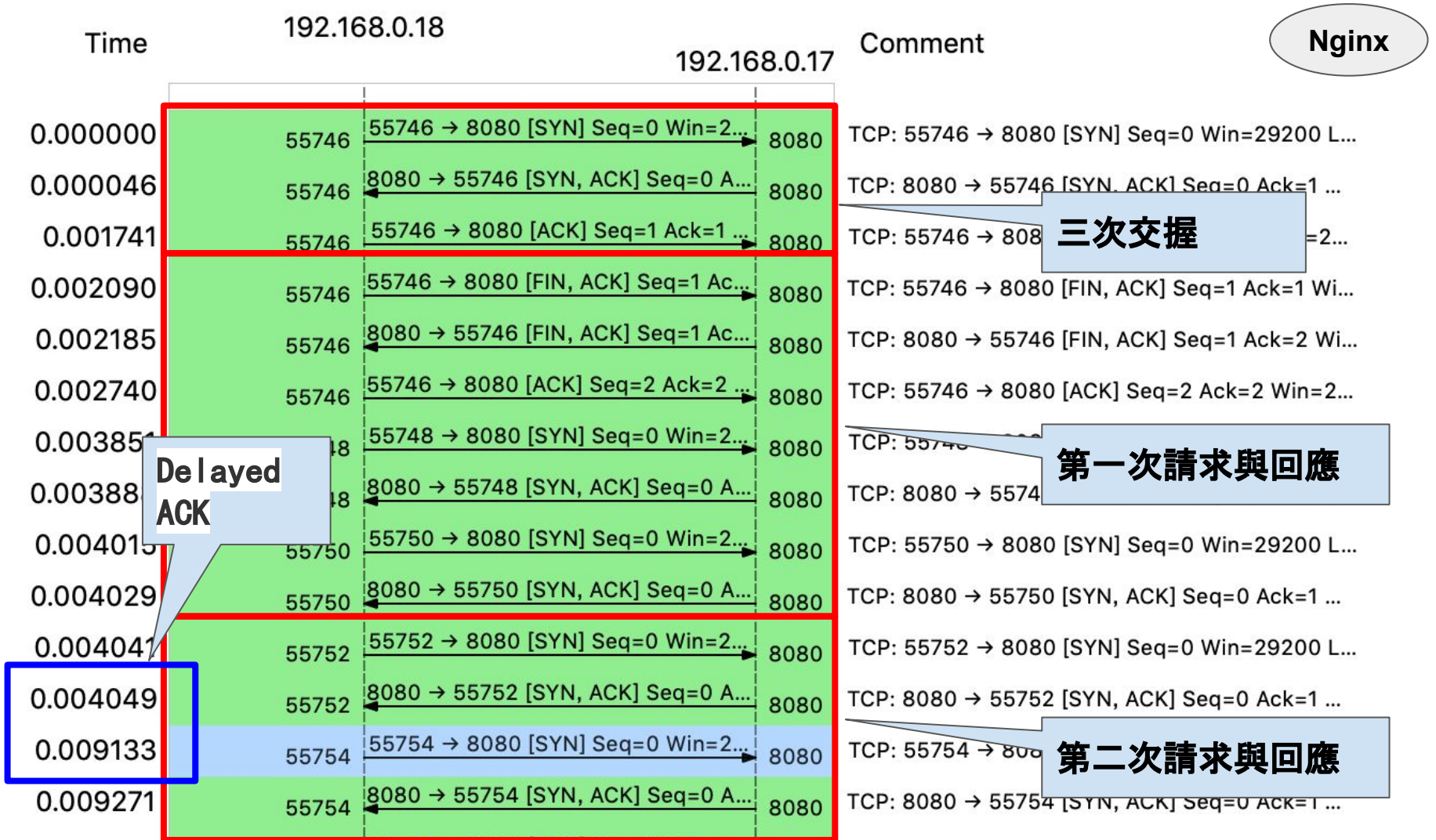
```
[root@localhost ~]# wrk --latency -c 100 -t 2 --timeout 2 http://192.168.0.18:8080/
Running 10s test @ http://192.168.0.18:8080/
  2 threads and 100 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    66.84ms  108.95ms   1.20s    93.58%
    Req/Sec     1.11k   198.16     1.70k    70.50%
  Latency Distribution
     50%   41.56ms
     75%   50.10ms
     90%   67.36ms
     99%  634.13ms
  22244 requests in 10.09s, 18.03MB read
Requests/sec:   2204.47
Transfer/sec:      1.79MB
```

教材改良後

```
[root@localhost ~]# wrk --latency -c 100 -t 2 --timeout 2 http://192.168.0.18:8080/
Running 10s test @ http://192.168.0.18:8080/
  2 threads and 100 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    52.03ms    8.39ms  86.59ms   76.48%
    Req/Sec     0.96k   121.72     1.20k    74.00%
  Latency Distribution
     50%   50.22ms
     75%   56.07ms
     90%   63.55ms
     99%   77.07ms
  19115 requests in 10.07s, 15.52MB read
Requests/sec:   1897.52
Transfer/sec:      1.54MB
```

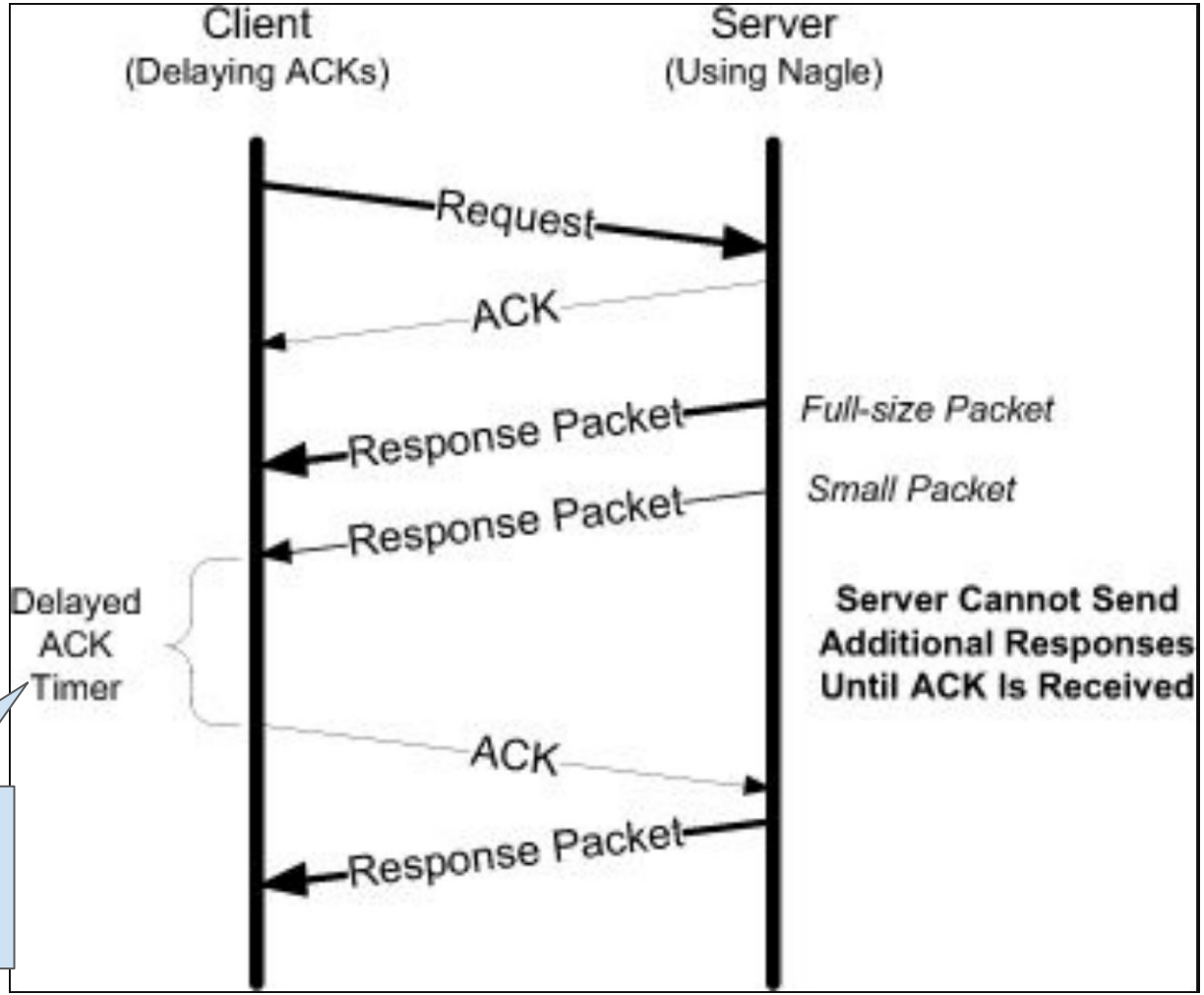
教材改良前




Time
192.168.0.18
192.168.0.17
Comment
Nginx
0.000000
55746 → 8080 [SYN] Seq=0 Win=2...
55746
8080
TCP: 55746 → 8080 [SYN] Seq=0 Win=29200 L...
0.000046
8080 → 55746 [SYN, ACK] Seq=0 A...
TCP: 8080 → 55746 [SYN, ACK] Seq=0 Ack=1 ...
0.001741
55746 → 8080 [ACK] Seq=1 Ack=1 ...
三次交握
0.002090
55746 → 8080 [FIN, ACK] Seq=1 Ac...
TCP: 55746 → 8080 [FIN, ACK] Seq=1 Ack=1 Wi...
0.002185
8080 → 55746 [FIN, ACK] Seq=1 Ac...
TCP: 8080 → 55746 [FIN, ACK] Seq=1 Ack=2 Wi...
0.002740
55746 → 8080 [ACK] Seq=2 Ack=2 ...
TCP: 55746 → 8080 [ACK] Seq=2 Ack=2 Win=2...
55748 → 8080 [SYN] Seq=0 Win=2...
第一次請求與回應
Delayed ACK
8080 → 55748 [SYN, ACK] Seq=0 A...
TCP: 8080 → 5574
0.004015
55750 → 8080 [SYN] Seq=0 Win=2...
TCP: 55750 → 8080 [SYN] Seq=0 Win=29200 L...
0.004029
8080 → 55750 [SYN, ACK] Seq=0 A...
TCP: 8080 → 55750 [SYN, ACK] Seq=0 Ack=1 ...
0.004041
55752 → 8080 [SYN] Seq=0 Win=2...
TCP: 55752 → 8080 [SYN] Seq=0 Win=29200 L...
0.004049
8080 → 55752 [SYN, ACK] Seq=0 A...
TCP: 8080 → 55752 [SYN, ACK] Seq=0 Ack=1 ...
0.009133
55754 → 8080 [SYN] Seq=0 Win=2...
第二次請求與回應
0.009271
8080 → 55754 [SYN, ACK] Seq=0 A...

Nagle（纳格算法）

TCP 协议中用于减少小包发送数量的一种优化算法，目的是为了提高实际带宽的利用率



等一下的概念

客戶端導致？

strace -f wrk --latency -c 100 -t 2 --timeout 2 http://192.168.0.18:8080/

```
[root@localhost ~]# strace -f wrk --latency -c 100 -t 2 --timeout 2 http://192.168.0.18:8080/ | grep setsockopt
execve("/usr/local/bin/wrk", ["wrk", "--latency", "-c", "100", "-t", "2", "--timeout", "2", "http://192.168.0.18:8080/"],
 0x7ffd0a85c968 /* 24 vars */) = 0
brk(NULL)                               = 0xc69000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7eff072e9000
[pid  1642] <... setsockopt resumed>)   = 0
[pid  1643] setsockopt(37, SOL_TCP, TCP_NODELAY, [1], 4 <unfinished ...>
[pid  1642] epoll_ctl(3, EPOLL_CTL_ADD, 36, {EPOLLIN|EPOLLOUT, {u32=36, u64=36}} <unfinished ...>
[pid  1643] <... setsockopt resumed>)   = 0
[pid  1642] <... epoll_ctl resumed>)    = 0
```

wrk 只设置了 TCP_NODELAY 选项，而没有设置 TCP_QUICKACK

Nginx設定檔

Nginx

```
                    '"$http_user_agent" "$http_x_forwa

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    tcp_nopush      off;
    tcp_nodelay     off;

    keepalive_timeout  65;
```

no起來！
關閉nagle演算

me的案例討論 by 2015

有一信用卡請款檔欲上傳至金流單位，但傳一半就TIME OUT失敗。
信用卡正負交易都很正常。

如果是您要怎麼查？

最後有解決 但沒探究原因

個人經驗查找問題 by RD

1. 先CURL看回應速度 => 查看路由有沒有問題

   => 看看傳輸是否太肥

2. 到該主機TOP是否CPU很忙 => 查看服務是否已卡死
3. 看程式在忙什麼
   a. DB撈很久 => 調整SQL
   b. 存IO => mount遠端儲存是否有問題
   c. 檢視程式流程減少不必要的外部連線或IO => 重構

參考文件:

wireshark MAC下載點: https://1.as.dl.wireshark.org/osx/

Nagle's Algorithm 和 Delayed ACK

:https://medium.com/fcamels-notes/nagles-algorithm-%E5%92%8C-delayed-ack-%E4%BB%A5%E5%8F%8A-minshall-%E7%9A%84%E5%8A%A0%E5%BC%B7%E7%89%88-8fadcb84d96f

Nginx性能優化: https://imququ.com/post/my-nginx-conf-for-wpo.html

CURL用法: https://blog.techbridge.cc/2019/02/01/linux-curl-command-tutorial/

hping3DDOS: https://kknews.cc/zh-tw/news/3n9lrya.html