TCP SOCKET中backlog参数的用途是什么?

在前年时，业务中遇到好多次因为PHP-FPM的backlog参数引发的性能问题，一直想去详细研究一番，还特意在2013年总结里提到这事《为何PHP5.5.6中fpm backlog Changed default listen() backlog to 65535》，然而，我稍于懒惰，一拖再拖，直至今日，才动脑去想，动笔去写。

2013年12月14发布的PHP5.5.6中，changelog中有一条变更，

> FPM: Changed default listen() backlog to 65535.

这条改动，是在10月28日改的，见increase backlog to the highest value everywhere

> It makes no sense to use -1 for *BSD (which is the highest value there)
> and still use 128 for Linux.
> Lets raise it right to up the limit and let the people lower it if they
> think that 3.5Mb is too much for a process.
> IMO this is better than silently dropping connections.

patch提交者认为，提高backlog数量，哪怕出现timeout之类错误，也比因为backlog满了之后，悄悄的忽略TCP SYN的请求要好。

当我最近开始想好好了解一下backlog的细节问题时，发现fpm的默认backlog已经不是65535了，现在是511了。（没写在changelog中，你没注意到，这不怪你。）我翻阅了github提交记录，找到这次改动，是在2014年7月22日，Set FPM_BACKLOG_DEFAULT to 511

> It is too large for php-fpm to set the listen backlog to 65535.
> It is realy NOT a good idea to clog the accept queue especially
> when the client or nginx has a timeout for this connection.
>
> Assume that the php-fpm qps is 5000. It will take 13s to completely
> consume the 65535 backloged connections. The connection maybe already
> have been closed cause of timeout of nginx or clients. So when we accept
> the 65535th socket, we get a broken pipe.
>
> Even worse, if hundreds of php-fpm processes get a closed connection
> they are just wasting time and resouces to run a heavy task and finally
> get error when writing to the closed connection(error: Broken Pipe).
>
> The really max accept queue size will be backlog+1(ie, 512 here). We take 511
> which is the same as nginx and redis.

其中理由是“backlog值为65535太大了。会导致前面的nginx（或者其他客户端）超时”，

而且提交者举例计算了一下，假设FPM的QPS为5000，那么65535个请求全部处理完需要13s的样子。但前端的nginx（或其他客户端）已经等待超时，关闭了这个连接。当FPM处理完之后，再往这个SOCKET ID 写数据时，却发现连接已关闭，得到的是“error: Broken Pipe”，在nginx、redis、apache里，默认的backlog值兜是511。故这里也建议改为511。（后来发现，此patch提交者，是360团队「基础架构快报」中《TCP三次握手之backlog》的作者shafreeck）

对于backlog的参数的含义，在linux kernel手册中也给了注释说明listen – listen for connections on a socket

> #include /* See NOTES */
> #include
> int listen(int sockfd, int backlog);
> Description
>
> listen() marks the socket referred to by sockfd as a passive socket, that is, as a socket that will be used to accept incoming connection requests using accept(2). The sockfd argument is a file descriptor that refers to a socket of type SOCK_STREAM or SOCK_SEQPACKET.
>
> The backlog argument defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED or, if the underlying protocol supports retransmission, the request may be ignored so that a later reattempt at connection succeeds.
>
> Notes
>
> To accept connections, the following steps are performed:
> 1.
> A socket is created with socket(2).
> 2.
> The socket is bound to a local address using bind(2), so that other sockets may be connect(2)ed to it.
> 3.
> A willingness to accept incoming connections and a queue limit for incoming connections are specified with listen().
> 4.
> Connections are accepted with accept(2).
> POSIX.1-2001 does not require the inclusion of , and this header file is not required on Linux. However, some historical (BSD) implementations required this header file, and portable applications are probably wise to include it.

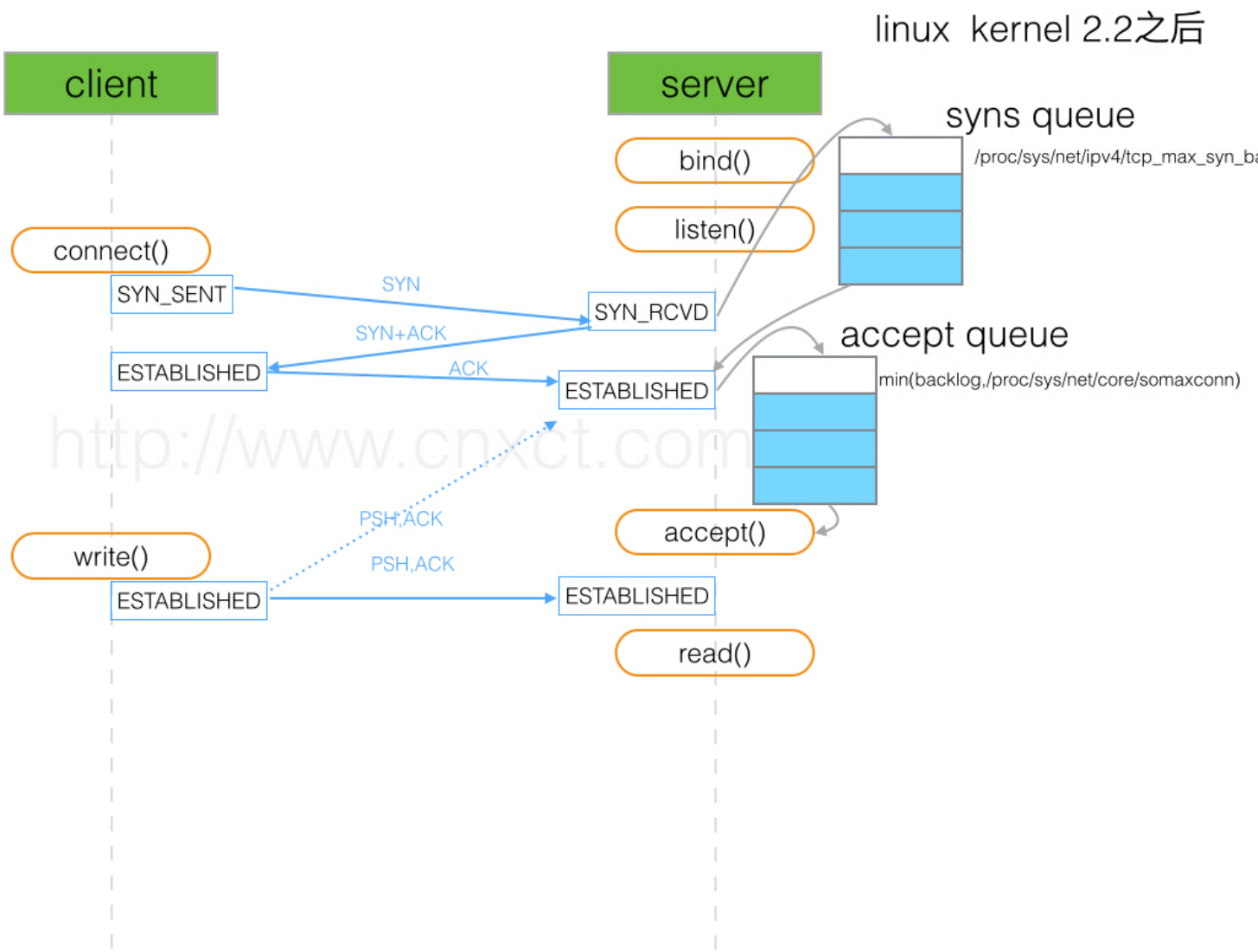
> The behavior of the backlog argument on TCP sockets changed with Linux 2.2. Now it specifies the queue length for completely established sockets waiting to be accepted, instead of the number of incomplete connection requests. The maximum length of the queue for incomplete sockets can be set using /proc/sys/net/ipv4/tcp_max_syn_backlog. When syncookies are enabled there is no logical maximum length and this setting is ignored. See tcp(7) for more information.
>
> If the backlog argument is greater than the value in /proc/sys/net/core/somaxconn, then it is silently truncated to that value; the default value in this file is 128. In kernels before 2.4.25, this limit was a hard coded value, SOMAXCONN, with the value 128.

backlog的定义是已连接但未进行accept处理的SOCKET队列大小，已是（并非syn的SOCKET队列）。如果这个队列满了，将会发送一个ECONNREFUSED错误信息给到客户端,即 linux 头文件 /usr/include/asm-generic/errno.h中定义的“Connection refused”，（如果协议不支持重传，该请求会被忽略）如下：

```
01
02
03  #define ENETDOWN    100 /* Network is down */
04  #define ENETUNREACH 101 /* Network is unreachable */
05  #define ENETRESET   102 /* Network dropped connection because of reset */
06  #define ECONNABORTED    103 /* Software caused connection abort */
07  #define ECONNRESET  104 /* Connection reset by peer */
08  #define ENOBUFS     105 /* No buffer space available */
09  #define EISCONN     106 /* Transport endpoint is already connected */
10  #define ENOTCONN    107 /* Transport endpoint is not connected */
11  #define ESHUTDOWN   108 /* Cannot send after transport endpoint shutdown */
12  #define ETOOMANYREFS    109 /* Too many references: cannot splice */
13  #define ETIMEDOUT   110 /* Connection timed out */
14  #define ECONNREFUSED    111 /* Connection refused */
15  #define EHOSTDOWN   112 /* Host is down */
16  #define EHOSTUNREACH    113 /* No route to host */
17  #define EALREADY    114 /* Operation already in progress */
18  #define EINPROGRESS 115 /* Operation now in progress */
```

在linux 2.2以前，backlog大小包括了半连接状态和全连接状态两种队列大小。linux 2.2以后，分离为两个backlog来分别限制半连接SYN_RCVD状态的未完成连接队列大小跟全连接ESTABLISHED状态的已完成连接队列大小。互联网上常见的TCP SYN FLOOD恶意DOS攻击方式就是用/proc/sys/net/ipv4/tcp_max_syn_backlog来控制的，可参见《TCP洪水攻击（SYN Flood）的诊断和处理》。

在使用listen函数时，内核会根据传入参数的backlog跟系统配置参数/proc/sys/net/core/somaxconn中，二者取最小值，作为“ESTABLISHED状态之后，完成TCP连接，等待服务程序ACCEPT”的队列大小。在kernel 2.4.25之前，是写死在代码常量SOMAXCONN，默认值是128。在kernel 2.4.25之后，在配置文件/proc/sys/net/core/somaxconn (即 /etc/sysctl.conf 之类)中可以修改。我稍微整理了流程图，如下：



如图，服务端收到客户端的syn请求后，将这个请求放入syns queue中，然后服务器端回复syn+ack给客户端，等收到客户端的ack后，将此连接放入accept queue。

大约了解其参数代表意义之后，我稍微测试了一番，并抓去了部分数据，首先确认系统默认参数

```
root@vmware-cnxct:/home/cfc4n# cat /proc/sys/net/core/somaxconn
```

```
128

root@vmware-cnxct:/home/cfc4n# ss -lt

State      Recv-Q Send-Q         Local Address:Port                       Peer
Address:Port

LISTEN     0      128                        *:ssh                              *:*

LISTEN     0      128                0.0.0.0:9000                               *:*

LISTEN     0      128                       *:http                             *:*

LISTEN     0      128                       :::ssh                             :::*

LISTEN     0      128                      :::http                             :::*
```

在FPM的配置中，listen.backlog值默认为511，而如上结果中看到的Send-Q却是128，可见确实是以/proc/sys/net/core/somaxconn跟listen参数的最小值作为backlog的值。

```


cfc4n@cnxct:~$ ab -n 10000 -c 300 http:

This is ApacheBench, Version 2.3 <$Revision: 1604373 $>

Copyright 1996 Adam Twiss, Zeus Technology Ltd, http:

Licensed to The Apache Software Foundation, http:

Benchmarking 172.16.218.128 (be patient)

Completed 1000 requests

Completed 2000 requests

Completed 3000 requests

Completed 4000 requests

Completed 5000 requests

Completed 6000 requests

Completed 7000 requests

Completed 8000 requests

Completed 9000 requests

Completed 10000 requests

Finished 10000 requests
```

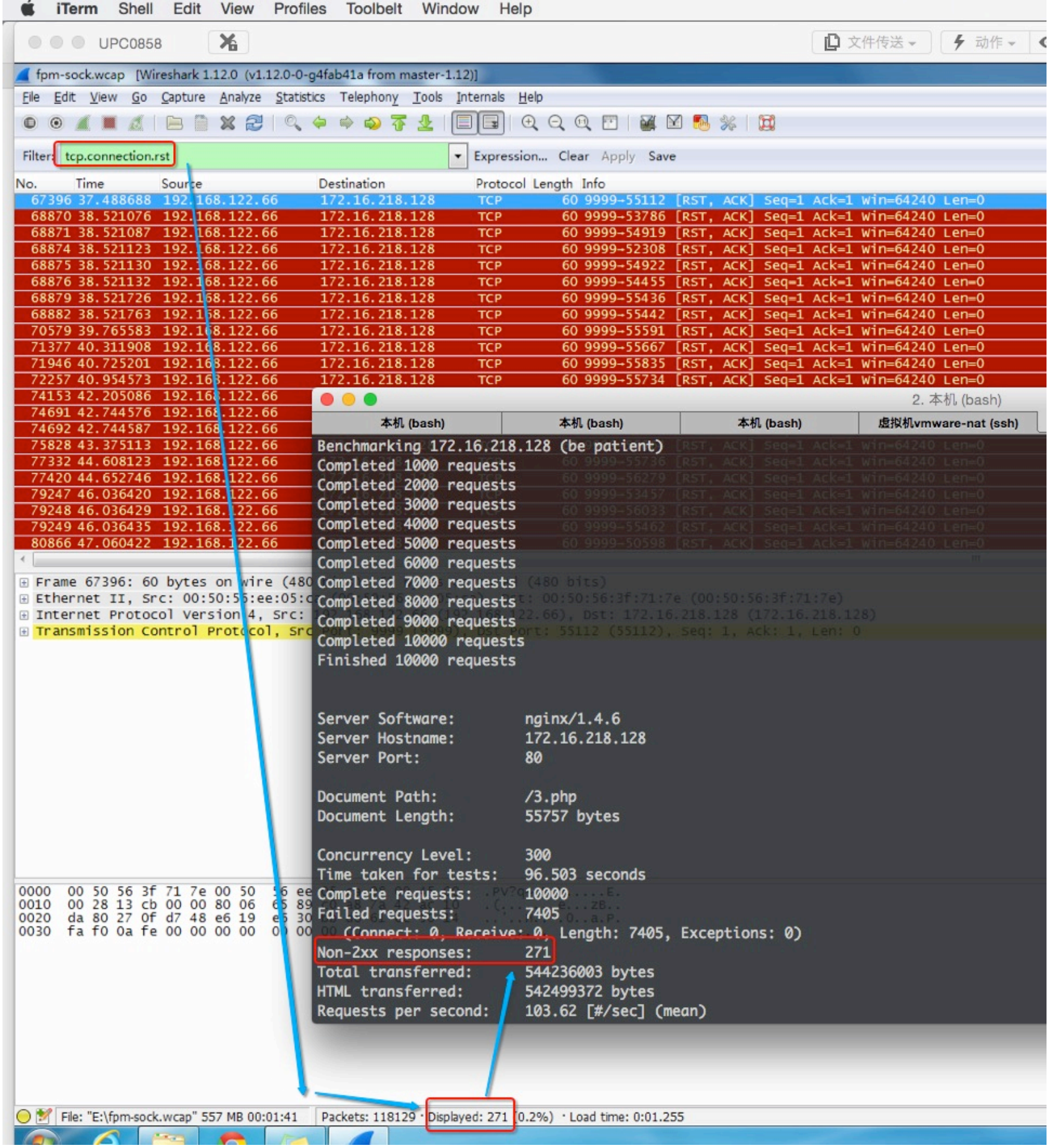
```
Server Software:        nginx/1.4.6
Server Hostname:        172.16.218.128
Server Port:            80
Document Path:          /3.php
Document Length:        55757 bytes
Concurrency Level:      300
Time taken for tests:   96.503 seconds
Complete requests:      10000
Failed requests:        7405
   (Connect: 0, Receive: 0, Length: 7405, Exceptions: 0)
Non-2xx responses:      271
Total transferred:      544236003 bytes
HTML transferred:       542499372 bytes
Requests per second:    103.62 [#/sec] (mean)
Time per request:       2895.097 [ms] (mean)
Time per request:       9.650 [ms] (mean, across all concurrent requests)
Transfer rate:          5507.38 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    9  96.7      0    1147
Processing:     8 2147 6139.2    981   60363
Waiting:        8 2137 6140.1    970   60363
Total:          8 2156 6162.8    981   61179

Percentage of the requests served within a certain time (ms)
  50%    981
  66%   1074
  75%   1192
  80%   1283
  90%   2578
  95%   5352
  98%  13534
  99%  42346
 100%  61179 (longest request)
```

```
52
53
54
55
56
```

apache ab这边的结果中，非2xx的http响应有271个，在NGINX日志数据如下：

```
01 root@vmware-cnxct:/var/log/nginx# cat grep.error.log |wc -l
02 271
03 root@vmware-cnxct:/var/log/nginx# cat grep.access.log |wc -l
04 10000
05 root@vmware-cnxct:/var/log/nginx# cat grep.access.log |awk '{print $9}'|sort|uniq
   -c
06    9729 200
07     186 502
08      85 504
09 root@vmware-cnxct:/var/log/nginx# cat grep.error.log |awk '{print $8  $9  $10
   $11}'|sort |uniq -c
10     186 (111: Connection refused) while
11      85 out (110: Connection timed
```

从nginx结果中看出，本次压测总请求数为10000。http 200响应数量9729个；http 502响应数量186个；http 504响应数量未85个；即非2xx响应总数为502+504总数，为271个。同时，也跟error.log中数据吻合。同时，也跟TCP数据包中的RST包数量吻合。



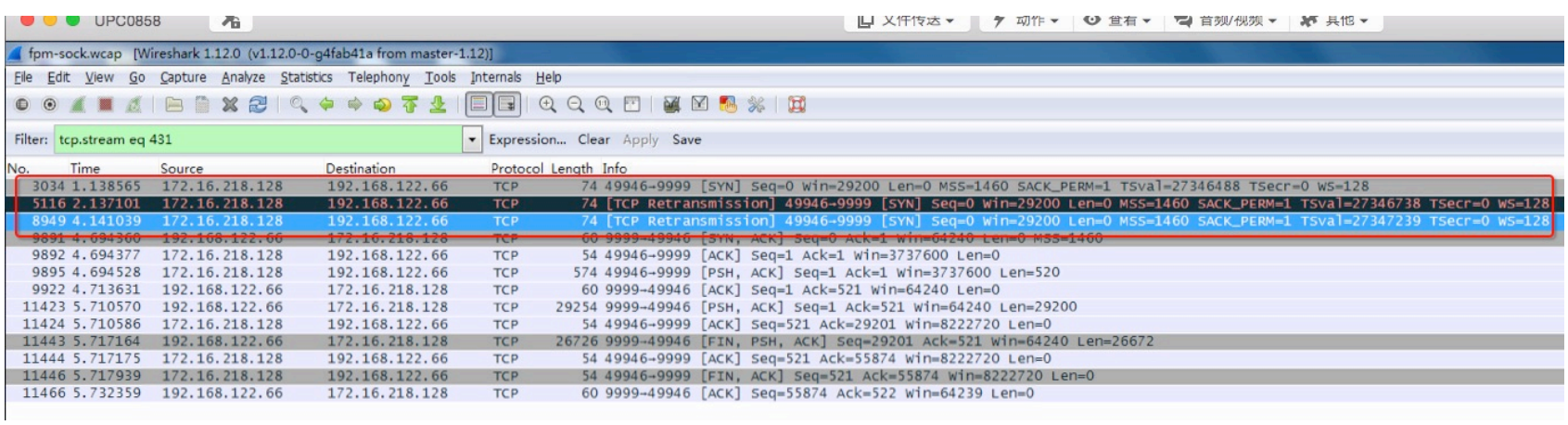
在nginx error中，错误号为111，错误信息为“Connection refused”的有186条，对应着所有http 502响应错误的请求；错误号为110，错误信息为“Connection timed out”的有85条,对应着所有http 504响应错误的请求。在linux errno.h头文件定义中，错误号111对应着ECONNREFUSED；错误号110对应着ETIMEDOUT。linux man手册里，对listen参数的说明中，也提到，若client连不上server时，会报告ECONNREFUSED的错。

Nginx error日志中的详细错误如下:

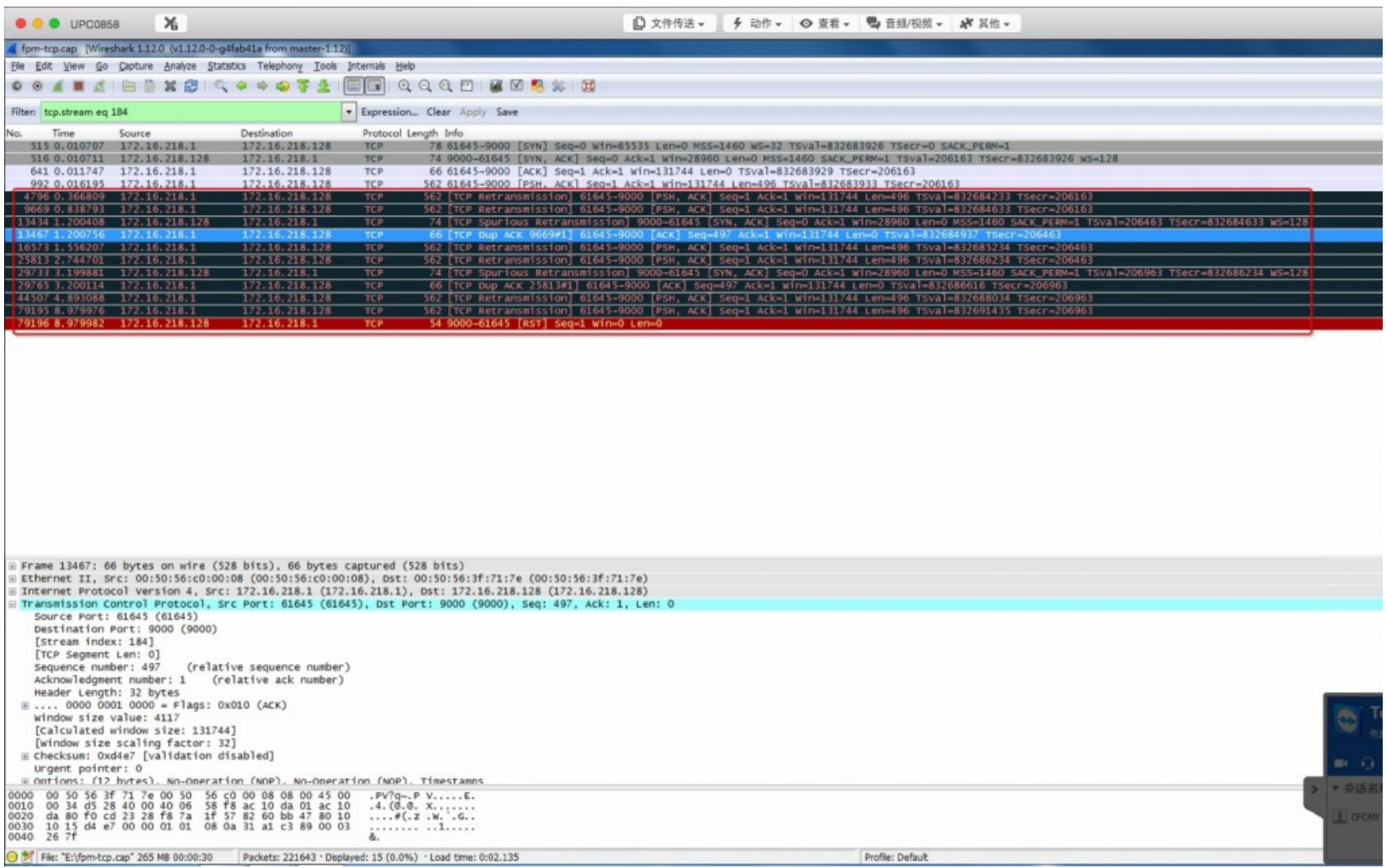
```
1
54414#0: *24135 upstream timed out (110: Connection timed out) while connecting to
2
upstream, client: 172.16.218.1, server: localhost, request: "GET /3.php
HTTP/1.0", upstream: "fastcgi://192.168.122.66:9999", host: "172.16.218.128"

3
[error] 54416#0: *38728 connect() failed (111: Connection refused) while
```

```
connecting to upstream, client: 172.16.218.1, server: localhost, request: "GET
/3.php HTTP/1.0", upstream: "fastcgi://192.168.122.66:9999", host:
"172.16.218.128"
```

在压测的时候，我用tcpdump抓了通讯包，配合通信包的数据，也可以看出，当backlog为某128时，accept queue队列塞满后，TCP建立的三次握手完成，连接进入ESTABLISHED状态，客户端（nginx）发送给PHP-FPM的数据，FPM处理不过来，没有调用accept将其从accept ququе队列取出时，那么就没有ACK包返回给客户端nginx，nginx那边根据TCP 重传机制会再次发从尝试...报了“111: Connection refused”错。当SYNS QUEUE满了时，TCPDUMP的结果如下，不停重传SYN包。



对于已经调用accept函数，从accept queue取出，读取其数据的TCP连接，由于FPM本身处理较慢，以至于NGINX等待时间过久，直接终止了该fastcgi请求，返回“110: Connection timed out”。当FPM处理完成后，往FD里写数据时，发现前端的nginx已经断开连接了，就报了“Write broken pipe”。当ACCEPT QUEUE满了时，TCPDUMP的结果如下，不停重传PSH SCK包。（别问我TCP RTO重传的机制，太复杂了，太深奥了 、[TCP的定时器系列 — 超时重传定时器]）

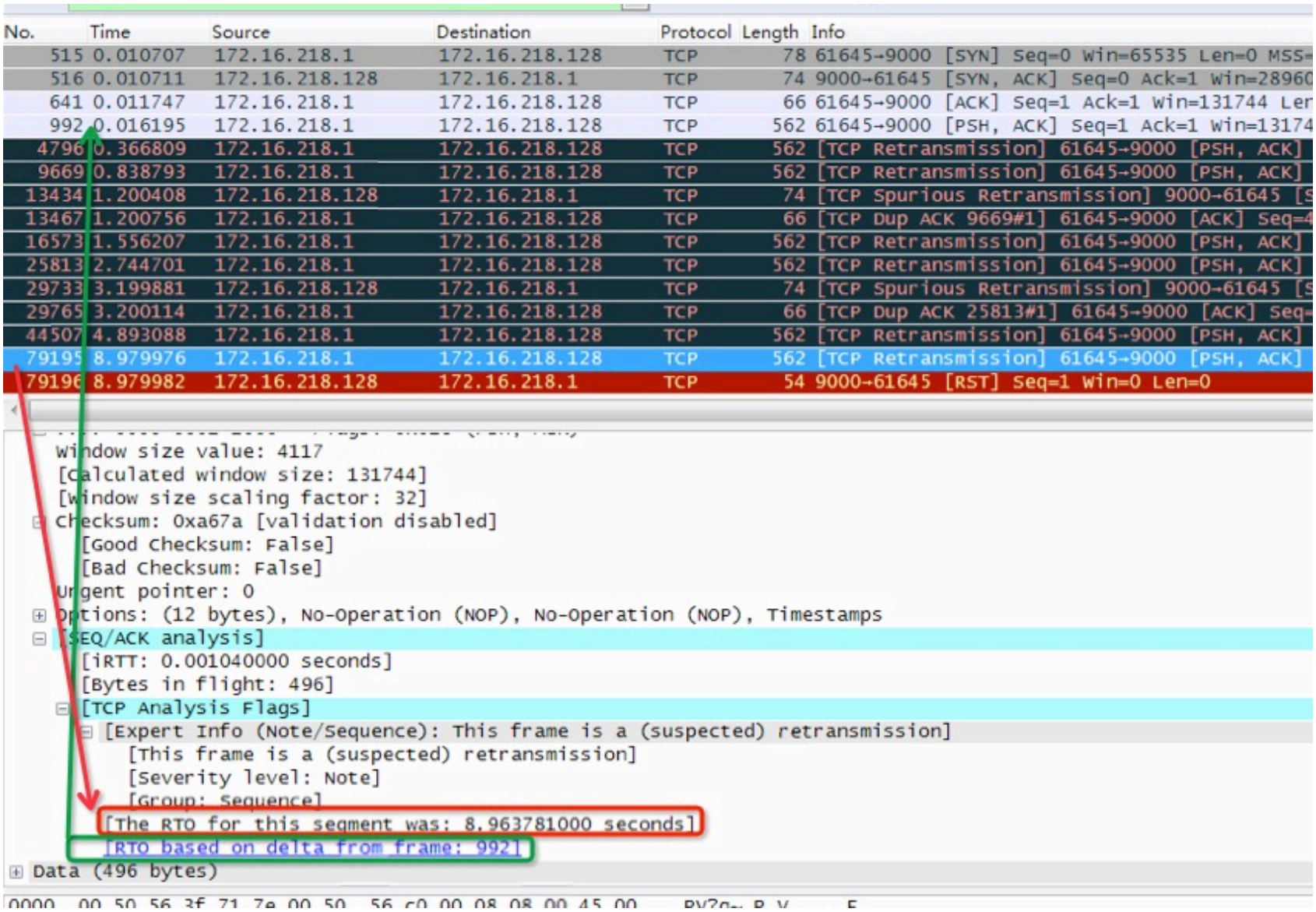


对于这些结论，我尝试搜了很多资料，后来在360公司的「基础架构快报」中也看到了他们的研究资料《TCP三次握手之backlog》，也验证了我的结论。

关于ACCEPT QUEUE满了之后的表现问题，早上IM鑫爷给我指出几个错误，感谢批评及指导，在这里，我把这个问题再详细描述一下。如上图所示

- NO.515 client发SYN到server，我的seq是0，消息包内容长度为0.（这里的seq并非真正的0，而是wireshark为了显示更好阅读，使用了Relative SeqNum相对序号）
- NO.516 server回SYN ACK给client，我的seq是0，消息包内容长度是0，已经收到你发的seq 1 之前的TCP包。(请发后面的)
- NO.641 client发ACK给server，我是seq 1 ，消息包内容长度是0，已经收到你发的seq 1 之前的TCP包。
- NO.992 client发PSH给server，我是seq 1 ，消息包内容长度是496，已经收到你发的seq 1 之前的TCP包。
- .....等了一段时间之后（这里约0.2s左右）
- NO.4796 client没等到对方的ACK包，开始TCP retransmission这个包，我是seq 1，消息包长度496，已经收到你发的seq 1 之前的TCP包。
- .....又...等了一段时间
- NO.9669 client还是没等到对方的ACK包，又开始TCP retransmission这个包，我是seq 1，消息包长度496，已经收到你发的seq 1 之前的TCP包。
- NO.13434 server发了SYN ACK给client，这里是tcp spurious retransmission 伪重传，我的seq是0，消息包内容长度是0，已经收到你发的seq 1 之前的TCP包。距离其上次发包给client是NO.516 已1秒左右了，因为没有收到NO.641 包ACK。这时，client收到过server的SYN,ACK包，将此TCP 连接状态改为ESTABLISHED,而server那边没有收到client的ACK包，则其TCP连接状态是SYN_RCVD状态。（感谢IM鑫爷指正）也可能是因为accept queue满了，暂时不能将此TCP连接从syns queue拉到accept queue，导致这情况，这需要翻阅内核源码才能确认。
- NO.13467 client发TCP DUP ACK包给server，其实是重发了N0.641 ,只是seq变化了，因为要包括它之前发送过的seq的序列号总和。即..我的seq 497 ，消息包内容长度是0，已经收到你发的seq 1 之前的TCP包。
- NO.16573 client继续重新发消息数据给server，包的内容还是NO.992的内容，因为之前发的几次，都没收到确认。
- NO.25813 client继续重新发消息数据给server，包的内容还还是NO.992的内容，仍没收到确认。（参见下图中绿色框内标识）
- NO.29733 server又重复了NO.13434包的流程，原因也一样，参见NO.13434包注释
- NO.29765 client只好跟NO.13467一样，重发ACK包给server。
- NO.44507 重复NO.16573的步骤
- NO.79195 继续重复NO.16573的步骤
- NO.79195 server立刻直接回了RST包，结束会话

详细的包内容备注在后面，需要关注的不光是包发送顺序，包的seq重传之类，还有一个重要的，TCP retransmission timeout，即TCP超时重传。对于这里已经抓到的数据包，wireshark可以看下每次超时重传的时间间隔，如下图：



RTO的重传次数是系统可配置的，见/proc/sys/net/ipv4/tcp_retries1 ，而重传时间间隔，间隔增长频率等，是比较复杂的方式计算出来的，见《TCP/IP重传超时–RTO》。

backlog大小设置为多少合适?
从上面的结论中可以看出，这跟FPM的处理能力有关，backlog太大了，导致FPM处理不过来，nginx那边等待超时，断开连接，报504 gateway timeout错。同时FPM处理完准备write 数据给nginx时，发现TCP连接断开了，报“Broken pipe”。backlog太小的话，NGINX之类client，根本进入不了FPM的accept queue，报“502 Bad Gateway”错。所以，这还得去根据FPM的QPS来决定backlog的大小。计算方式最好为QPS=backlog。对了这里的QPS是正常业务下的QPS，千万别用echo hello world这种结果的QPS去欺骗自己。当然，backlog的数值，如果指定在FPM中的话，记得把操作系统的net.core.somaxconn设置的起码比它大。另外，ubuntu server 1404上/proc/sys/net/core/somaxconn 跟/proc/sys/net/ipv4/tcp_max_syn_backlog 默认值都是128，这个问题，我为了抓数据，测了好几遍才发现。
对于测试时，TCP数据包已经drop掉的未进入syns queue，以及未进入accept queue的数据包，可以用netstat -s来查看：

```
root@vmware-cnxct:/# netstat -s
```

```
TcpExt:
    //...
    91855 times the listen queue of a socket overflowed
    102324 SYNs to LISTEN sockets dropped   //未进入syns queue的数据包数量
    444 packets directly queued to recvmsg prequeue.
    30408 bytes directly in process context from backlog
    //...
    TCPSackShiftFallback: 27
    TCPBacklogDrop: 2334    //未进入accept queue的数据包数量
    TCPTimeWaitOverflow: 229347
    TCPReqQFullDoCookies: 11591
    TCPRcvCoalesce: 29062
    //...

```

经过相关资料查阅，技术点研究，再做一番测试之后，又加深了我对TCP通讯知识点的记忆，以及对sync queue、accept queue所处环节知识点薄弱的补充，也是蛮有收获，这些知识，在以后的纯TCP通讯程序研发过程中，包括高性能的互联网通讯中，想必有很大帮助，希望自己能继续找些案例来实践检验一下对这些知识的掌握。

参考资料

- TCP三次握手之backlog
- linux里的backlog详解
- tcp-queue-的一些问题
- The meaning of listen(2)’s backlog parameter
- TCP 的那些事儿（上）
- TCP/IP重传超时–RTO
- TCP的定时器系列—超时重传定时器
- Coping with the TCP TIME-WAIT state on busy Linux servers