TCP三次握手之backlog

2014-11-12 基础架构快报

注：本文可结合《web常见问题排查》（回复 4）进行阅读。
欢迎业务团队的兄弟提供更多的案例，直接回复或联系 g-infra@360.cn

TCP三次握手之backlog

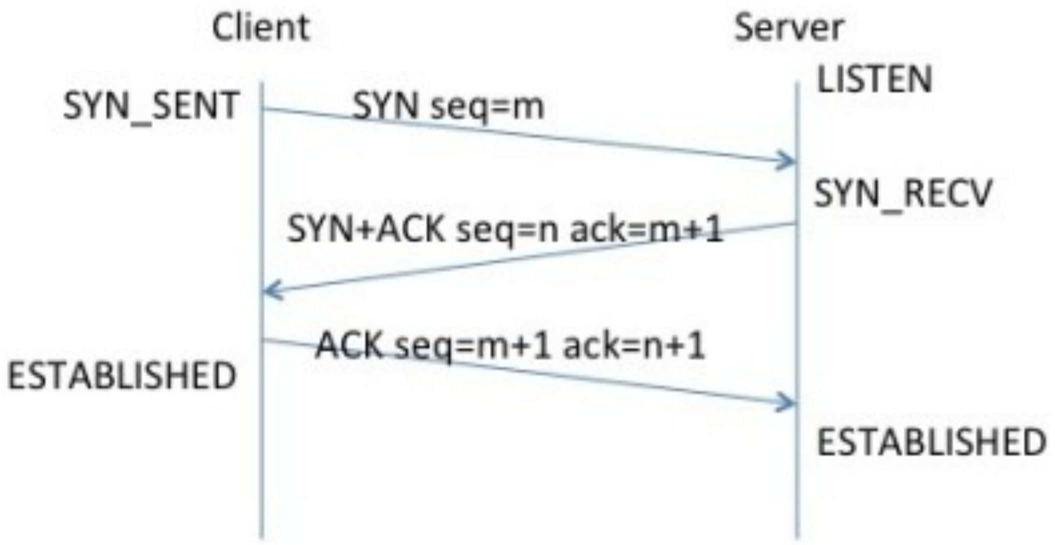
Backlog设置不当引来的问题

TCP可靠传输

- 有序
  - 为每一个字节分配一个序列号
  - 接收方维护序列号顺序
- 丢包
  - 对收到的序列号进行应答
  - 重传无应答的序列号
- SYN，FIN各占一个序列号，ACK，RST不占序列号
- RST不需要应答

- SYN synchronise packet
  - 协商各自数据开始的序列号（ISN）
- ACK Acknowledgement
  - 应答数据包
- ISN initial sequence number
  - 初始序列号

三次握手流程



Client

Server

LISTEN

SYN_SENT

SYN seq=m

SYN_RECV

SYN+ACK seq=n ack=m+1

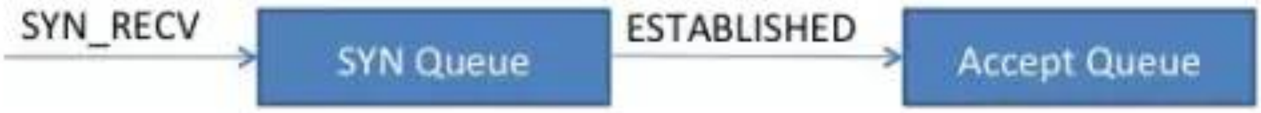
ACK seq=m+1 ack=n+1

ESTABLISHED

ESTABLISHED

Server端状态维护

- Server 端接收到SYN
  - 保存socket等待对方ACK
- Server端接收到ACK
  - 保存socket等待用户程序调用accept

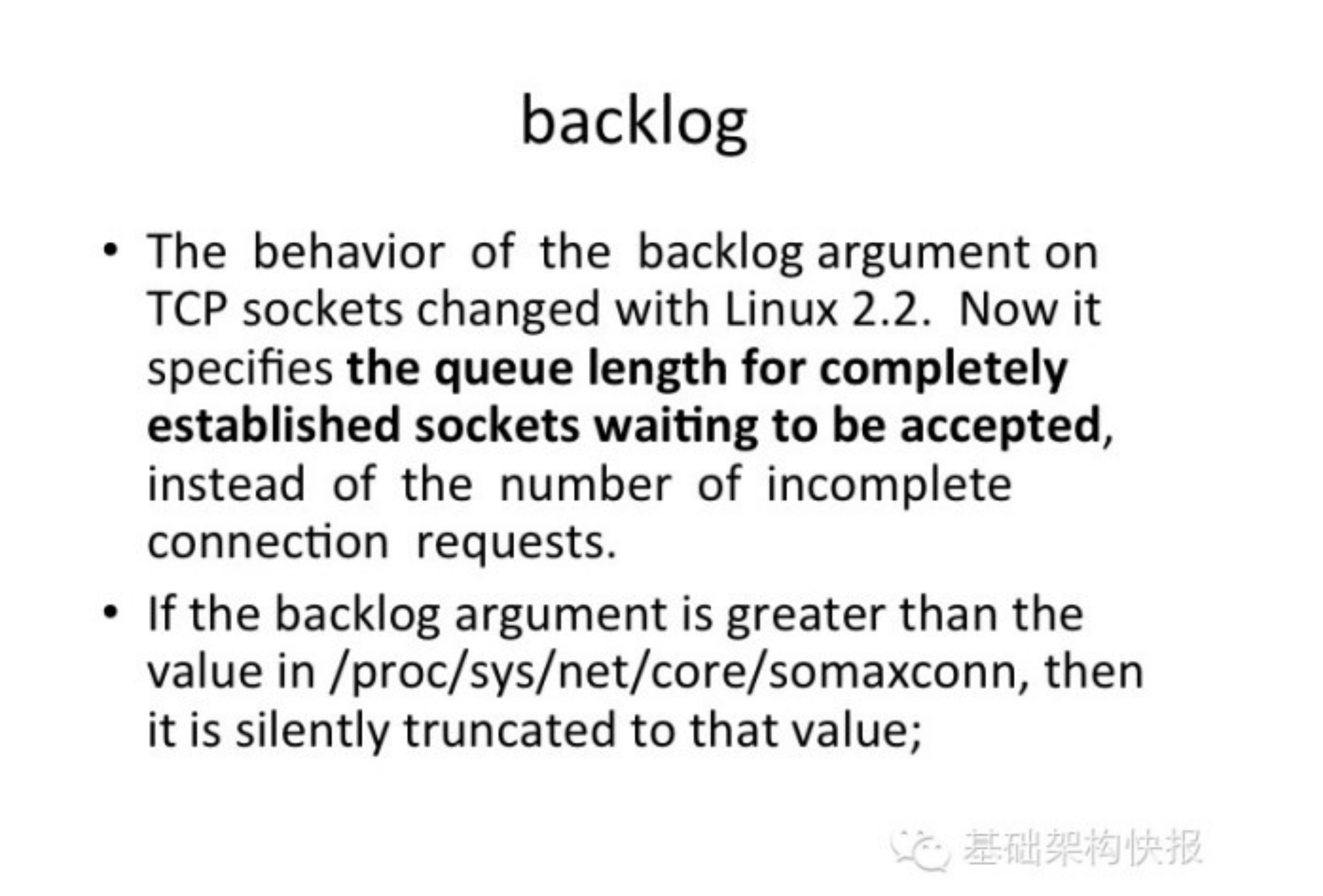


SYN_RECV

SYN Queue

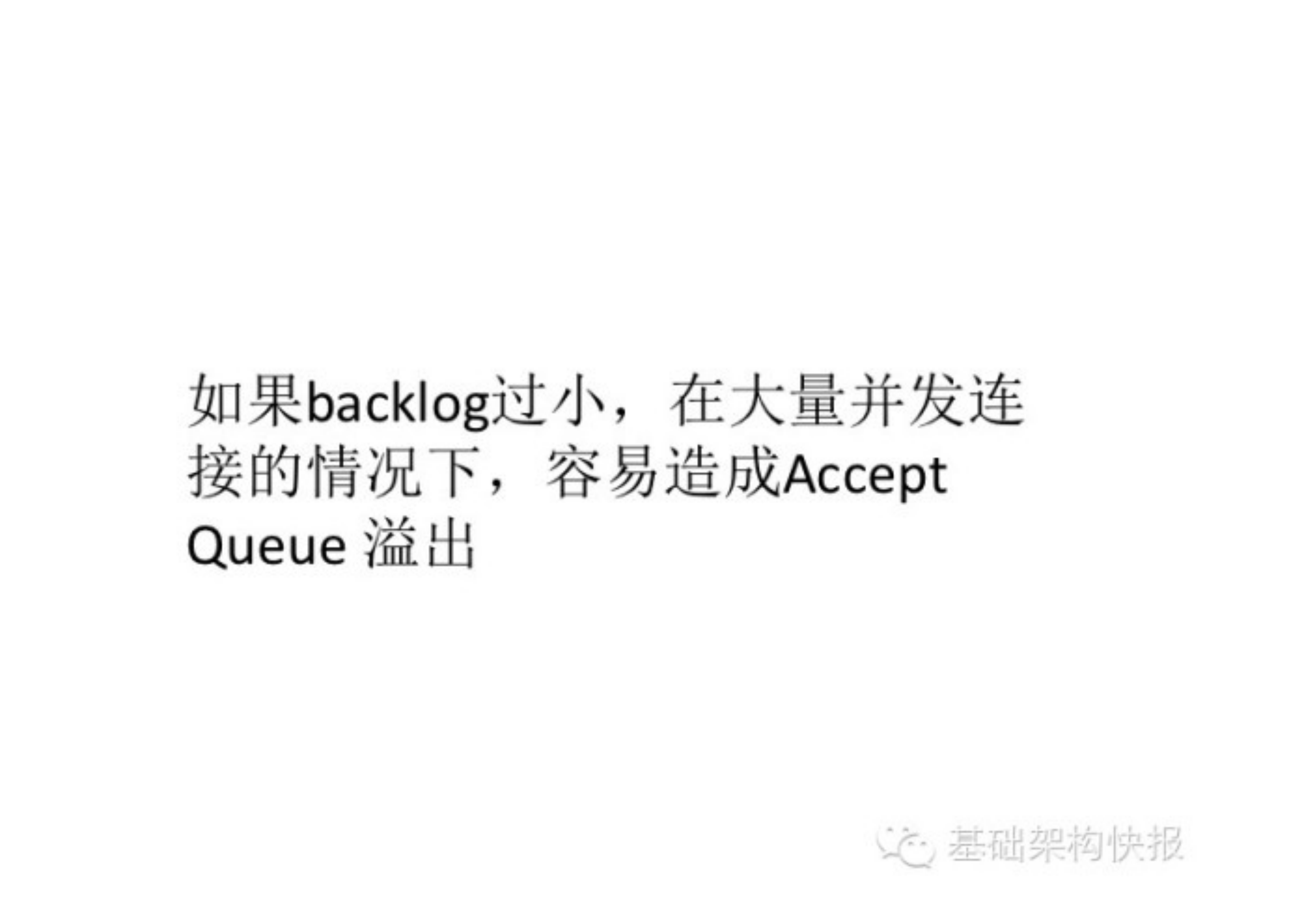
ESTABLISHED

Accept Queue



backlog

- 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.
- If the backlog argument is greater than the value in /proc/sys/net/core/somaxconn, then it is silently truncated to that value;



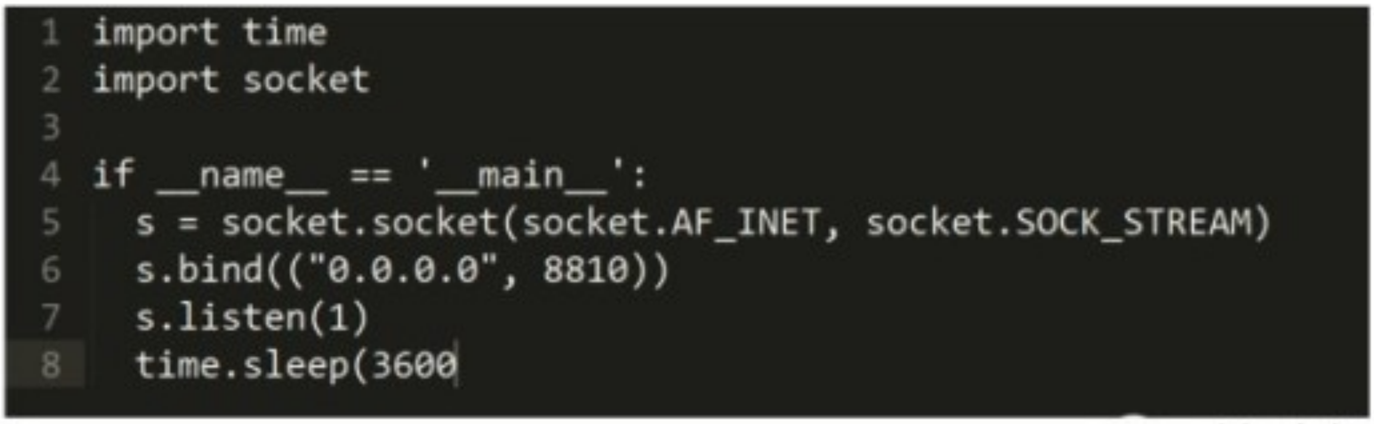
如果backlog过小，在大量并发连接的情况下，容易造成Accept Queue 溢出

Accept Queue溢出

- 三次握手
  - LISTEN状态下，接收到SYN，怎么处理？
  - SYN_RECV 状态下，接收到ACK，怎么处理？

Accept Queue溢出（SYN）-server

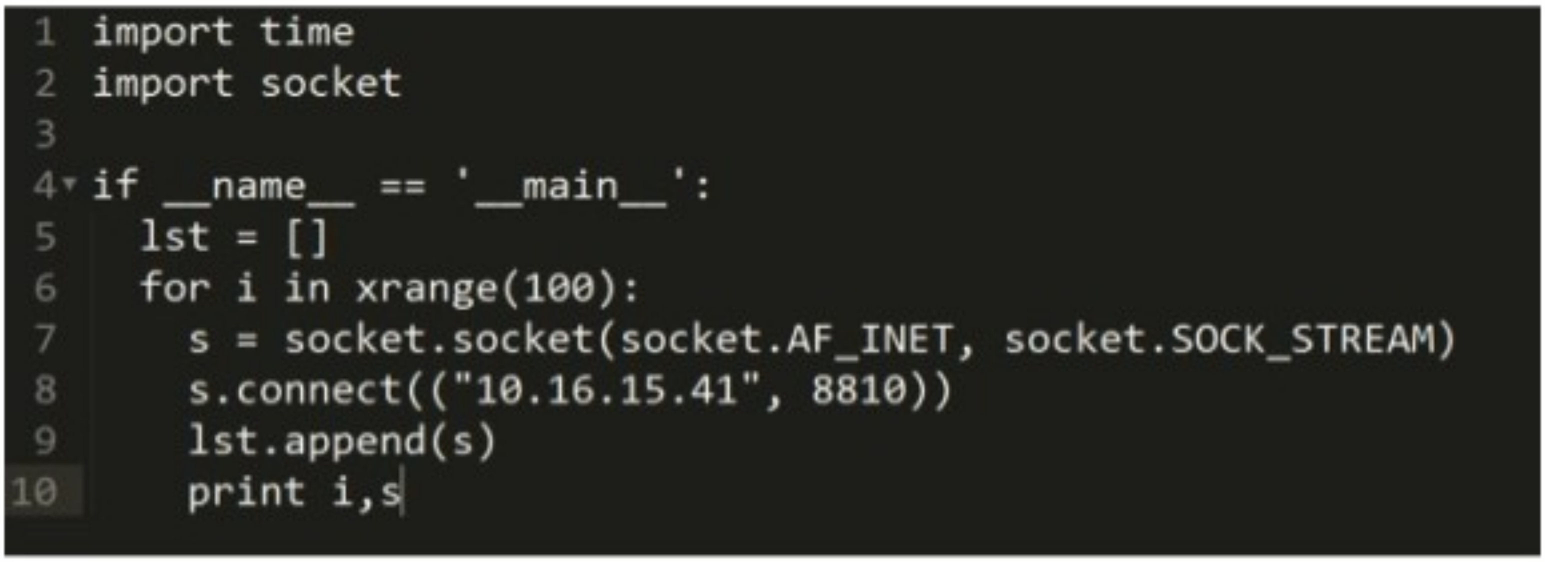
- net.ipv4.tcp_max_syn_backlog = 8192
- net.core.somaxconn = 204800
- netstat -s | grep LISTEN
  - 1653 SYNs to LISTEN sockets ignored



```
import time
import socket

if __name__ == '__main__':
  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  s.bind(("0.0.0.0", 8810))
  s.listen(1)
  time.sleep(3600
```

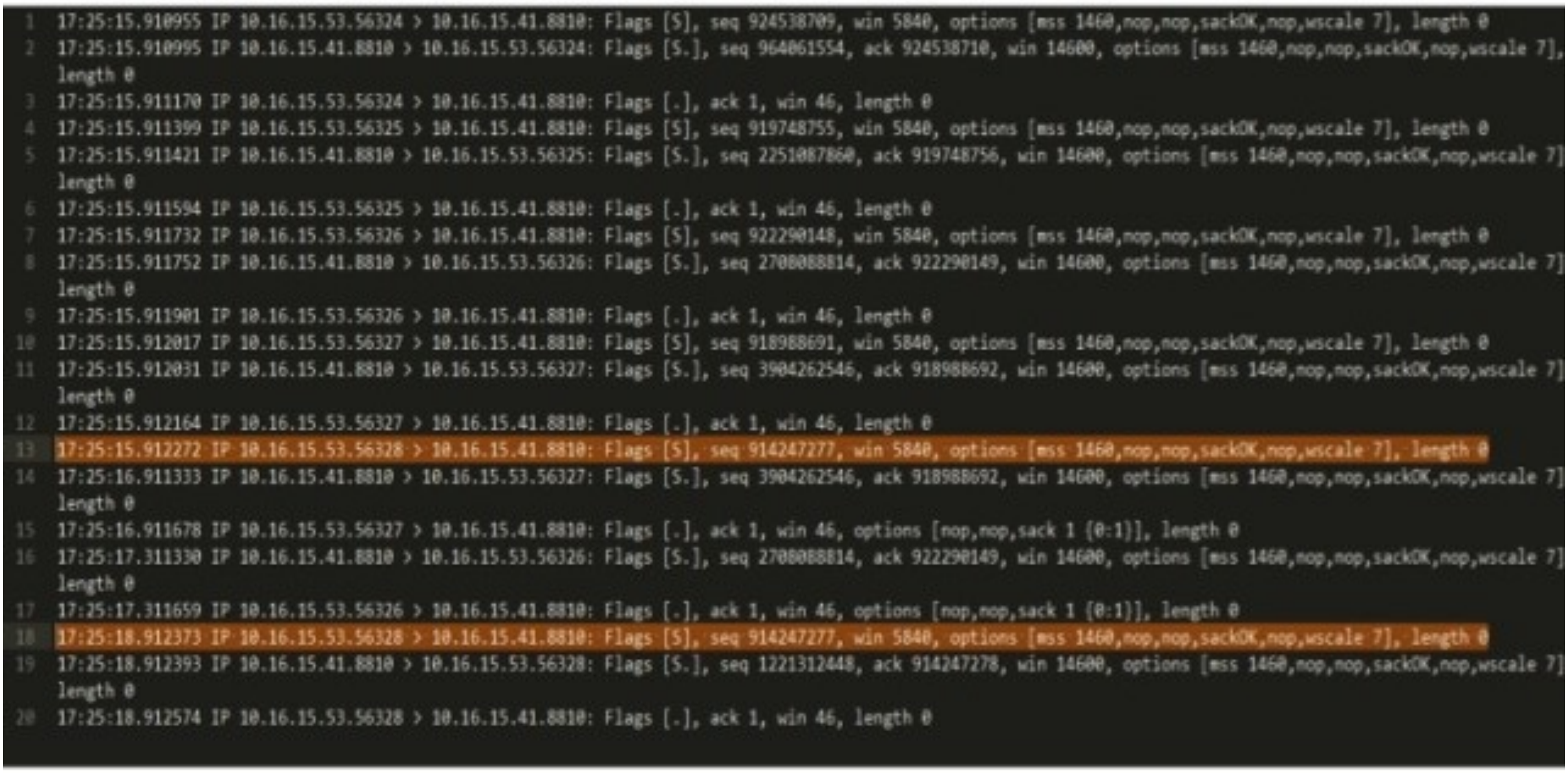
Accept Queue溢出（SYN）-client



```
import time
import socket

if __name__ == '__main__':
  lst = []
  for i in xrange(100):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect(("10.16.15.41", 8810))
    lst.append(s)
    print i,s
```

tcpdump

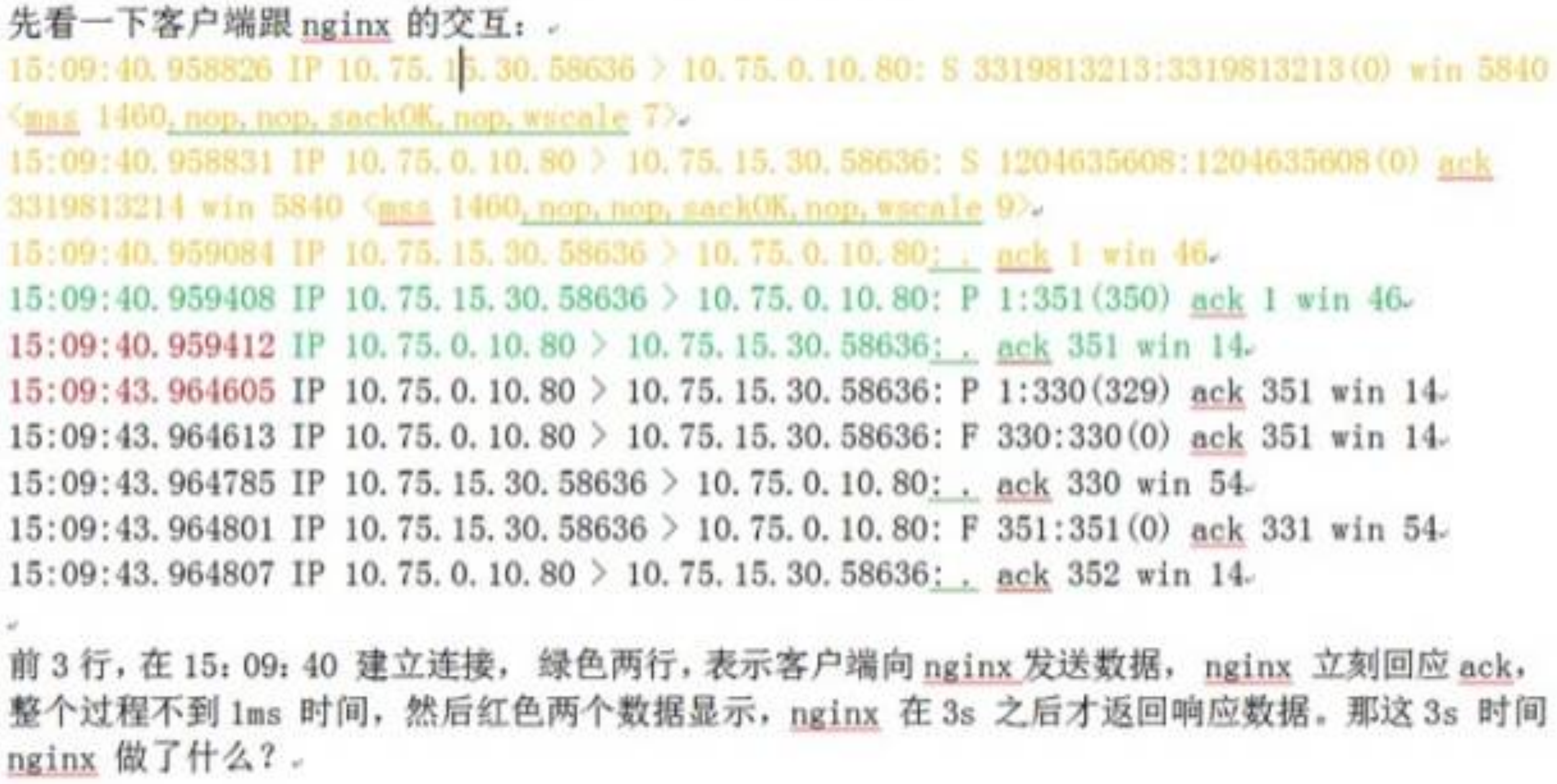


```
17:25:15.910955 IP 10.16.15.53.56324 > 10.16.15.41.8810: Flags [S], seq 924538709, win 5840, options [mss 1460,nop,nop,sackOK,nop,wscale 7], length 0
17:25:15.910995 IP 10.16.15.41.8810 > 10.16.15.53.56324: Flags [S.], seq 964061554, ack 924538710, win 14600, options [mss 1460,nop,nop,sackOK,nop,wscale 7],
length 0
17:25:15.911170 IP 10.16.15.53.56324 > 10.16.15.41.8810: Flags [.], ack 1, win 46, length 0
17:25:15.911399 IP 10.16.15.53.56325 > 10.16.15.41.8810: Flags [S], seq 919748755, win 5840, options [mss 1460,nop,nop,sackOK,nop,wscale 7], length 0
17:25:15.911421 IP 10.16.15.41.8810 > 10.16.15.53.56325: Flags [S.], seq 2251087860, ack 919748756, win 14600, options [mss 1460,nop,nop,sackOK,nop,wscale 7]
length 0
17:25:15.911594 IP 10.16.15.53.56325 > 10.16.15.41.8810: Flags [.], ack 1, win 46, length 0
17:25:15.911732 IP 10.16.15.53.56326 > 10.16.15.41.8810: Flags [S], seq 922290148, win 5840, options [mss 1460,nop,nop,sackOK,nop,wscale 7], length 0
17:25:15.911752 IP 10.16.15.41.8810 > 10.16.15.53.56326: Flags [S.], seq 2708088814, ack 922290149, win 14600, options [mss 1460,nop,nop,sackOK,nop,wscale 7]
length 0
17:25:15.911901 IP 10.16.15.53.56326 > 10.16.15.41.8810: Flags [.], ack 1, win 46, length 0
17:25:15.912017 IP 10.16.15.53.56327 > 10.16.15.41.8810: Flags [S], seq 918988691, win 5840, options [mss 1460,nop,nop,sackOK,nop,wscale 7], length 0
17:25:15.912031 IP 10.16.15.41.8810 > 10.16.15.53.56327: Flags [S.], seq 3904262546, ack 918988692, win 14600, options [mss 1460,nop,nop,sackOK,nop,wscale 7]
length 0
17:25:15.912164 IP 10.16.15.53.56327 > 10.16.15.41.8810: Flags [.], ack 1, win 46, length 0
17:25:15.912272 IP 10.16.15.53.56328 > 10.16.15.41.8810: Flags [S], seq 914247277, win 5840, options [mss 1460,nop,nop,sackOK,nop,wscale 7], length 0
17:25:16.911333 IP 10.16.15.41.8810 > 10.16.15.53.56327: Flags [S.], seq 3904262546, ack 918988692, win 14600, options [mss 1460,nop,nop,sackOK,nop,wscale 7]
length 0
17:25:16.911678 IP 10.16.15.53.56327 > 10.16.15.41.8810: Flags [.], ack 1, win 46, options [nop,nop,sack 1 {0:1}], length 0
17:25:17.311330 IP 10.16.15.41.8810 > 10.16.15.53.56326: Flags [S.], seq 2708088814, ack 922290149, win 14600, options [mss 1460,nop,nop,sackOK,nop,wscale 7]
length 0
17:25:17.311659 IP 10.16.15.53.56326 > 10.16.15.41.8810: Flags [.], ack 1, win 46, options [nop,nop,sack 1 {0:1}], length 0
17:25:18.912373 IP 10.16.15.53.56328 > 10.16.15.41.8810: Flags [S], seq 914247277, win 5840, options [mss 1460,nop,nop,sackOK,nop,wscale 7], length 0
17:25:18.912393 IP 10.16.15.41.8810 > 10.16.15.53.56328: Flags [S.], seq 1221312448, ack 914247278, win 14600, options [mss 1460,nop,nop,sackOK,nop,wscale 7]
length 0
17:25:18.912574 IP 10.16.15.53.56328 > 10.16.15.41.8810: Flags [.], ack 1, win 46, length 0
```

案例分析（1）

- 背景
  - nginx为七层反向代理
  - 开放平台自己开发“透明代理”，代理第三方接口
  - 客户端HTTP请求→ NGINX → 透明代理
- 问题
  - 透明代理接口存在大量慢请求

案例分析（1）

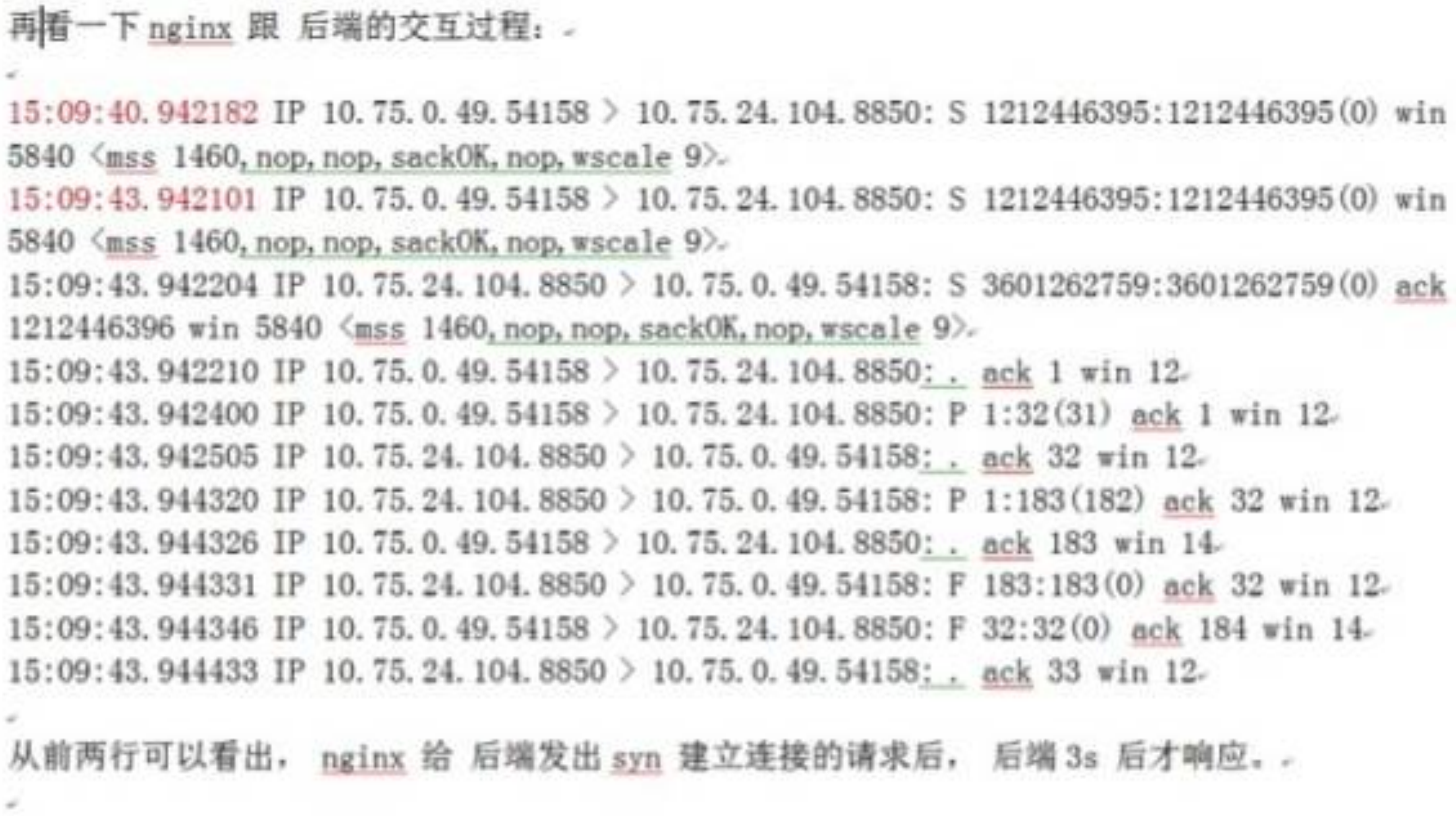


先看一下客户端跟 nginx 的交互：

```
15:09:40.958826 IP 10.75.15.30.58636 > 10.75.0.10.80: S 3319813213:3319813213(0) win 5840
<mss 1460,nop,nop,sackOK,nop,wscale 7>
15:09:40.958831 IP 10.75.0.10.80 > 10.75.15.30.58636: S 1204635608:1204635608(0) ack
3319813214 win 5840 <mss 1460,nop,nop,sackOK,nop,wscale 9>
15:09:40.959084 IP 10.75.15.30.58636 > 10.75.0.10.80: . ack 1 win 46
15:09:40.959408 IP 10.75.15.30.58636 > 10.75.0.10.80: P 1:351(350) ack 1 win 46
15:09:40.959412 IP 10.75.0.10.80 > 10.75.15.30.58636: . ack 351 win 14
15:09:43.964605 IP 10.75.0.10.80 > 10.75.15.30.58636: P 1:330(329) ack 351 win 14
15:09:43.964613 IP 10.75.0.10.80 > 10.75.15.30.58636: F 330:330(0) ack 351 win 14
15:09:43.964785 IP 10.75.15.30.58636 > 10.75.0.10.80: . ack 330 win 54
15:09:43.964801 IP 10.75.15.30.58636 > 10.75.0.10.80: F 351:351(0) ack 331 win 54
15:09:43.964807 IP 10.75.0.10.80 > 10.75.15.30.58636: . ack 352 win 14
```

前 3 行，在 15：09：40 建立连接， 绿色两行，表示客户端向 nginx 发送数据， nginx 立刻回应 ack，整个过程不到 1ms 时间，然后红色两个数据显示，nginx 在 3s 之后才返回响应数据。那这 3s 时间 nginx 做了什么？

案例分析（1）



再看一下 nginx 跟 后端的交互过程：

```
15:09:40.942182 IP 10.75.0.49.54158 > 10.75.24.104.8850: S 1212446395:1212446395(0) win
5840 <mss 1460,nop,nop,sackOK,nop,wscale 9>
15:09:43.942101 IP 10.75.0.49.54158 > 10.75.24.104.8850: S 1212446395:1212446395(0) win
5840 <mss 1460,nop,nop,sackOK,nop,wscale 9>
15:09:43.942204 IP 10.75.24.104.8850 > 10.75.0.49.54158: S 3601262759:3601262759(0) ack
1212446396 win 5840 <mss 1460,nop,nop,sackOK,nop,wscale 9>
15:09:43.942210 IP 10.75.0.49.54158 > 10.75.24.104.8850: . ack 1 win 12
15:09:43.942400 IP 10.75.0.49.54158 > 10.75.24.104.8850: P 1:32(31) ack 1 win 12
15:09:43.942505 IP 10.75.24.104.8850 > 10.75.0.49.54158: . ack 32 win 12
15:09:43.944320 IP 10.75.24.104.8850 > 10.75.0.49.54158: P 1:183(182) ack 32 win 12
15:09:43.944326 IP 10.75.0.49.54158 > 10.75.24.104.8850: . ack 183 win 14
15:09:43.944331 IP 10.75.24.104.8850 > 10.75.0.49.54158: F 183:183(0) ack 32 win 12
15:09:43.944346 IP 10.75.0.49.54158 > 10.75.24.104.8850: F 32:32(0) ack 184 win 14
15:09:43.944433 IP 10.75.24.104.8850 > 10.75.0.49.54158: . ack 33 win 12
```

从前两行可以看出， nginx 给 后端发出 syn 建立连接的请求后， 后端 3s 后才响应。

结论

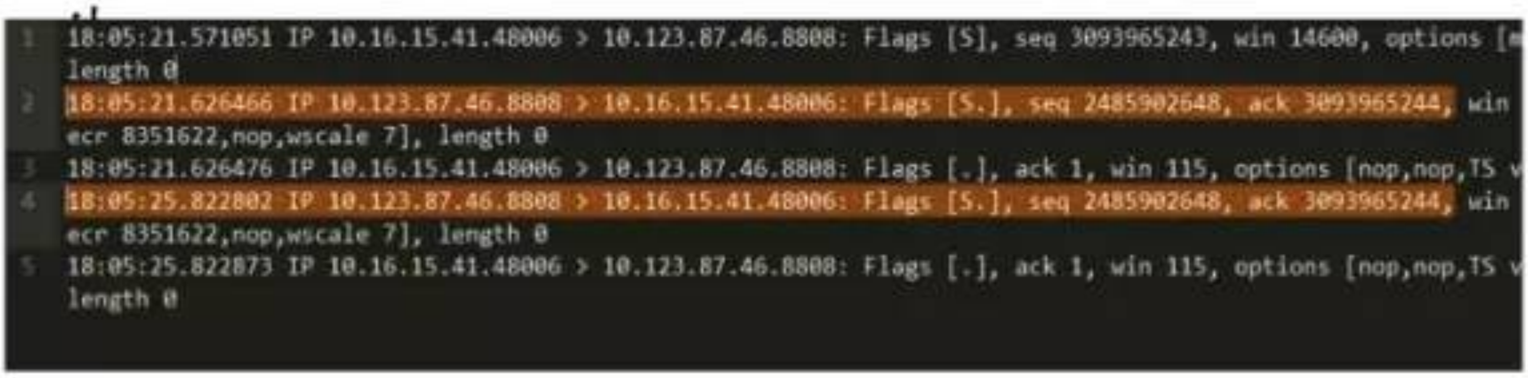
- Backlog 设置过小，　导致Accept Queue溢出
  - SYN 被丢弃，导致3s重传
- 解决方案
  - 增加backlog到512（原为50）
  - 修改somaxconn 到512

案例分析（2）

- 背景
  - Libarchive自动化测试
  - Testserver 随机生成RAR/ZIP文件
  - Testclient 访问Testserver，获取生成的文件及MD5
  - 为了简单，所有调用都用block方式
- 问题
  - 程序运行一段时间后，程序永久堵塞住
  - Strace显示testclient 堵塞在recvmsg

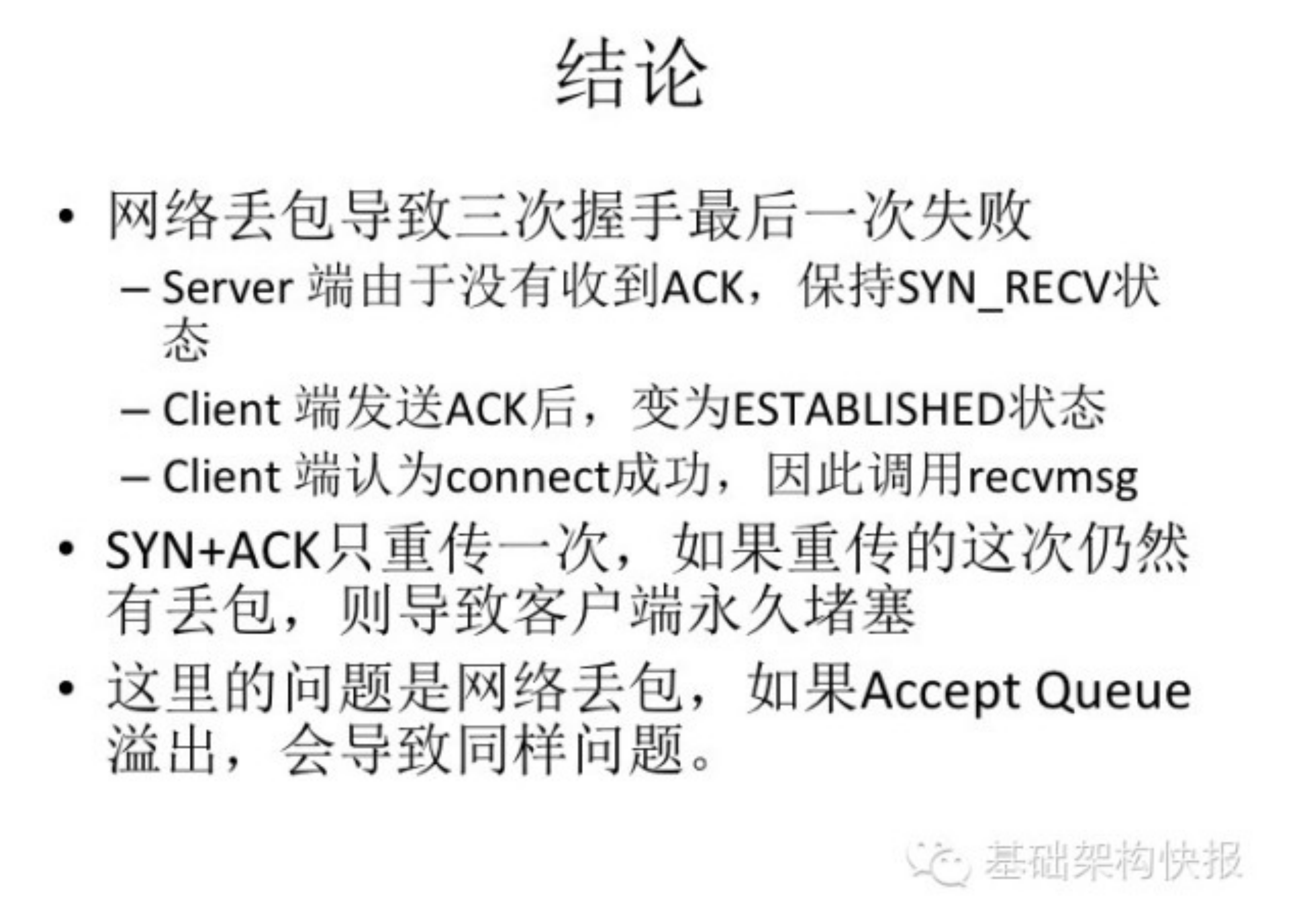
案例分析（2）

- 由于网络丢包，　导致第三次握手的ACK丢失



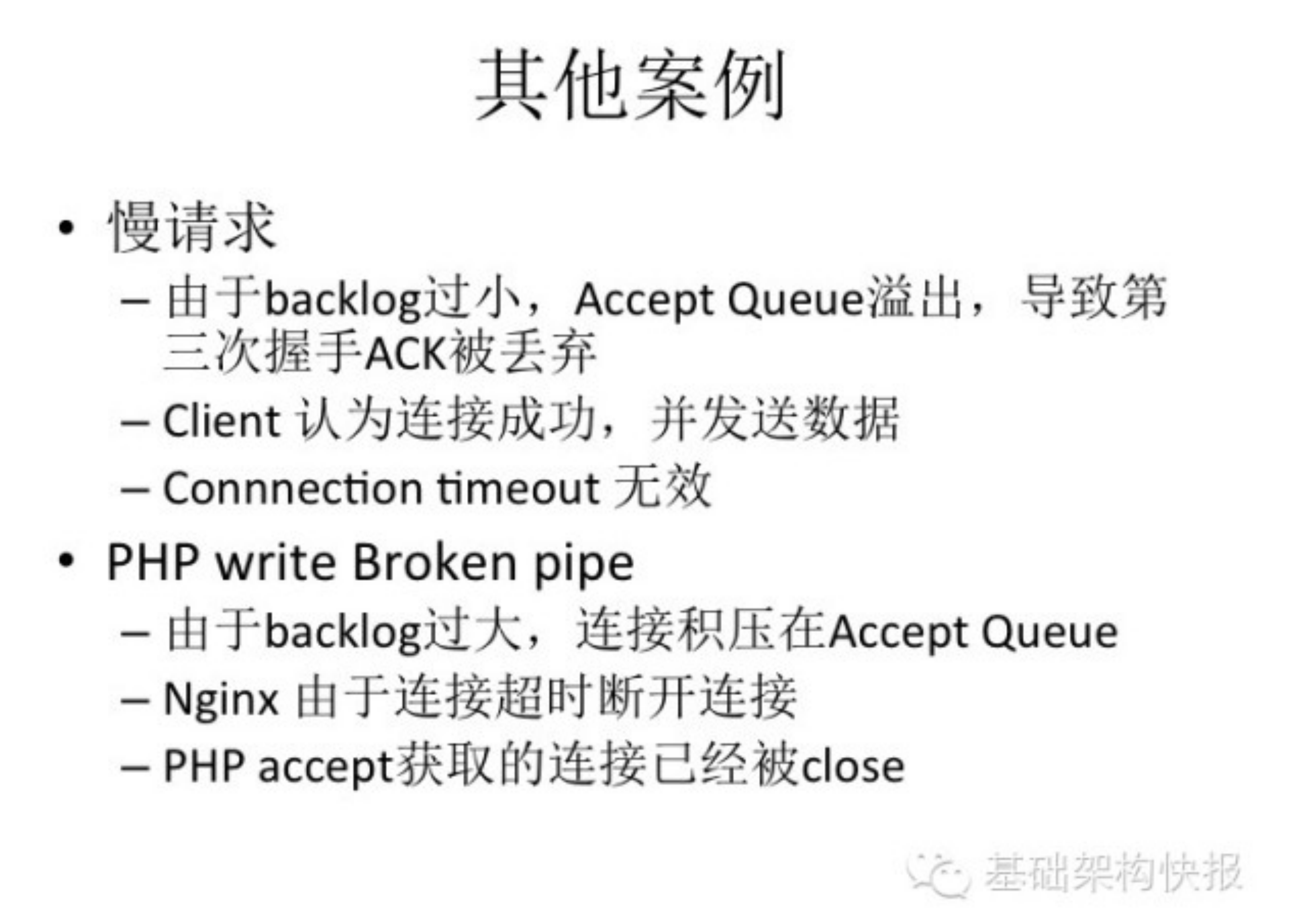
```
18:05:21.571051 IP 10.16.15.41.48006 > 10.123.87.46.8808: Flags [S], seq 3093965243, win 14600, options [m
length 0
18:05:21.626466 IP 10.123.87.46.8808 > 10.16.15.41.48006: Flags [S.], seq 2485902648, ack 3093965244, win
ecr 8351622,nop,wscale 7], length 0
18:05:21.626476 IP 10.16.15.41.48006 > 10.123.87.46.8808: Flags [.], ack 1, win 115, options [nop,nop,TS v
18:05:25.822802 IP 10.123.87.46.8808 > 10.16.15.41.48006: Flags [S.], seq 2485902648, ack 3093965244, win
ecr 8351622,nop,wscale 7], length 0
18:05:25.822873 IP 10.16.15.41.48006 > 10.123.87.46.8808: Flags [.], ack 1, win 115, options [nop,nop,TS v
length 0
```

- net.ipv4.tcp_synack_retries = 1
  - SYN+ACK只重传一次



结论

- 网络丢包导致三次握手最后一次失败
  - Server 端由于没有收到ACK，保持SYN_RECV状态
  - Client 端发送ACK后，变为ESTABLISHED状态
  - Client 端认为connect成功，因此调用recvmsg
- SYN+ACK只重传一次，如果重传的这次仍然有丢包，则导致客户端永久堵塞
- 这里的问题是网络丢包，如果Accept Queue溢出，会导致同样问题。



其他案例

- 慢请求
  - 由于backlog过小，Accept Queue溢出，导致第三次握手ACK被丢弃
  - Client 认为连接成功，并发送数据
  - Connnection timeout 无效
- PHP write Broken pipe
  - 由于backlog过大，连接积压在Accept Queue
  - Nginx 由于连接超时断开连接
  - PHP accept获取的连接已经被close

问题？

欢迎大家一起学习讨论

谢谢！

@shafreeck 2014-07-08