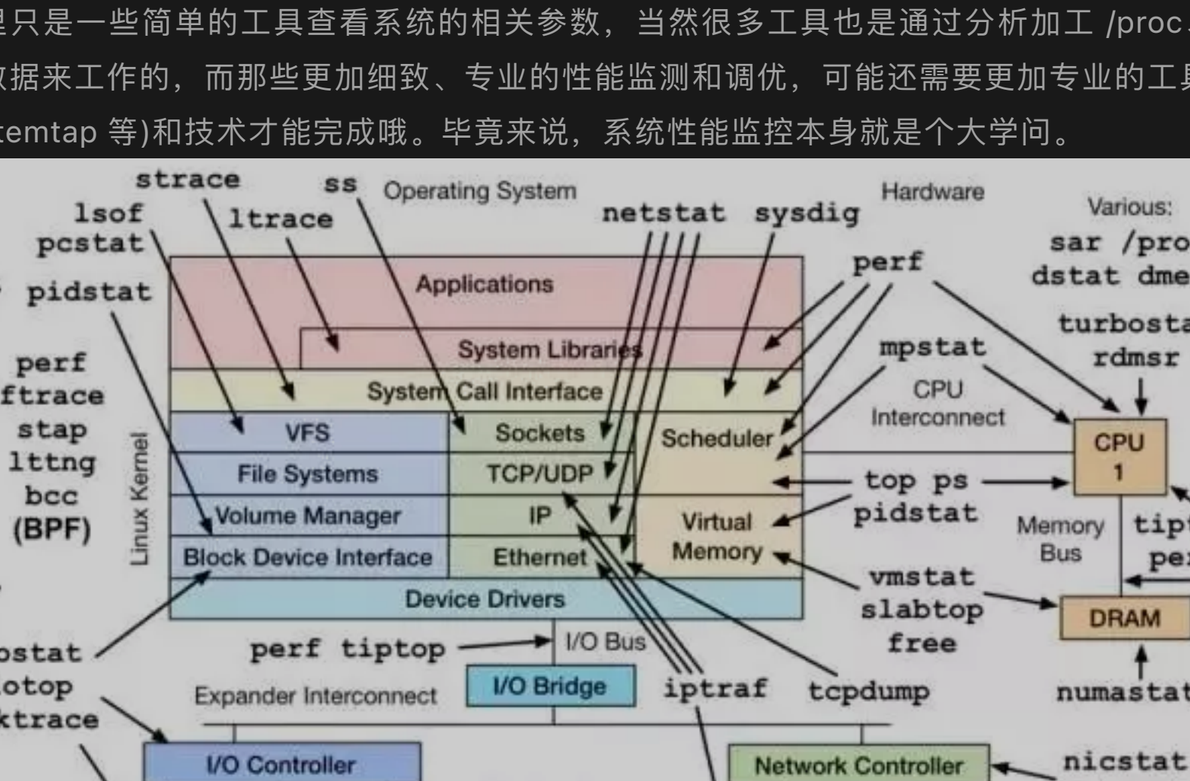
Linux 服务器的性能参数指标总结

程序员面试吧 昨天

一个基于 Linux 操作系统的服务器运行的同时，也会表征出各种各样参数信息。通常来说运维人员、系统管理员会对这些数据会极为敏感，但是这些参数对于开发者来说也十分重要，尤其当你的程序非正常工作的时候，这些蛛丝马迹往往会帮助快速定位跟踪问题。

这里只是一些简单的工具查看系统的相关参数，当然很多工具也是通过分析加工 /proc、/sys 下的数据来工作的，而那些更加细致、专业的性能监测和调优，可能还需要更加专业的工具(perf、systemtap 等)和技术才能完成哦。毕竟来说，系统性能监控本身就是个大学问。



一、CPU和内存类

1.1 top

➔ ~ top

```
top - 14:13:57 up 1 day, 21:01,  6 users,  load average: 3.13, 1.07, 0.41
Tasks: 245 total,   5 running, 240 sleeping,   0 stopped,   0 zombie
%Cpu(s): 91.7 us,  8.2 sy,  0.0 ni,  0.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 12077272 total,  4750852 free,  1729284 used,  5597136 buff/cache
KiB Swap:  7812092 total,  7812092 free,        0 used.  9996488 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
14796 taozj     20   0  233284  92316   5180 R  11.0  0.8   0:00.33 cc1plus
14800 taozj     20   0  226980  86248   5148 R   9.6  0.7   0:00.29 cc1plus
14805 taozj     20   0  216628  76228   5124 R   9.0  0.6   0:00.27 cc1plus
14808 taozj     20   0  197536  56392   4984 R   5.6  0.5   0:00.17 cc1plus
 1238 taozj     20   0  146268   2184   1424 R   0.7  0.0   0:00.10 top
   13 root      20   0       0      0      0 S   0.3  0.0   0:15.01 rcu_sched
   17 root      20   0       0      0      0 S   0.3  0.0   0:04.89 rcuos/3
 1051 root      20   0       0      0      0 S   0.3  0.0   0:00.51 kworker/2:0
28641 root      20   0       0      0      0 S   0.3  0.0   0:00.49 kworker/u8:0
29989 taozj     20   0  143348   2208    924 S   0.3  0.0   0:00.65 sshd
    1 root      20   0  188984   4096   2388 S   0.0  0.0   0:12.68 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.04 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   0:00.20 ksoftirqd/0
```

第一行后面的三个值是系统在之前 1、5、15 的平均负载，也可以看出系统负载是上升、平稳、下降的趋势，当这个值超过 CPU 可执行单元的数目，则表示 CPU 的性能已经饱和成为瓶颈了。

第二行统计了系统的任务状态信息。running 很自然不必多说，包括正在 CPU 上运行的和将要被调度运行的；sleeping 通常是等待事件(比如 IO 操作)完成的任务，细分可以包括 interruptible 和 uninterruptible 的类型；stopped 是一些被暂停的任务，通常发送 SIGSTOP 或者对一个前台任务操作 Ctrl-Z 可以将其暂停；zombie 僵尸任务，虽然进程终止资源会被自动回收，但是含有退出任务的 task descriptor 需要父进程访问后才能释放，这种进程显示为 defunct 状态，无论是因为父进程提前退出还是未 wait 调用，出现这种进程都应该格外注意程序是否设计有误。

第三行 CPU 占用率根据类型有以下几种情况：

√ (us) user：CPU 在低 nice 值(高优先级)用户态所占用的时间(nice<=0)。正常情况下只要服务器不是很闲，那么大部分的 CPU 时间应该都在此执行这类程序

√ (sy) system：CPU 处于内核态所占用的时间，操作系统通过系统调用(system call)从用户态陷入内核态，以执行特定的服务；通常情况下该值会比较小，但是当服务器执行的 IO 比较密集的时候，该值会比较大

√ (ni) nice：CPU 在高 nice 值(低优先级)用户态以低优先级运行占用的时间(nice>0)。默认新启动的进程 nice=0，是不会计入这里的，除非手动通过 renice 或者 setpriority() 的方式修改程序的nice值

√ (id) idle：CPU 在空闲状态(执行 kernel idle handler)所占用的时间

√ (wa) iowait：等待 IO 完成做占用的时间

√ (hi) irq：系统处理硬件中断所消耗的时间

√ (si) softirq：系统处理软中断所消耗的时间，记住软中断分为 softirqs、tasklets (其实是前者的特例)、work queues，不知道这里是统计的是哪些的时间，毕竟 work queues 的执行已经不是中断上下文了

√ (st) steal：在虚拟机情况下才有意义，因为虚拟机下 CPU 也是共享物理 CPU 的，所以这段时间表明虚拟机等待 hypervisor 调度 CPU 的时间，也意味着这段时间 hypervisor 将 CPU 调度给别的 CPU 执行，这个时段的 CPU 资源被“stolen”了。这个值在我 KVM 的 VPS 机器上是不为 0 的，但也只有 0.1 这个数量级，是不是可以用来判断 VPS 超售的情况？

CPU 占用率高很多情况下意味着一些东西，这也给服务器 CPU 使用率过高情况下指明了相应地排查思路：

√ 当 user 占用率过高的时候，通常是某些个别的进程占用了大量的 CPU，这时候很容易通过 top 找到该程序；此时如果怀疑程序异常，可以通过 perf 等思路找出热点调用函数来进一步排查；

√ 当 system 占用率过高的时候，如果 IO 操作(包括终端 IO)比较多，可能会造成这部分的 CPU 占用率高，比如在 file server、database server 等类型的服务器上，否则(比如>20%)很可能有些部分的内核、驱动模块有问题；

√ 当 nice 占用率过高的时候，通常是有意行为，当进程的发起者知道某些进程占用较高的 CPU，会设置其 nice 值确保不会淹没其他进程对 CPU 的使用请求；

√ 当 iowait 占用率过高的时候，通常意味着某些程序的 IO 操作效率很低，或者 IO 对应设备的性能很低以至于读写操作需要很长的时间来完成；

√ 当 irq/softirq 占用率过高的时候，很可能某些外设出现问题，导致产生大量的irq请求，这时候通过检查 /proc/interrupts 文件来深究问题所在；

√ 当 steal 占用率过高的时候，黑心厂商虚拟机超售了吧！

第四行和第五行是物理内存和虚拟内存(交换分区)的信息：

total = free + used + buff/cache，现在buffers和cached Mem信息总和到一起了，但是buffers和cached Mem 的关系很多地方都没说清楚。其实通过对比数据，这两个值就是 /proc/meminfo 中的 Buffers 和 Cached 字段：Buffers 是针对 raw disk 的块缓存，主要是以 raw block 的方式缓存文件系统的元数据(比如超级块信息等)，这个值一般比较小(20M左右)；而 Cached 是针对于某些具体的文件进行读缓存，以增加文件的访问效率而使用的，可以说是用于文件系统中文件缓存使用。

而 avail Mem 是一个新的参数值，用于指示在不进行交换的情况下，可以给新开启的程序多少内存空间，大致和 free + buff/cached 相当，而这也印证了上面的说法，free + buffers + cached Mem才是真正可用的物理内存。并且，使用交换分区不见得是坏事情，所以交换分区使用率不是什么严重的参数，但是频繁的 swap in/out 就不是好事情了，这种情况需要注意，通常表示物理内存紧缺的情况。

最后是每个程序的资源占用列表，其中 CPU 的使用率是所有 CPU core 占用率的总和。通常执行 top 的时候，本身该程序会大量的读取 /proc 操作，所以基本该 top 程序本身也会是名列前茅的。

top 虽然非常强大，但是通常用于控制台实时监测系统信息，不适合长时间(几天、几个月)监测系统的负载信息，同时对于短命的进程也会遗漏无法给出统计信息。

1.2 vmstat

vmstat 是除 top 之外另一个常用的系统检测工具，下面截图是我用-j4编译boost的系统负载。

```
▪ ~ vmstat --wide 1 --unit M
procs -----------------------memory---------------------- ---swap-- -----io---- -system-- --------cpu--------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy  id wa st
 4  0      0   3454      1   6238    0    0     3     9   15   34  0  0 100  0  0
 4  0      0   3406      1   6239    0    0     0  1476 4532 1169 94  6   0  0  0
 4  0      0   3795      1   6239    0    0     0  2108 4659 1705 92  8   0  0  0
 4  0      0   3763      1   6238    0    0     0     0 4463 1246 92  8   0  0  0
 4  0      0   4073      1   6239    0    0     0  5892 4794 2061 90 10   0  0  0
 4  0      0   3380      1   6238    0    0     0     0 4365  919 94  6   0  0  0
 4  0      0   3535      1   6244    0    0     0  3626 4688 1495 95  5   0  0  0
 5  0      0   3955      1   6244    0    0     0     0 4931 2084 92  8   0  0  0
 4  0      0   3315      1   6240    0    0     0     0 4996 2240 95  5   0  0  0
 4  0      0   3114      1   6242    0    0     0    24 4613 1412 95  5   0  0  0
```

r 表示可运行进程数目，数据大致相符；而b表示的是 uninterruptible 睡眠的进程数目；swpd 表示使用到的虚拟内存数量，跟 top-Swap-used 的数值是一个含义，而如手册所说，通常情况下 buffers 数目要比 cached Mem 小的多，buffers 一般20M这么个数量级；io 域的 bi、bo 表明每秒钟向磁盘接收和发送的块数目(blocks/s)；system 域的 in 表明每秒钟的系统中断数(包括时钟中断)，cs表明因为进程切换导致上下文切换的数目。

说到这里，想到以前很多人纠结编译 linux kernel 的时候 -j 参数究竟是 CPU Core 还是 CPU Core+1？通过上面修改 -j 参数值编译 boost 和 linux kernel 的同时开启 vmstat 监控，发现两种情况下 context switch 基本没有变化，且也只有显著增加 -j 值后 context switch 才会有显著的增加，看来不必过于纠结这个参数了，虽然具体编译时间长度我还没有测试。资料说如果不是在系统启动或者 benchmark 的状态，参数 context switch>100000 程序肯定有问题。

1.3 pidstat

如果想对某个进程进行全面具体的追踪，没有什么比 pidstat 更合适的了——栈空间、缺页情况、主被动切换等信息尽收眼底。这个命令最有用的参数是-t，可以将进程中各个线程的详细信息罗列出来。

-r：显示缺页错误和内存使用状况，缺页错误是程序需要访问映射在虚拟内存空间中但是还尚未被加载到物理内存中的一个分页，缺页错误两个主要类型是

√ minflt/s 指的 minor faults，当需要访问的物理页面因为某些原因(比如共享页面、缓存机制等)已经存在于物理内存中了，只是在当前进程的页表中没有引用，MMU 只需要设置对应的 entry 就可以了，这个代价是相当小的

√ majflt/s 指的 major faults，MMU 需要在当前可用物理内存中申请一块空闲的物理页面(如果没有可用的空闲页面，则需要将别的物理页面切换到交换空间去以释放得到空闲物理页面)，然后从外部加载数据到该物理页面中，并设置好对应的 entry，这个代价是相当高的，和前者有几个数据级的差异

-s：栈使用状况，包括 StkSize 为线程保留的栈空间，以及 StkRef 实际使用的栈空间。使用ulimit -s发现CentOS 6.x上面默认栈空间是10240K，而 CentOS 7.x、Ubuntu系列默认栈空间大小为8196K

```
▪ ~ pidstat -s -t -p 18117
Linux 3.10.0-229.el7.x86_64 (localhost.localdomain)     01/04/2017

02:27:00 PM   UID      TGID       TID StkSize  StkRef  Command
02:27:00 PM  1007     18117         -     136      16  ailawd
02:27:00 PM  1007         -     18117     136      16  |__ailawd
02:27:00 PM  1007         -     18125    8192       8  |__ailawd
02:27:00 PM  1007         -     18126    8192      12  |__ailawd
02:27:00 PM  1007         -     18127    8192      12  |__ailawd
```

-u：CPU使用率情况，参数同前面类似

-w：线程上下文切换的数目，还细分为cswch/s因为等待资源等因素导致的主动切换，以及nvcswch/s线程CPU时间导致的被动切换的统计

如果每次都先ps得到程序的pid后再操作pidstat会显得很麻烦，所以这个杀手锏的-C可以指定某个字符串，然后Command中如果包含这个字符串，那么该程序的信息就会被打印统计出来，-l可以显示完整的程序名和参数

➔ ~ pidstat -w -t -C “ailaw” -l

这么看来，如果查看单个尤其是多线程的任务时候，pidstat比常用的ps更好使！

1.4 其他

当需要单独监测单个 CPU 情况的时候，除了 htop 还可以使用 mpstat，查看在 SMP 处理器上各个 Core 的工作量是否负载均衡，是否有某些热点线程占用 Core。

➔ ~ mpstat -P ALL 1

如果想直接监测某个进程占用的资源，既可以使用top -u taozj的方式过滤掉其他用户无关进程，也可以采用下面的方式进行选择，ps命令可以自定义需要打印的条目信息：

> while :; do ps -eo user,pid,ni,pri,pcpu,psr,comm | grep 'ailawd'; sleep 1; done

如想理清继承关系，下面一个常用的参数可以用于显示进程树结构，显示效果比pstree详细美观的多

➔ ~ ps axjf

二、磁盘IO类

iotop 可以直观的显示各个进程、线程的磁盘读取实时速率；lsof 不仅可以显示普通文件的打开信息(使用者)，还可以操作 /dev/sda1 这类设备文件的打开信息，那么比如当分区无法 umount 的时候，就可以通过 lsof 找出磁盘该分区的使用状态了，而且添加 +fg 参数还可以额外显示文件打开 flag 标记。

2.1 iostat

➔ ~ iostat -xz 1

其实无论使用 iostat -xz 1 还是使用 sar -d 1，对于磁盘重要的参数是：

√ avgqu-s：发送给设备 I/O 请求的等待队列平均长度，对于单个磁盘如果值>1表明设备饱和，对于多个磁盘阵列的逻辑磁盘情况除外

√ await(r_await、w_await)：平均每次设备 I/O 请求操作的等待时间(ms)，包含请求排列在队列中和被服务的时间之和；

√ svctm：发送给设备 I/O 请求的平均服务时间(ms)，如果 svctm 与 await 很接近，表示几乎没有 I/O 等待，磁盘性能很好，否则磁盘队列等待时间较长，磁盘响应较差；

√ %util：设备的使用率，表明每秒中用于 I/O 工作时间的占比，单个磁盘当 %util>60% 的时候性能就会下降(体现在 await 也会增加)，当接近100%时候就设备饱和了，但对于有多个磁盘阵列的逻辑磁盘情况除外；

还有，虽然监测到的磁盘性能比较差，但是不一定会对应用程序的响应造成影响，内核通常使用 I/O asynchronously 技术，使用读写缓存技术来改善性能，不过这又跟上面的物理内存的限制相制约了。

上面的这些参数，对网络文件系统也是受用的。

三、网络类

网络性能对于服务器的重要性不言而喻，工具 iptraf 可以直观的现实网卡的收发速度信息，比较的简洁方便通过 sar -n DEV 1 也可以得到类似的吞吐量信息，而网卡都标配了最大速率信息，比如百兆网卡千兆网卡，很容易查看设备的利用率。

通常，网卡的传输速率并不是网络开发中最为关切的，而是针对特定的 UDP、TCP 连接的丢包率、重传率，以及网络延时等信息。

3.1 netstat

➔ ~ netstat -s

显示自从系统启动以来，各个协议的总体数据信息。虽然参数信息比较丰富有用，但是累计值，除非两次运行做差才能得出当前系统的网络状态信息，亦或者使用 watch 眼睛直观其数值变化趋势。所以netstat通常用来检测端口和连接信息的：

> netstat –all(a) –numeric(n) –tcp(t) –udp(u) –timers(o) –listening(l) –program(p)

–timers可以取消域名反向查询，加快显示速度；比较常用的有

➔ ~ netstat -antp #列出所有TCP的连接

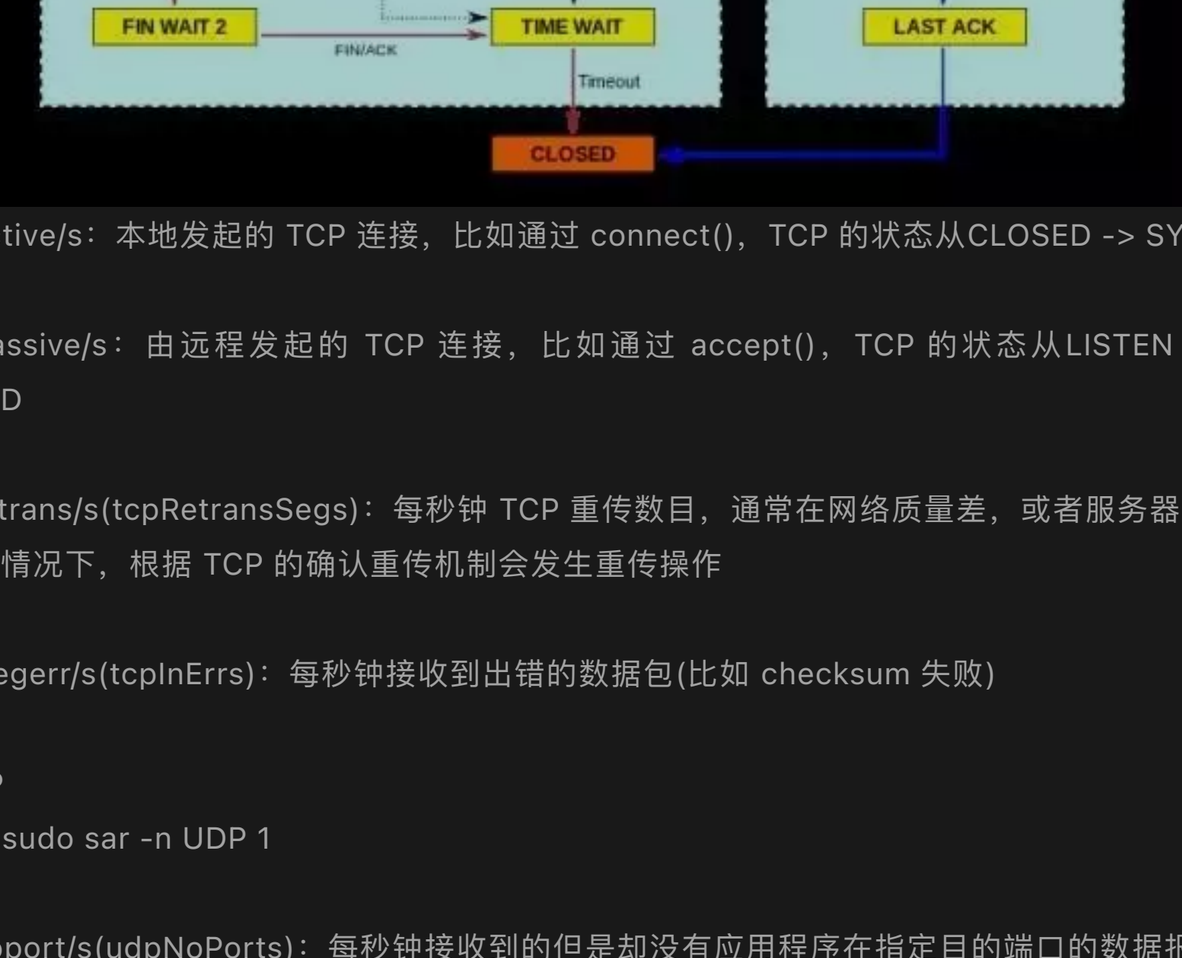
➔ ~ netstat -nltp #列出本地所有TCP侦听套接字，不要加-a参数

3.2 sar

sar 这个工具太强大了，什么 CPU、磁盘、页面交换啥都管，这里使用 -n 主要用来分析网络活动，虽然网络中它还给细分了 NFS、IP、ICMP、SOCK 等各种层次各种协议的数据信息，我们只关心 TCP 和 UDP。下面的命令除了显示常规情况下段、数据报的收发情况，还包括

TCP

➔ ~ sudo sar -n TCP,ETCP 1



√ active/s：本地发起的 TCP 连接，比如通过 connect()，TCP 的状态从CLOSED -> SYN-SENT

√ passive/s：由远程发起的 TCP 连接，比如通过 accept()，TCP 的状态从LISTEN -> SYN-RCVD

√ retrans/s(tcpRetransSegs)：每秒钟 TCP 重传数目，通常在网络质量差，或者服务器过载后丢包的情况下，根据 TCP 的确认重传机制会发生重传操作

√ isegerr/s(tcpInErrs)：每秒钟接收到出错的数据包(比如 checksum 失败)

UDP

➔ ~ sudo sar -n UDP 1

√ noport/s(udpNoPorts)：每秒钟接收到的但是却没有应用程序在指定目的端口的数据报个数

√ idgmerr/s(udpInErrors)：除了上面原因之外的本机接收到但却无法派发的数据报个数

当然，这些数据一定程度上可以说明网络可靠性，但也只有同具体的业务需求场景结合起来才具有意义。

3.3 tcpdump

tcpdump 不得不说是个好东西。大家都知道本地调试的时候喜欢使用 wireshark，但是线上服务端出现问题怎么弄呢？

附录的参考文献给出了思路：复原环境，使用 tcpdump 进行抓包，当问题复现(比如日志显示或者某个状态显现)的时候，就可以结束抓包了，而且 tcpdump 本身带有 -C/-W 参数，可以限制抓取包存储文件的大小，当达到这个这个限制的时候保存的包数据自动 rotate，所以抓包数量总体还是可控的。此后将数据包拿下线来，用 wireshark 想怎么看就怎么看，岂不乐哉！tcpdump 虽然没有 GUI 界面，但是抓包的功能丝毫不弱，可以指定网卡、主机、端口、协议等各项过滤参数，抓下来的包完整又带有时间戳，所以线上程序的数据包分析也可以这么简单。

下面就是一个小的测试，可见 Chrome 启动时候自动向 Webserver 发起建立了三条连接，由于这里限制了 dst port 参数，所以服务端的应答包被过滤掉了，拿下来用 wireshark 打开，SYNC、ACK 建立连接的过程还是很明显的！在使用 tcpdump 的时候，需要尽可能的配置抓取的过滤条件，一方面便于接下来的分析，二则 tcpdump 开启后对网卡和系统的性能会有影响，进而会影响到在线业务的性能。

文章转自：twt企业IT社区