

第二节：tcpdump

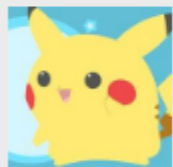
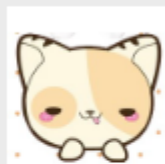
小对话

< 微信

test

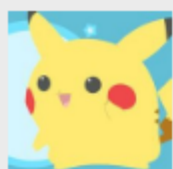
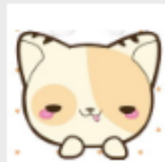


服务端怎么这么不稳定，是不是挂了？



没有呀，数据都发送出去了。

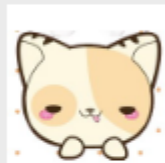
你怎么保证你发送成功了？

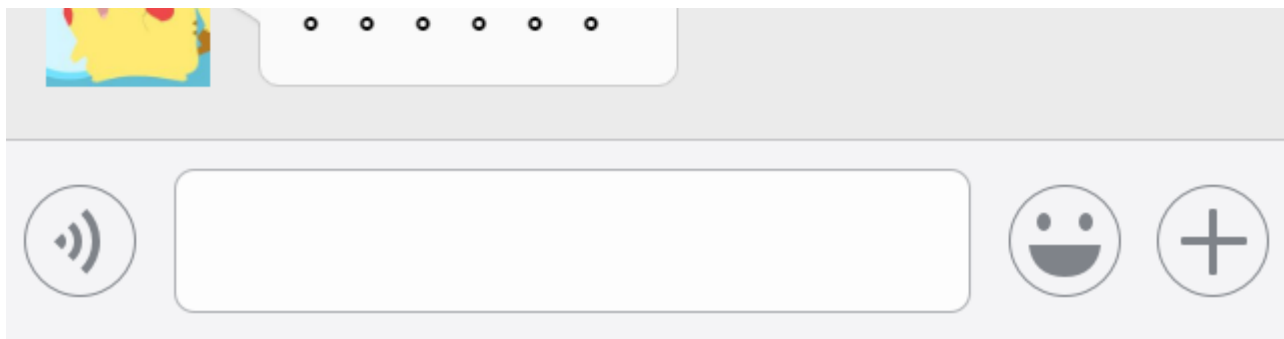


我利用系统底层网络发送的。
没有报错，这还怎么证明。



反正我没收到，我下班了，你赶紧排查。

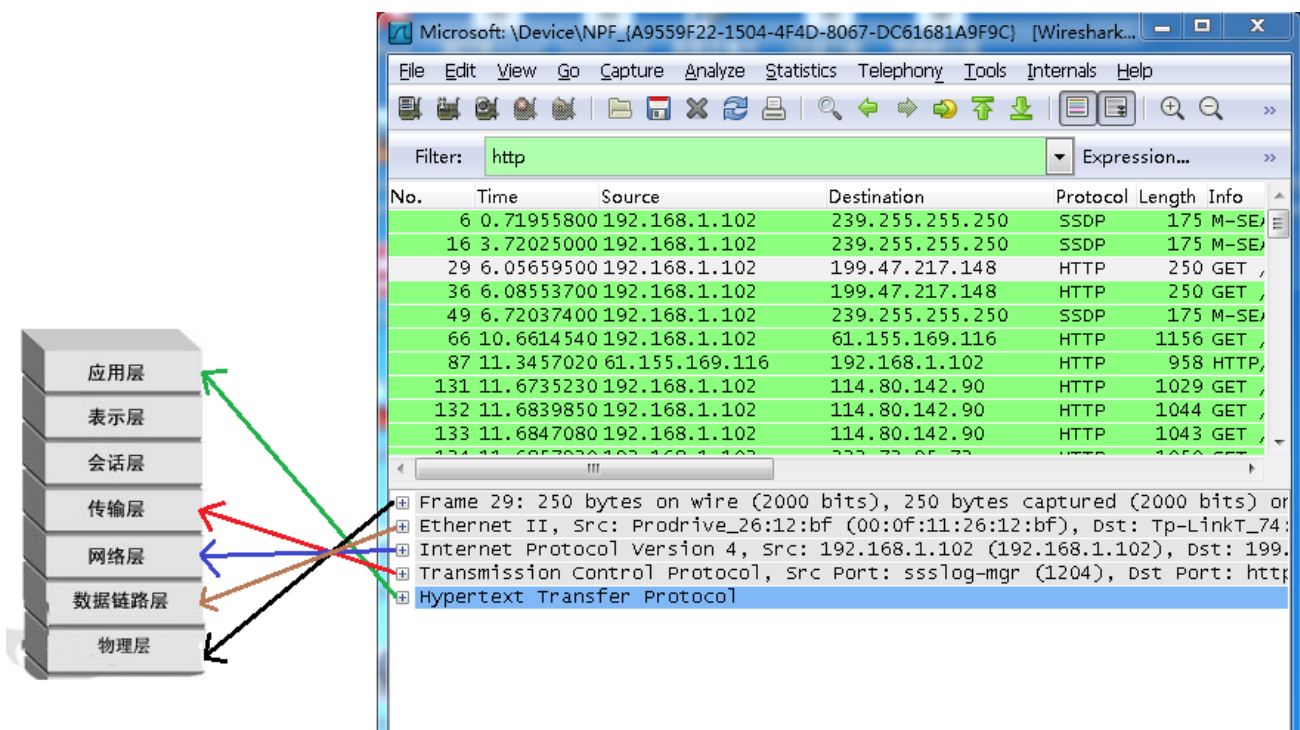




tcpdump 就是证据

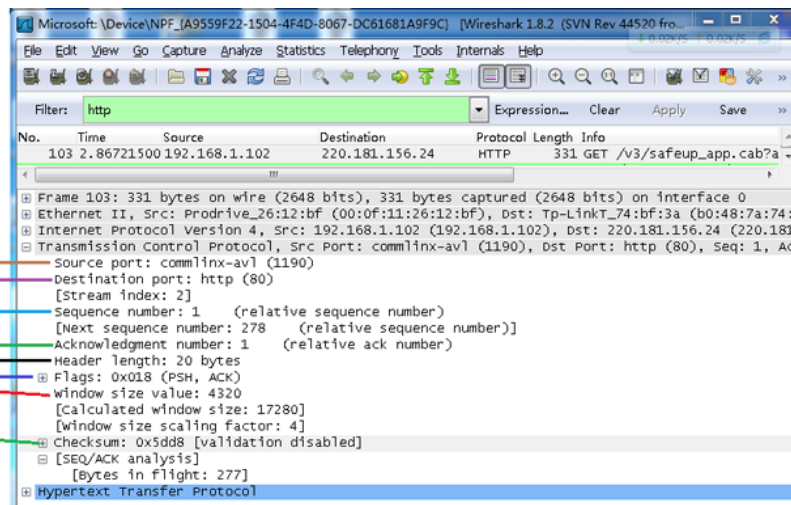
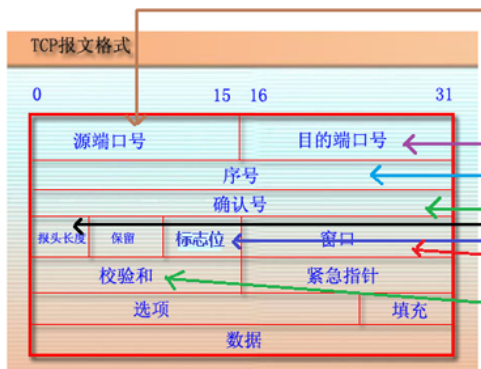
tcpdump - dump traffic on a network

```
tcpdump -i eth0 '(tcp port 80)' -v -w dump.pcap (服务器nginx 模拟一次请求)  
sz dump.pcap
```



网络层负责ip数据报的产生以及ip数据包在逻辑网络上的路由转发(选择哪个路径)

传输层提供端到端通信服务层次,提供可靠及非可靠连接(保证路径上传输可靠 tcp)



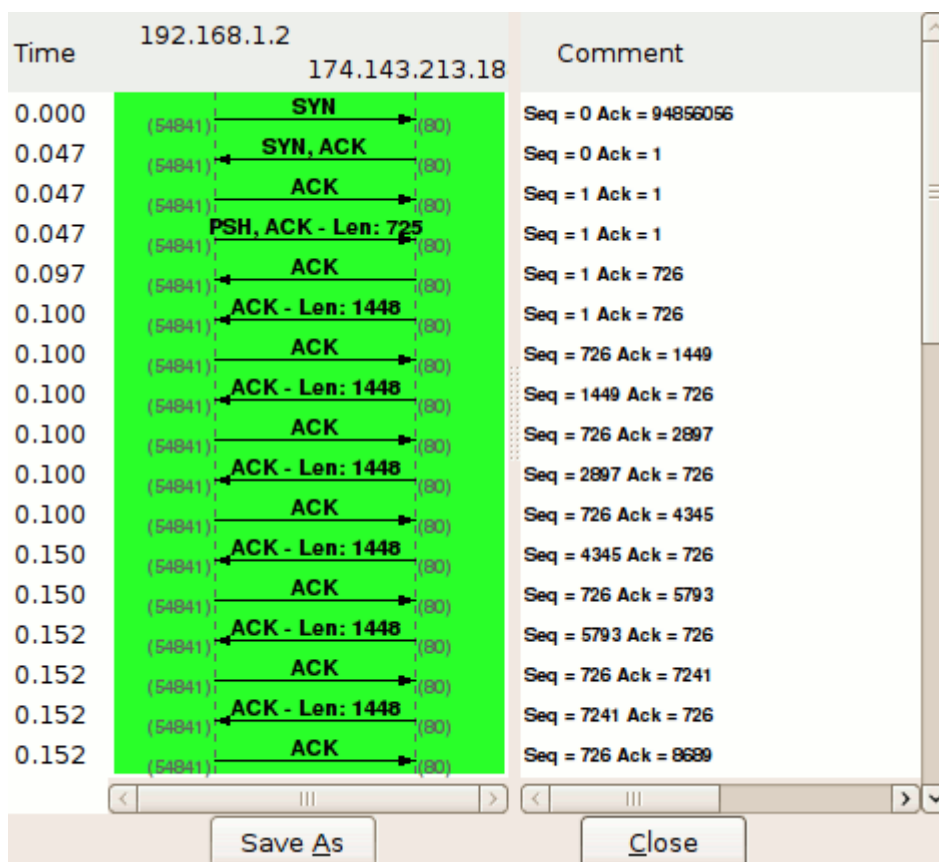
- TCP序列号 (Sequence Number) 和确认号 (Acknowledgment Number)

Next sequence number:=Sequence number+len(data) Sequence number=Next sequence number

Acknowledgment number: Sequence number

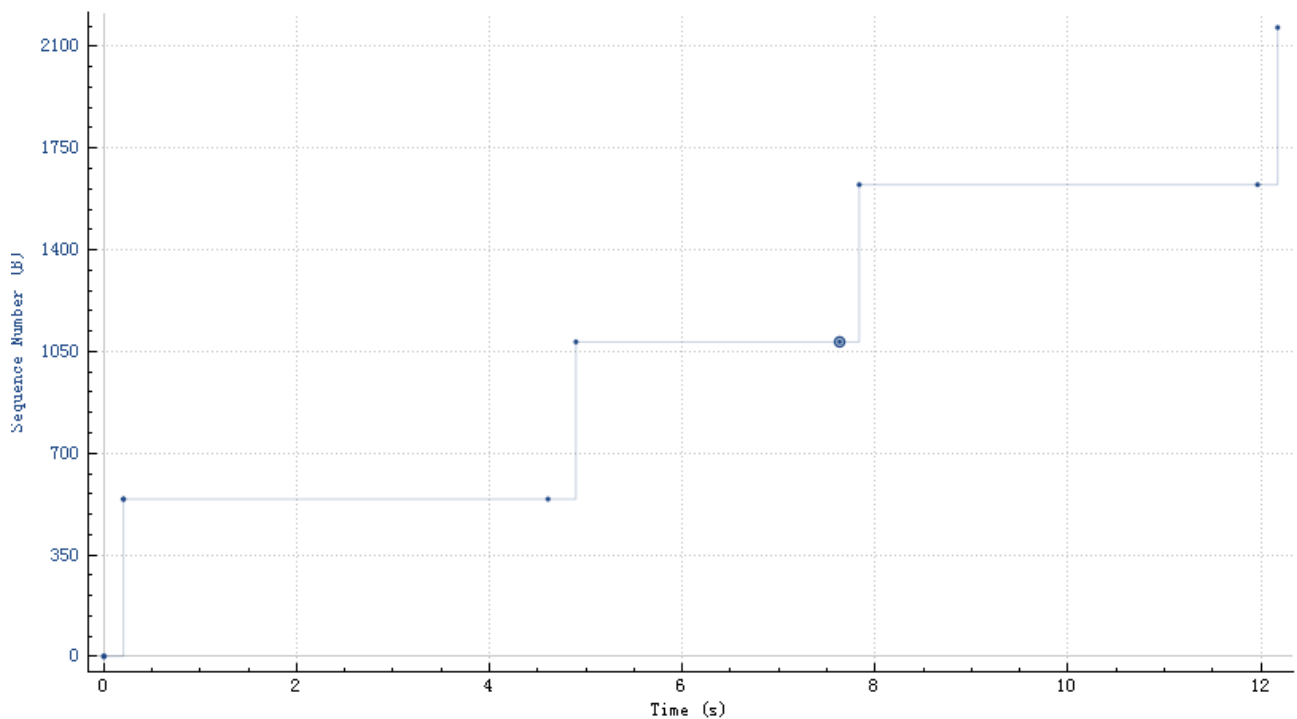
为了更好的理解在整个TCP会话期间，TCP序列号和确认号是如何工作的，我们可以使用Wireshark内置的绘制流功能，选择菜单栏中的 **Statistics -> Flow Graph... -> TCP flow -> OK**

- 这个是老版本的



序列号 (Stevens) 对于 ~~10.0.0.1:54433 → 10.0.0.1:80~~

dump.pcap



点击选取分组 13 (7.632s len 541 seq 1083 ack 361 win 66048) → 11 分组, 2164 bytes → 7 分组, 900 bytes

• 数据传输

[TCP Segment Len: 541]

Sequence number: 542 (relative sequence number)

[Next sequence number: 1083 (relative sequence number)]

Acknowledgment number: 181 (relative ack number)

Sequence number: 181 (relative sequence number)

[Next sequence number: 361 (relative sequence number)]

Acknowledgment number: 1083 (relative ack number)

看序列号完整行

• 连接

Time	Delta	Source	Destination	Length	Protocol	Info
24 5.069188	0.000000	10.0.0.145	157.166.238.17	66	TCP	54433 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS.
27 5.143961	0.074773	157.166.238.17	10.0.0.145	66	TCP	80 → 54433 [SYN, ACK] Seq=0 Ack=1 Win=4380 Len=0
28 5.144071	0.000110	10.0.0.145	157.166.238.17	54	TCP	54433 → 80 [ACK] Seq=1 Ack=1 Win=17520 Len=0
29 5.144274	0.000203	10.0.0.145	157.166.238.17	1022	HTTP	GET / HTTP/1.1

三次握手

• 释放

四次握手

参考

1. <https://danielmiessler.com/study/tcpdump/>
2. [Understanding TCP Sequence and Acknowledgment Numbers](#)
3. https://www.youtube.com/watch?v=AX2D_n1yZko
4. How TCP Works - The Handshake <https://www.youtube.com/watch?v=HCHFX5O1laQ>
5. <https://blog.csdn.net/a19881029/article/details/38091243>

-----【未完待续 这是最基本的使用】-----

扩展阅读

strace - trace system calls and signals

```
strace -o output.txt -T -tt -e trace=all -p 6107(nginx)
```

[10:16:34.228468 -tt 在输出中的每一行前加上时间信息,微秒级]

[writev(3, [{"HTTP/1.1 304 Not Modified\r\nServe"...}, 180]], 1) = 180 -e trace=all

只跟踪指定的系统调用.例如:-e trace=open,close,read,write表示只跟踪这四个系统调用.

默认的为set=all.

-e trace=file

只跟踪有关文件操作的系统调用.

-e trace=process

只跟踪有关进程控制的系统调用.

-e trace=network

跟踪与网络有关的所有系统调用.

-e strace=signal

跟踪所有与系统信号有关的系统调用]

[<0.000041> -T 显示每一调用所耗的时间.]

```
10:17:14.230606 epoll_wait(8, {{EPOLLIN, {u32=1455525904, u64=140631569698832}}}, 512, -1) = 1 <252.363683>
10:21:26.594416 gettimeofday({1546914086, 594452}, NULL) = 0 <0.000022>
10:21:26.594555 accept4(6, {sa_family=AF_INET, sin_port=htons(63951), sin_addr=inet_addr("10.2.31.67")}, [16], SOCK_NONBLOCK) = 3 <0.000029>
10:21:26.594726 epoll_ctl(8, EPOLL_CTL_ADD, 3, {EPOLLIN|EPOLLRDHUP|EPOLLET, {u32=1455526353, u64=140631569699281}}) = 0 <0.000011>
10:21:26.594776 epoll_wait(8, {{EPOLLIN, {u32=1455526353, u64=140631569699281}}}, 512, 60000) = 1 <0.000007>
10:21:26.594801 gettimeofday({1546914086, 594808}, NULL) = 0 <0.000005>
10:21:26.594829 recvfrom(3, "GET / HTTP/1.1\r\nHost: 10.112.178"... 1024, 0, NULL, NULL) = 541 <0.000009>
10:21:26.594937 stat("/usr/local/nginx/html/index.html", {st_mode=S_IFREG|0644, st_size=612, ...}) = 0 <0.000036>
10:21:26.595035 open("/usr/local/nginx/html/index.html", O_RDONLY|O_NONBLOCK) = 10 <0.000024>
10:21:26.595091 fstat(10, {st_mode=S_IFREG|0644, st_size=612, ...}) = 0 <0.000020>
10:21:26.595173 writev(3, [{"HTTP/1.1 304 Not Modified\r\nServe"...}, 180]], 1) = 180 <0.000038>
10:21:26.595260 write(4, "10.2.31.67 - - [08/Jun/2019:10:2"... 187) = 187 <0.000065>
10:21:26.595369 close(10) = 0 <0.000022>
```

Fiddler