如何才能真实模拟出来我们之前所学的三次握手和四次挥手的过程呢？**本文通过packetdrill工具来实现三次握手、数据传输、四次挥手整个过程的模拟。**

本篇有点难度，只要能大概看懂packetdrill语法即可，不必太纠结于某一行脚本的含义，重要的还是对三次握手、四次挥手过程的消化和理解。

一、packetdrill概述

我们知道，网络协议是现代计算机系统的一个重要组成部分，不过网络协议本身十分庞大复杂，比如 TCP 的 roadmap RFC 包含了 32 个其它的 RFC 文档，而 Linux 实现了其中的大多数特性。不过新的事物仍然在涌现，使得TCP等越来越复杂，测试起来也十分麻烦，针对这个情况，Google开发了packetdrill，packetdrill 是一个跨平台的脚本工具，可以用来测试整个 TCP/UDP/IP 网络栈实现的正确性和性能，从系统调用一直到硬件网络接口，从 IPv4 到 IPv6。

二、packetdrill安装

本文测试机器系统版本：CentOS Linux release 7.8.2003 (Core)

本文测试机器内核版本：Linux VM-0-13-centos 3.10.0-514.21.1.el7.x86_64 #1 SMP Thu May 25 17:04:51 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux

- 1、下载源码： https://github.com/google/packetdrill
- 2、切换为root，并进入源码目录：cd gtests/net/packetdrill
- 3、安装bison和flex库用于构建词法和语法分析器：yum install -y bison flex
- 4、为避免 offload 机制对包大小的影响，修改 netdev.c 注释掉 set_device_offload_flags 函数所有内容

```
/* Set the offload flags to be like a typical ethernet device */
static void set_device_offload_flags(struct local_netdev *netdev)
{
//#ifdef linux
//      const u32 offload =
//          TUN_F_CSUM | TUN_F_TSO4 | TUN_F_TSO6 | TUN_F_TSO_ECN;
//      if (ioctl(netdev->tun_fd, TUNSETOFFLOAD, offload) != 0)
//              die_perror("TUNSETOFFLOAD");
//endif
}
```

- 5、执行./configure
- 6、修改Makefile，去掉第一行的末尾的-static
- 7、执行make命令编译

- 8、确认编译无误地生成了 packetdrill 可执行文件

```
-rwxr-xr-x 1 root root 735040 Oct  7 14:07 packetdrill
-rw-r--r-- 1 root root  12208 Oct  7 14:07 checksum_test.o
-rwxr-xr-x 1 root root 734536 Oct  7 14:07 checksum_test
-rw-r--r-- 1 root root  39064 Oct  7 14:07 packet_parser_test.o
-rwxr-xr-x 1 root root 746720 Oct  7 14:07 packet_parser_test
-rw-r--r-- 1 root root  24800 Oct  7 14:07 packet_to_string_test.o
-rwxr-xr-x 1 root root 740752 Oct  7 14:07 packet_to_string_test
[root@VM-0-13-centos packetdrill]# ./packetdrill
error: missing script path
Usage: packetdrill
        [--ip_version=[ipv4,ipv4-mapped-ipv6,ipv6]]
        [--bind_port=bind_port]
        [--code_command=code_command]
        [--code_format=code_format]
        [--code_sockopt=TCP_INFO]
```

- 9、使用方法：./packetdrill test.pkt
- 10、添加到环境变量中：
  - vim /etc/profile
  - 在最后一行添加：export PATH=$PATH:/root/swg/packetdrill-master/gtests/net/packetdrill（按照你的实际目录写）
  - 保存退出，执行source /etc/profile使之生效
  - 在其他任意目录执行packetdrill命令

> test.pkt为按Packetdrill语法编写的测试脚本，packetdrill 脚本采用 c 语言和 tcpdump 混合的语法。脚本文件名一般以 .pkt 为后缀，成功：无输出，表示脚本正确，一切都符合预期；失败：指出脚本的错误地方，以及原因。

三、小试牛刀

利用此工具模拟三次握手、数据传输和四次挥手，整体脚本如下：

```
//1、启动服务端，等待客户端发起连接
0   socket(..., SOCK_STREAM, IPPROTO_TCP) = 3
+0  bind(3, ..., ...) = 0
+0  listen(3, 1) = 0

//2、三次握手
+0  < S 0:0(0) win 4000 <mss 1000>
+0  > S. 0:0(0) ack 1 <...>
+.1 < . 1:1(0) ack 1 win 1000
+0 accept(3, ..., ...) = 4

//3、服务端发送给客户端10字节数据包
+0.2 write(4,...,10) = 10
```

```
+0.0 > P. 1:11(10) ack 1
+0.0 < . 1:1(0) ack 11 win 1000

//4、客户端主动断开，四次挥手
+0 < F. 1:1(0) ack 11 win 1000
+0 > .11:11(0) ack 2
+0.1 close(4)=0
+0 > F. 11:11(0) ack 2 <...>
+0.01 < . 2:2(0) ack 12 win 1000

//5、休息10秒，打印finish
+0 `sleep 10`
+0 `echo finish`
```

3.1、服务端启动

```
0 socket(..., SOCK_STREAM, IPPROTO_TCP) = 3
```

表示第0秒，执行系统调用socket，在 packetdrill 脚本中用 ... 来表示程序的默认值。第一个0表示绝对时间0秒，数字前面带+号则表示是相对时间。

第一个参数表示协议族使用AF_INET （IPV4)还是AF_INET6(IPV6)，一般选择IPv4；第二个参数表示套接字传输类型，TCP是基于数据流的则设置为SOCK_STREAM，UDP是基于数据报的则设置为SOCK_DGRAM；第三个参数是协议，默认是TCP协议。

最后的=3有点令人费解，其中=表示断言，断言的意思是期望得到某个结果，否则就应该报错，这是程序开发和调试阶段的一种调试方法。

其次就是3，实际上是脚本返回新建的socket文件句柄，实际上更加准确地说是文件描述符，在linux中， 每当进程打开一个文件时（还记得linux中一切皆文件的基本哲学思想吗），系统就为其分配一个唯一对应的整型文件描述符（从0开始），用来标识这个文件。这里断言句柄为3，是因为linux 在每个程序开始的时刻，都会有 3 个已经打开的文件句柄，分别是：标准输入stdin(0)、标准输出stdout(1)、错误输出stderr(2) 默认的，其它新建的文件句柄则排在之后，从 3 开始。

```
+0 bind(3, ..., ...) = 0
+0 listen(3, 1) = 0
```

首先调用bind函数，这里的socket地址省略会使用默认的端口8080，第一个参数3是套接字的fd，即上面断言获取到的3。这句话执行后，服务端就会占用8080端口启动。

调用listen函数，服务端进入监听状态，第一个参数3也是套接字fd，到此为止，socket已经可以接受客户端的tcp连接了。

3.2、三次握手

```
+0 < S 0:0(0) win 4000 <mss 1000>
+0 > S. 0:0(0) ack 1 <...>
+.1 < . 1:1(0) ack 1 win 1000
+0 accept(3, ..., ...) = 4
```

这里四行就是三次握手过程。

第一行的 < 表示输入的数据包（input packets），packetdrill会构造一个真实的数据包注入到内核协议栈。

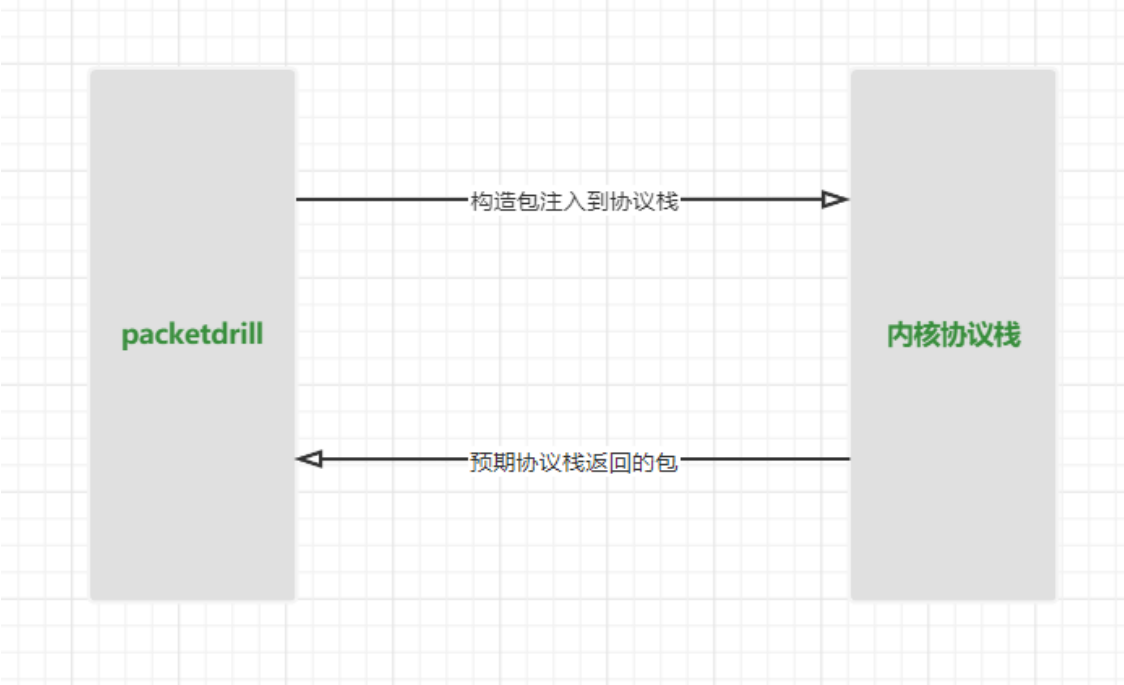
S表示为SYN，S.表示SYN+ACK，具体如下：

缩写	全称	描述
S	SYN	发起连接请求报文
S.	SYN+ACK	连接请求确认报文
F	FIN	连接释放报文
F.	FIN+ACK	连接释放确认报文
R	RST	连接复位
P	PSH	尽快将数据送给上层应用

0:0(0) 三个0分别表示发送包的起始 seq、结束 seq、包的长度。由于我们是一个SYN报文，没有携带任何数据，所以数据报长度为0。此外，还需要携带win和mss两个字段，发起SYN方给出自己的接收窗口win为4000，MSS即最大段大小设置为了1000.

第二行的 > 表示预期协议栈会响应的包（outbound packets），S.表示预期返回 SYN+ACK 包，数据包长度为0.（注意，这个包不是 packetdrill 构造的，是由协议栈发出的，packetdrill 会检查协议栈是不是真的发出了这个包，如果没有，则脚本报错停止执行。）

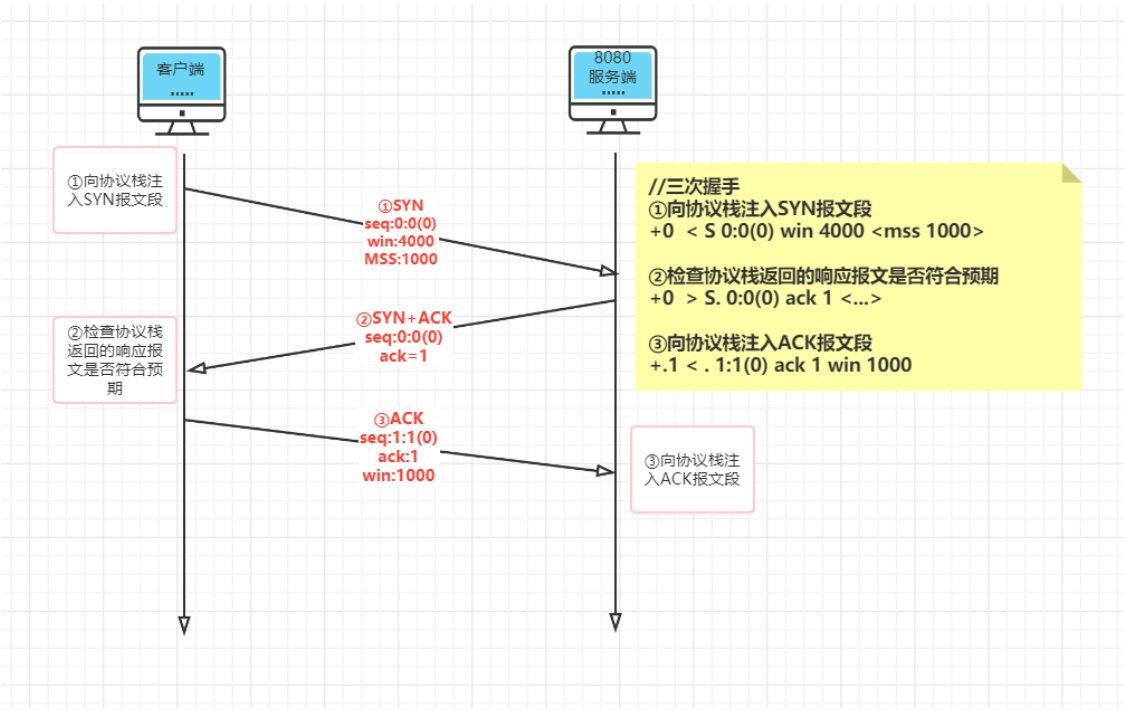
第一行是packetdrill构造注入到内核协议栈的，第二行是预期内核协议栈的响应值，他两做了一个互动：



第三行，在0.1秒后向协议栈中压入模拟的报文，完成三次握手。我们之前说过，SYN报文是需要得到确认的，所以要占用一个序列号，因此最后一次数据包的序列号是1，数据报长度为0，这是一个普通不含有数据的ack报文段。

第四行，三方握手完成后，服务器调用accept()接受连接，如果服务器调用accept()时还没有客户端的连接请求，就阻塞等待直到有客户端连接上来。accept函数成功返回一个新的socket文件描述符4，用于和客户端通信，下面写数据就会用到这个新的文件描述符。

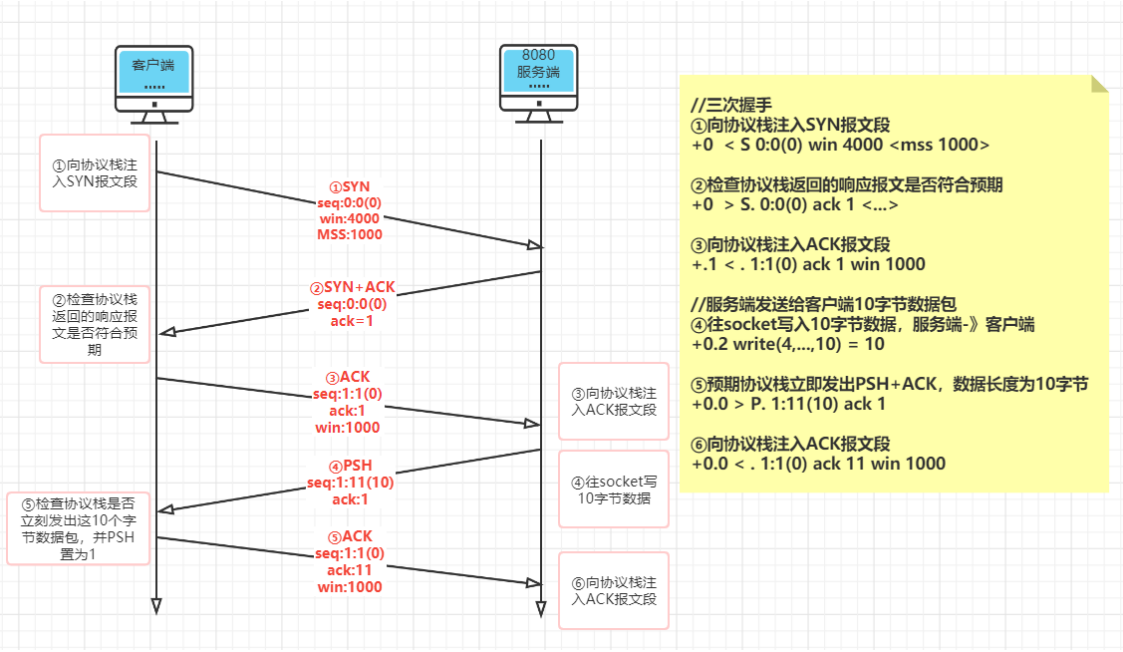
至此完成三次握手。以上整体完成的工作如图所示：



3.3、数据传输

```
//服务端发送给客户端10字节数据包
+0.2 write(4,...,10) = 10
+0.0 > P. 1:11(10) ack 1
+0.0 < . 1:1(0) ack 11 win 1000
```

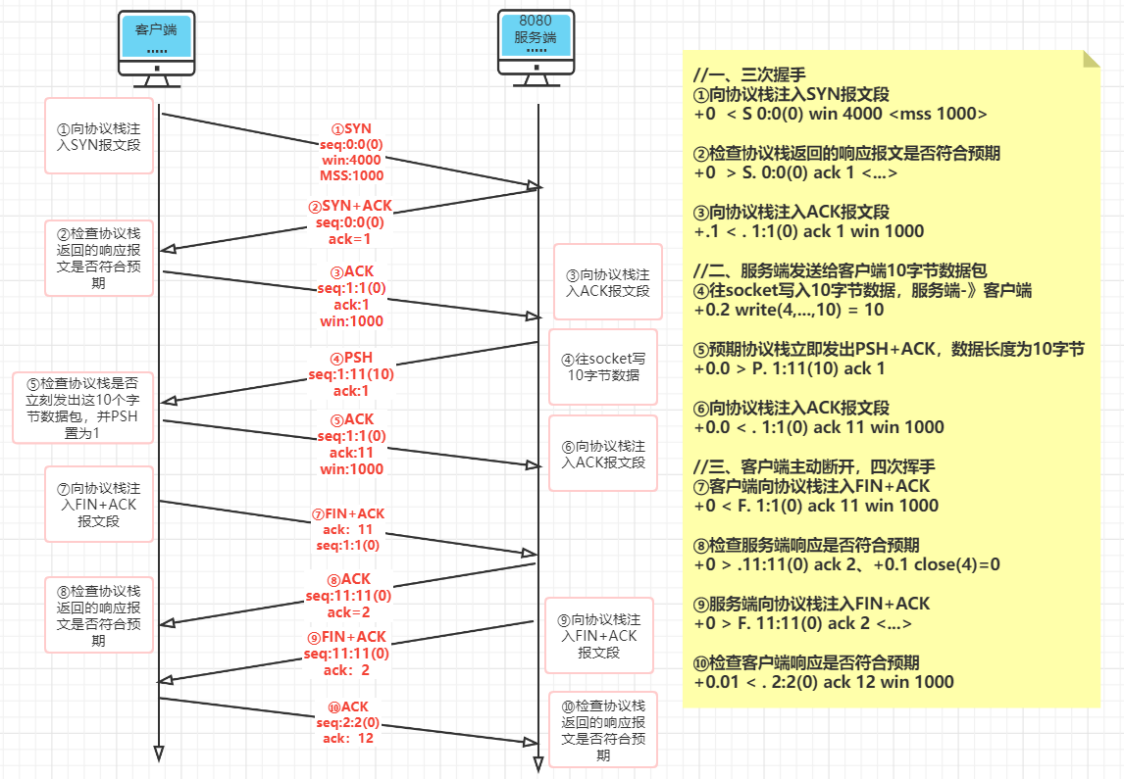
服务端写了10字节的数据，预期协议栈会发出这10字节数据包，由服务端发给客户端；最后一行是客户端收到10个数据包后，需要给协议栈注入ack报文告诉服务端已收到这10字节数据。



3.4、四次挥手

```
//客户端主动断开，四次挥手
+0 < F. 1:1(0) ack 11 win 1000
+0 > .11:11(0) ack 2
+0.1 close(4)=0
+0 > F. 11:11(0) ack 2 <...>
+0.01 < . 2:2(0) ack 12 win 1000
```

客户端主动断开，四次挥手结束连接，第一行为FIN类型报文，模拟客户端主动断开连接，第二行为预期想得到的服务端应答报文，表示知道了客户端断开连接的请求，第三行为服务端执行系统调用close，关闭当前文件句柄，结束与远端socket的连接，第四行和第五行为服务端通知客户端结束连接。



3.5、最后的两行

```
+0 sleep 10
+0 echo finish
```

一个是休眠10秒，一个是打印结束。这是方便调试的，下面我们会进行一个效果的验证。

四、tcpdump抓包

在启动脚本前，我们要进行抓包，**在liunx等服务器上，只能通过tcpdump抓取网络包**。

大部分 Linux 发行包都预装了 tcpdump，如果没有预装，可以用对应操作系统的包管理命令安装，比如在 Centos 下，可以用 yum install -y tcpdump 来进行安装。

本台试验的云服务器已预装了 tcpdump。

验证是否预装了tcpdump很简单，只要在命令行上输入tcpdump，如果突然涌现出很多的报文，那么就没啥问题了：

```
[root@VM-0-13-centos swg]# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
19:42:29.225433 IP VM-0-13-centos.ssh > 222.94.67.180.13183: Flags [P.], seq 2827571698:2827571
886, ack 1714267466, win 317, length 188
19:42:29.225713 IP VM-0-13-centos.60117 > 183.60.83.19.domain: 37232+ PTR? 180.67.94.222.in-add
r.arpa. (44)
19:42:29.235873 IP 222.94.67.180.13183 > VM-0-13-centos.ssh: Flags [.], ack 188, win 509, lengt
h 0
19:42:29.289462 IP 183.60.83.19.domain > VM-0-13-centos.60117: 37232 NXDomain 0/1/0 (93)
19:42:29.290450 IP VM-0-13-centos.58015 > 183.60.83.19.domain: 13718+ PTR? 13.0.17.172.in-addr.
arpa. (42)
19:42:29.291443 IP 183.60.83.19.domain > VM-0-13-centos.58015: 13718 NXDomain 0/1/0 (101)
19:42:29.291581 IP VM-0-13-centos.56327 > 183.60.83.19.domain: 31429+ PTR? 19.83.60.183.in-addr
.arpa. (43)
19:42:29.291643 IP VM-0-13-centos.ssh > 222.94.67.180.13183: Flags [P.], seq 188:360, ack 1, wi
n 317, length 172
19:42:29.292686 IP 183.60.83.19.domain > VM-0-13-centos.56327: 31429 NXDomain 0/1/0 (107)
19:42:29.292875 IP VM-0-13-centos.ssh > 222.94.67.180.13183: Flags [P.], seq 360:1204, ack 1, w
in 317, length 844
19:42:29.292913 IP VM-0-13-centos.ssh > 222.94.67.180.13183: Flags [P.], seq 1204:1360, ack 1,
win 317, length 156
19:42:29.292942 IP VM-0-13-centos.ssh > 222.94.67.180.13183: Flags [P.], seq 1360:1516, ack 1,
win 317, length 156
19:42:29.292969 IP VM-0-13-centos.ssh > 222.94.67.180.13183: Flags [P.], seq 1516:1672, ack 1,
win 317, length 156
19:42:29.293000 IP VM-0-13-centos.ssh > 222.94.67.180.13183: Flags [P.], seq 1672:1828, ack 1,
win 317, length 156
19:42:29.293044 IP VM-0-13-centos.ssh > 222.94.67.180.13183: Flags [P.], seq 1828:1984, ack 1,
win 317, length 156
19:42:29.303760 IP 222.94.67.180.13183 > VM-0-13-centos.ssh: Flags [.], ack 1204, win 512, leng
th 0
19:42:29.303779 IP VM-0-13-centos.ssh > 222.94.67.180.13183: Flags [P.], seq 1984:2140, ack 1,
win 317, length 156
19:42:29.303787 IP 222.94.67.180.13183 > VM-0-13-centos.ssh: Flags [.], ack 1516, win 511, leng
th 0
19:42:29.303865 IP VM-0-13-centos.ssh > 222.94.67.180.13183: Flags [P.], seq 2140:2496, ack 1,
win 317, length 356
19:42:29.303898 IP VM-0-13-centos.ssh > 222.94.67.180.13183: Flags [P.], seq 2496:2652, ack 1,
win 317, length 156
19:42:29.303926 IP VM-0-13-centos.ssh > 222.94.67.180.13183: Flags [P.], seq 2652:2808, ack 1,
win 317, length 156
19:42:29.304115 IP 222.94.67.180.13183 > VM-0-13-centos.ssh: Flags [.], ack 1828, win 510, leng
th 0
```

当然了，tcpdump后面可以跟很多选项，比如：

tcpdump -i any，-i表示指定抓取哪个网卡的网络包，any表示所有，也可以抓取比如tcpdump -i eth0，如何查看网卡？就是熟知的ifconfig：

```
[root@VM-0-13-centos swg]# ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.18.0.1  netmask 255.255.0.0  broadcast 0.0.0.0
        inet6 fe80::42:38ff:fec3:2ccf  prefixlen 64  scopeid 0x20<link>
        ether 02:42:38:c3:2c:cf  txqueuelen 0  (Ethernet)
        RX packets 149  bytes 8035 (7.8 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 95  bytes 10014 (9.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.13  netmask 255.255.240.0  broadcast 172.17.15.255
        inet6 fe80::5054:ff:fea3:7d31  prefixlen 64  scopeid 0x20<link>
        ether 52:54:00:a3:7d:31  txqueuelen 1000  (Ethernet)
        RX packets 207454972  bytes 22903466150 (21.3 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 227139737  bytes 43959037763 (40.9 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1  (Local Loopback)
        RX packets 4786979  bytes 2797694165 (2.6 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4786979  bytes 2797694165 (2.6 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

lo就是本地回环地址，即127.0.0.1，这个相信大家已经很清楚了。

当然了也可以针对某个特定的ip或端口进行筛选抓取，不然抓出来的网络包会太大，如tcpdump -i any host 10.211.55.2可以指定只抓取ip为10.211.55.2 的网络包。我们本地抓包所用的命令如下：

```
tcpdump -i any port 8080 -s 0 -w /root/swg/catch.pcap
```

首先-i any是上面所述的抓取任意网卡的网络包，port 8080指定只抓取8080的网络包，-s 0表示不对网络包截断，抓取完整的数据包，-w表示直接将包写入文件中，并不分析和打印出来，最后就是输出的文件位置，注意后缀名是pcap，即wireshark所认识的包。

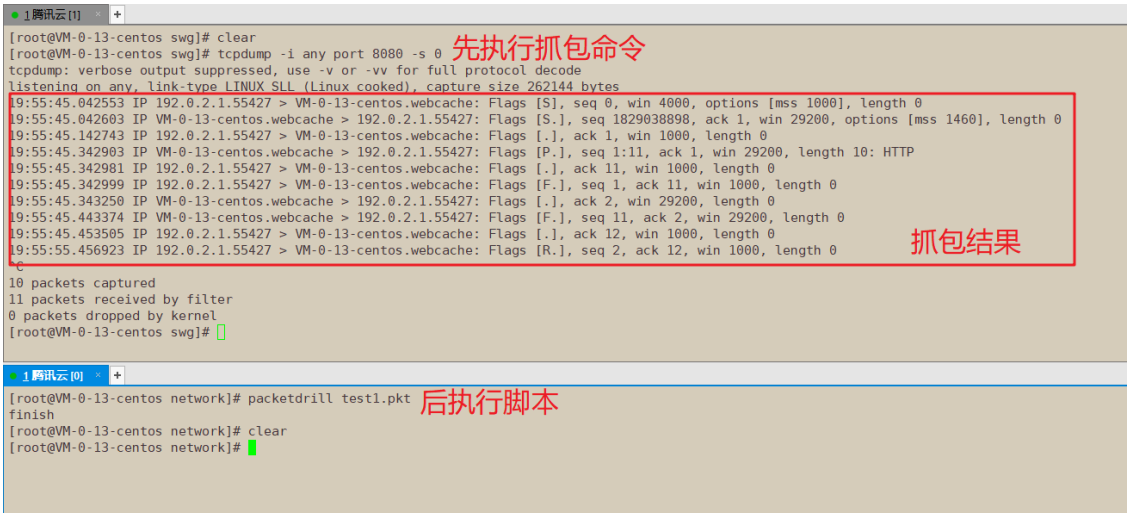
或许读者会说，wireshark你又没介绍过咋用，是的，打算在应用层好好说一说wireshark的使用方式，不过即使你没使用过，也是没有问题的，我们可以将命令改为：

```
tcpdump -i any port 8080 -s 0
```

我们直接在黑窗口上看输出即可，就是难看一点，下面我们针对这两种抓包结果展示形式都简单提一下。

五、对抓包分析分析

我们所谓的验证，就是看实际的网络包是否符合脚本的预期，我们先直接在命令行上看抓包结果，只需要打开两个窗口，一个窗口先执行tcpdump，另一个窗口后执行脚本（请注意，一定是要先抓包，再执行，不然会抓不到或遗漏），执行结果如下：



我们来分析下结果，我将结果拷贝出来：

```
19:55:45.042553 IP 192.0.2.1.55427 > VM-0-13-centos.webcache:
Flags [S], seq 0, win 4000, options [mss 1000], length 0
19:55:45.042603 IP VM-0-13-centos.webcache > 192.0.2.1.55427:
Flags [S.], seq 1829038898, ack 1, win 29200, options [mss 1460],
length 0
19:55:45.142743 IP 192.0.2.1.55427 > VM-0-13-centos.webcache:
Flags [.], ack 1, win 1000, length 0
19:55:45.342903 IP VM-0-13-centos.webcache > 192.0.2.1.55427:
Flags [P.], seq 1:11, ack 1, win 29200, length 10: HTTP
19:55:45.342981 IP 192.0.2.1.55427 > VM-0-13-centos.webcache:
Flags [.], ack 11, win 1000, length 0
19:55:45.342999 IP 192.0.2.1.55427 > VM-0-13-centos.webcache:
Flags [F.], seq 1, ack 11, win 1000, length 0
19:55:45.343250 IP VM-0-13-centos.webcache > 192.0.2.1.55427:
Flags [.], ack 2, win 29200, length 0
19:55:45.443374 IP VM-0-13-centos.webcache > 192.0.2.1.55427:
Flags [F.], seq 11, ack 2, win 29200, length 0
19:55:45.453505 IP 192.0.2.1.55427 > VM-0-13-centos.webcache:
Flags [.], ack 12, win 1000, length 0
19:55:55.456923 IP 192.0.2.1.55427 > VM-0-13-centos.webcache:
Flags [R.], seq 2, ack 12, win 1000, length 0
```

首先看前三行：

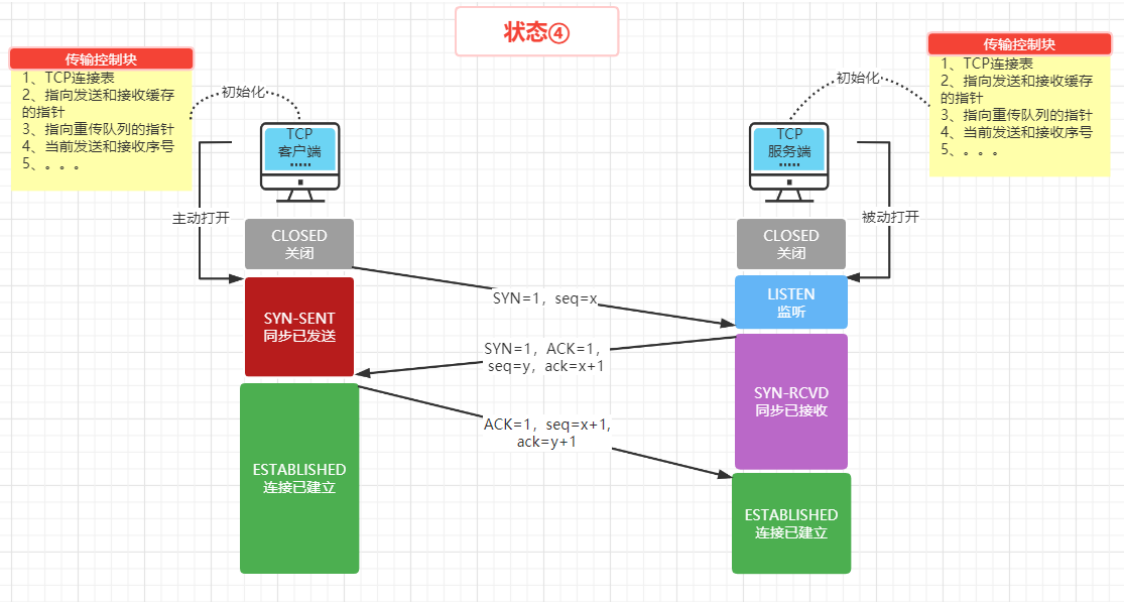
```
19:55:45.042553 IP 192.0.2.1.55427 > VM-0-13-centos.webcache:
Flags [S], seq 0, win 4000, options [mss 1000], length 0
19:55:45.042603 IP VM-0-13-centos.webcache > 192.0.2.1.55427:
Flags [S.], seq 1829038898, ack 1, win 29200, options [mss 1460],
length 0
19:55:45.142743 IP 192.0.2.1.55427 > VM-0-13-centos.webcache:
Flags [.], ack 1, win 1000, length 0
```

可以看到第一行是S报文，即SYN报文，相对序列号为0，不携带数据所以lendth为0。

第二行是S.报文，即SYN+ACK报文，确认号为1（虽然SYN不携带数据，但是要消耗一个序列号）。

第三行是.报文，即ACK报文，确认号为1，针对服务端的SYN进行确认，不携带数据因此length也为0。

符合此流程：



三次握手建立成功，下面进行数据传输。

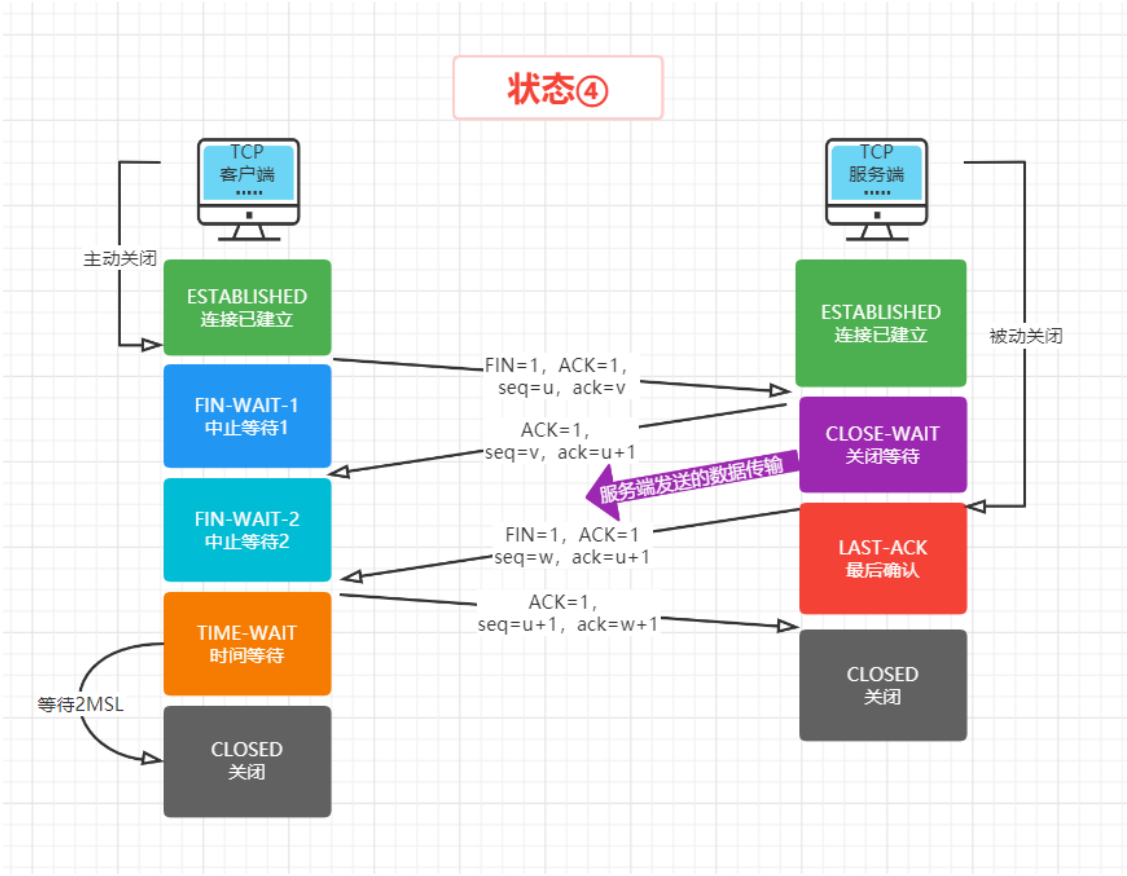
```
19:55:45.342903 IP VM-0-13-centos.webcache > 192.0.2.1.55427:
Flags [P.], seq 1:11, ack 1, win 29200, length 10: HTTP
19:55:45.342981 IP 192.0.2.1.55427 > VM-0-13-centos.webcache:
Flags [.], ack 11, win 1000, length 0
```

服务端给客户端发送10字节数据，这里看到的是P.报文，即PSH+ACK报文，ack还为1是对上次的重复确认，PSH表示这个数据无需缓存，直接上交给上层应用即可，序列号是从1开始，长度为10，因此是1-11的范围。

客户端返回.即ACK报文，ack为11，即对服务端发送的10个字节的收到确认，如果还有数据，请从11开始发送给我。

```
19:55:45.342999 IP 192.0.2.1.55427 > VM-0-13-centos.webcache:
Flags [F.], seq 1, ack 11, win 1000, length 0
19:55:45.343250 IP VM-0-13-centos.webcache > 192.0.2.1.55427:
Flags [.], ack 2, win 29200, length 0
19:55:45.443374 IP VM-0-13-centos.webcache > 192.0.2.1.55427:
Flags [F.], seq 11, ack 2, win 29200, length 0
19:55:45.453505 IP 192.0.2.1.55427 > VM-0-13-centos.webcache:
Flags [.], ack 12, win 1000, length 0
```

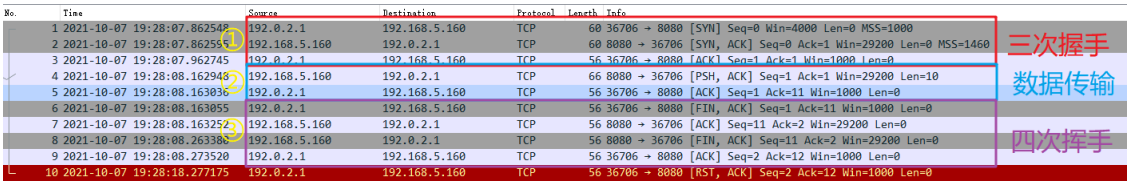
客户端主动发起断开，发出的是F.即FIN+ACK报文，服务端回复ACK，待会服务端发起FIN+ACK，客户端回复ACK，挥手结束。请注意看下这里的ack，可以佐证FIN也会消耗一个序列号。



下面来看看wireshark的抓包，我们执行下面这个命令：

```
tcpdump -i any port 8080 -s 0 -w /root/swg/catch.pcap
```

将输出的文件通过sz命令下载到本地，用wireshark打开，更加地清晰明了：



可以初步感受下wireshark的魅力，我们在应用层的时候将单独开辟一章节隆重介绍下。

六、packetdrill原理概述

脚本的最后延迟了10秒，实际上是用来看一个效果，我们再来执行下脚本，看下网卡信息：

```
[root@VM-0-13-centos swg]# ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.18.0.1  netmask 255.255.0.0  broadcast 0.0.0.0
        inet6 fe80::42:38ff:fec3:2ccf  prefixlen 64  scopeid 0x20<link>
        ether 02:42:38:c3:2c:cf  txqueuelen 0  (Ethernet)
        RX packets 149  bytes 8035 (7.8 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 95  bytes 10014 (9.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.13  netmask 255.255.240.0  broadcast 172.17.15.255
        inet6 fe80::5054:ff:fea3:7d31  prefixlen 64  scopeid 0x20<link>
        ether 52:54:00:a3:7d:31  txqueuelen 1000  (Ethernet)
        RX packets 207472725  bytes 22905372262 (21.3 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 227157301  bytes 43962108000 (40.9 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1  (Local Loopback)
        RX packets 4787209  bytes 2797730025 (2.6 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4787209  bytes 2797730025 (2.6 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

tun0: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST>  mtu 1500
        inet 192.168.228.145  netmask 255.255.0.0  destination 192.168.228.145
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 500  (UNSPEC)
        RX packets 5  bytes 204 (204.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4  bytes 174 (174.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[root@VM-0-13-centos swg]#
```

发现多出来一个tun0网卡，脚本执行完，tun0 会被销毁。基本原理：通过write等系统调用可以将数据写入，通过该网卡将数据送给内核协议栈；反过来，read 系统调用读取数据的过程类似，协议栈会将响应报文通过网卡tun0传递给packetdrill应用。