一——TCP三次握手下的细节

> 这并不是一篇涉及细节的文章，更多地是结合我的知识，从更高的一面看待TCP的三次握手。不是很了解TCP的话，可以先看一看《TCP的那些事儿（上）》与《TCP 的那些事儿（下）》。

在校园招聘面试时候，最经常问到的一个网络问题是TCP的三次握手。有关这个问题，这里我并不想写那些在互联网上烂大街的答案，而是结合我的所知，更深入地探究这个问题。

很自然地想到：为什么是**三次握手**呢?
我对三次握手原因的印象极为深刻，在大三学习《网络编程》的时候，老师为这个问题举了一个德军坦克的例子：两苏军军队夹击一德军，单独攻击必败，联合攻击必胜，苏军之间通信需要借助通信员，而通信员在传递消息的过程中可能被德军逮捕，苏军一方如何可靠地通知另一方攻击时间是一个难题。本质上，这个问题是由于信道不可靠，但是通信双方需要就某个问题达成一致，三次通信是理论上的最小值。TCP可靠的精髓源于它32位长的序列号（Initial Sequence Number），TCP的连接握手确定了通信双方数据原点的序列号。[1]若最后一次ACK丢失，一定时间之后另一方会重发SYN-ACK，重复次数根据实现来确定。

谈起序列号，在高速网络下回避不掉的是**序列号回绕**问题。RFC 1312[2]给出了以下数据：

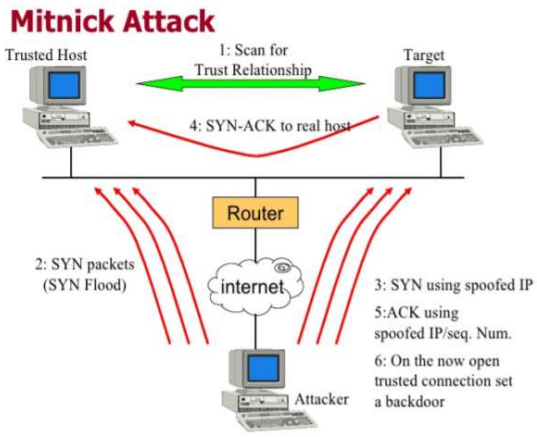
> ```
> Network        B*8          B          Twrap
>                bits/sec     bytes/sec  secs
> -------        -------      ------     ------
> ARPANET        56kbps       7KBps      3*10**5 (~3.6 days)
> DS1            1.5Mbps      190KBps    10**4 (~3 hours)
> Ethernet       10Mbps       1.25MBps   1700 (~30 mins)
> DS3            45Mbps       5.6MBps    380
> FDDI           100Mbps      12.5MBps   170
> Gigabit        1Gbps        125MBps    17
> ```
>
> Note: Twrap —— Time until Wraparound

在高带宽下，TCP序列号可能在较短的时间就被重复使用，可能导致同一条TCP流在短时间内出现序列号一样的两个合法的数据包及其确认数据，32位长的序列号已经满足不了需求。为了compatibility与overhead，通过使用TCP选项中的timestamp，记录数据包的发送时间，利用PAWS（Protect Againest Wrapped Sequence numbers）机制解决了此问题。此外，timestamp选项页提供了一种精确测量RTT（Round Trip Time）方法，RTT的测量对超时重传机制有着重要的影响，这也是timestamp的主要设计目的。timestamp并不要求时钟同步，因

为时标是在同一端被写入和读取的。[3]Linux下可以通过`net.ipv4.tcp_timestamps`选项打开。

记得课堂上老师还提到过TCP**序列号的安全性**问题，序列号产生的随机性影响着系统安全。举的例子是Kevin Mitnick's Christmas Day crack of Tsutomu Shimomura's machine[4][5]：

The Kevin Mitnick Attack



Target中TCP序列号的产生并非随机，并且可由Trusted Host无密码登陆。利用这些条件，Mitnick把Attacker伪装成Trusted Host，在Target上打开了一个后门。从而发展出了IP欺骗攻击（IP Spoofing Attack）与TCP序列号预测技术。还有论文《Strange Attractors and TCP/IP Sequence Number Analysis 》研究了那时候（2001）各个操作系统序列号初始化的随机性。

在三次握手中存在的另一个安全问题是**SYN洪泛**（SYN flood）。它是一种阻断服务攻击，只在服务器收到SYN后分配资源、但在收到ACK之前这段时间内资源有效，导致海量的SYN耗尽服务器的资源所致。使用SYN Cookie机制是一种防范手段，它的原理是不在SYN后而是收到ACK后为TCP连接分配资源，Linux下通过`net.ipv4.tcp_syncookies`选项打开。[6]

曾经在网上看到一个问题：TCP中滑动窗口的大小（Window Size）在什么阶段确定呢？答案是TCP三次握手中的SYN、SYN/ACK阶段。准确地说，Window Size告知了发送方能接收的最大数据字节数，防止接收缓冲区溢出的情况。在带宽延迟积（bandwidth-delay product，DBP）很大的情况下（这种网络又叫长肥网络，Long Fat Network，LFN），不仅面临TCP序列号回绕问题，也有`receive window`过小的问题，因此为了**保持管道满载**，引入了TCP window scale选项（RFC 1312[2]）。[3]Linux下通过`net.ipv4.tcp_window_scaling`选项打开。

所以，在TCP的三次握手过程中还发生了什么？对于这个问题，我使用Wireshark抓包，三次握手过程显示了如下结果：

```
Transmission Control Protocol, Src Port: 54730, Dst Port: 80, Seq: 0, Len: 0
    Source Port: 54730
    Destination Port: 80
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    [Next sequence number: 0    (relative sequence number)]
    Acknowledgment number: 0
    1010 .... = Header Length: 40 bytes (10)
  > Flags: 0x002 (SYN)
    Window size value: 29200
    [Calculated window size: 29200]
    Checksum: 0xfa3d [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  v Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
      > TCP Option - Maximum segment size: 1460 bytes
      > TCP Option - SACK permitted
      > TCP Option - Timestamps: TSval 2670317591, TSecr 0
      > TCP Option - No-Operation (NOP)
      > TCP Option - Window scale: 7 (multiply by 128)
    > [Timestamps]
Transmission Control Protocol, Src Port: 80, Dst Port: 54730, Seq: 0, Ack: 1, Len: 0
    Source Port: 80
    Destination Port: 54730
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    [Next sequence number: 0    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    1010 .... = Header Length: 40 bytes (10)
  > Flags: 0x012 (SYN, ACK)
    Window size value: 43440
    [Calculated window size: 43440]
    Checksum: 0x4271 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  v Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
      > TCP Option - Maximum segment size: 1460 bytes
      > TCP Option - SACK permitted
      > TCP Option - Timestamps: TSval 229959879, TSecr 2670317591
      > TCP Option - No-Operation (NOP)
      > TCP Option - Window scale: 11 (multiply by 2048)
    > [SEQ/ACK analysis]
    > [Timestamps]
Transmission Control Protocol, Src Port: 54730, Dst Port: 80, Seq: 1, Ack: 1, Len: 0
    Source Port: 54730
    Destination Port: 80
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 1    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x010 (ACK)
    Window size value: 229
    [Calculated window size: 29312]
    [Window size scaling factor: 128]
    Checksum: 0xfa35 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  v Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
      > TCP Option - No-Operation (NOP)
      > TCP Option - No-Operation (NOP)
      > TCP Option - Timestamps: TSval 2670317740, TSecr 229959879
    > [SEQ/ACK analysis]
    > [Timestamps]
```

在三次握手的TCP Options中，两部分的确符合描述，但剩下的三部分：Maximum segment size、SACK permitted、NOP是什么呢?
Wikipedia[7]上面给出了答案：

> - 1 (8 bits): No operation (NOP, Padding) This may be used to align option fields on 32-bit boundaries for better performance.
> - 1 (8 bits): No operation (NOP, Padding) This may be used to align option fields on 32-bit boundaries for better performance.
> - 2,4,*SS* (32 bits): Maximum segment size (*see maximum segment size*) [SYN]
> - 3,3,*S* (24 bits): Window scale (*see window scaling for details*) [SYN]

> - 4,2 (16 bits): Selective Acknowledgement permitted. [SYN] (*See selective acknowledgments for details*)
> - 5,*N*,*BBBB*,*EEEE*,... (variable bits, *N* is either 10, 18, 26, or 34)- Selective ACKnowledgement (SACK). These first two bytes are followed by a list of 1–4 blocks being selectively acknowledged, specified as 32-bit begin/end pointers.
> - 8,10,*TTTT*,*EEEE* (80 bits)- Timestamp and echo of previous timestamp (*see TCP timestamps for details*)

NOP用于字节对齐。而MSS（Maximum Segment Size），隐含着TCP虽然是流协议，却是以数据包的形式发送的，这是由承载协议确定的。为了获取最佳性能，应该将MSS设置得足够小以避免被IP分段，否则会导致数据包丢失的概率增加以及过多的重复传输。因此在建立连接的时候，通常会协商这个参数，它是通过数据链路层的MTU（Maximum Transmission Unit）导出的，TCP可以通过PMTUD（Path MTU Discovery）来推断通信链路上最小的MTU。
至于SACK（Selective ACKnowledgement，SACK），则与拥塞控制有关。拥塞控制的精髓在于：1.序列号，2.ACK。ACK的累加并非按照数据包的个数，而是发送的数据字节数。简单来说，标准的TCP的拥塞控制机制（从Tahoe到Reno，至New Reno）遵循线增积减（Additive Increase/Multiplicative Decrease，AIMD）、慢启动（Slow Start）、快速重传（Fast Retransmit）、快速恢复（Fast Recovery），以及可选的特性选择确认。更多关于拥塞控制的内容，打算另起一篇记录。

还想到一个问题：TCP可以在第三次握手的时候传输数据吗？答案是可以的，不过还是要看具体的实现，相关的术语是**捎带**。TCP的确认机制中允许**延迟确认**，允许新数据捎带ACK过去。Linux下可通过设置`TCP_QUICKACK`来禁止延迟确认机制。
更进一步，TCP怎样传送一个报文段？TCP属于字节流抽象，由TCP决定字节数是否达到足以发送一个报文段的要求，有三种机制**触发传输**：1. TCP从发送进程收集到MSS字节；2. 发送进程明确要求发送一个报文段，即TCP flag PUSH；3. 定时激活机制，结果报文中包含当前缓冲区中所有需要发送出去的字节。但这种定时器并非完美，在发送方传输小报文段或接收方打开小窗口时，会出现傻瓜窗口症状（silly window syndrome）。因此有了Nagle算法，默认开启，禁止的话TCP选项为`TCP_NODELAY`，这意味着数据被尽可能快地传输。[12] Linux下还可以通过`TCP_CORK`设置更加激进的Nagle算法，它完全禁止了小包的发送。
虽然TCP使用`connect / send / recv`这组API，但UDP也可以使用。对UDP来说，`connect`意味着为socket记录下了目的地址与端口，这样就可以使用`send`发送数据了。[11]

Reference

[1]: https://www.zhihu.com/question/24853633
[2]: https://tools.ietf.org/html/rfc1323
[3]: http://perthcharles.github.io/2015/08/27/timestamp-intro/
[4]: http://cs.boisestate.edu/~jxiao/cs333/01-kevinmitnick.pdf
[5]: http://wiki.cas.mcmaster.ca/index.php/The_Mitnick_attack

[6]: https://en.wikipedia.org/wiki/SYN_flood
[7]: https://en.wikipedia.org/wiki/Transmission_Control_Protocol#TCP_segment_structure
[8]: http://intronetworks.cs.luc.edu/1/html/ethernet.html
[9]: https://www.zhihu.com/question/21524257
[10]: https://inst.eecs.berkeley.edu/~ee122/fa05/projects/Project2/SACKRENEVEGAS.pdf
[11]: https://stackoverflow.com/questions/6189831/whats-the-purpose-of-using-sendto-recvfrom-instead-of-connect-send-recv-with-ud
[12]: 《计算机网络——系统方法》

---

by river[river@vvl.me]
2019.0205: initialization
2019.0215: add 捎带&触发传输