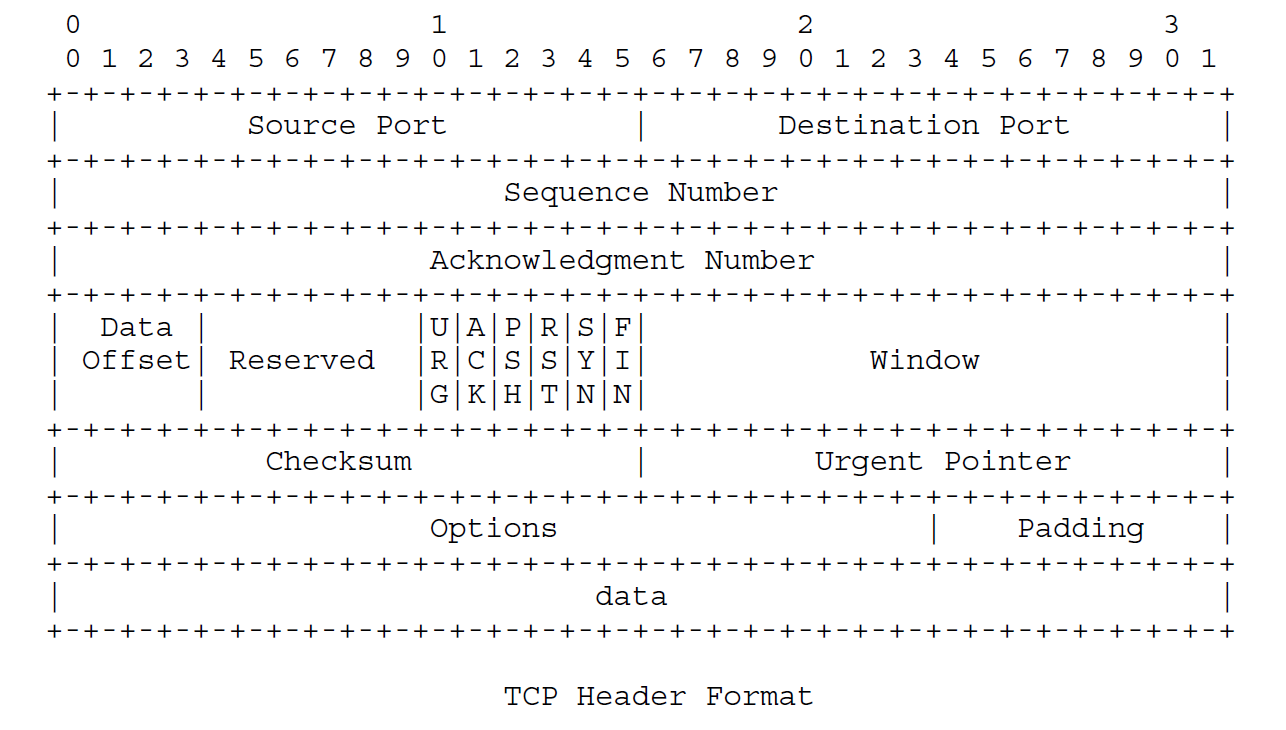
# RFC793

## 3 功能规范

### 3.1 头部格式

TCP段作为因特网数据报发送。IP报头包含多个信息字段，包括源和目标主机地址[2]。TCP头部跟在IP头部之后，提供特定于TCP协议的信息。这种划分允许存在除TCP之外的主机级协议。



源端口：16比特，源端口号；

目的端口：16比特，目的端口号；

序列号：32比特，此段中第一个数据字节的序列号（存在SYN时除外）。如果存在SYN，则序列号是初始序列号（ISN），第一个数据字节是ISN + 1。

确认号：32比特，如果设置了ACK控制位，则该字段包含该段的发送者期望接收的下一个序列号的值。建立连接后，始终会发送此连接。

数据偏移：4比特，TCP头中32位字的数目。这表示数据的起始位置。TCP头（甚至包括选项）是32位长的整数。

保留：6比特，保留供将来使用。必须为零。

控制位：6比特（从左到右）：

URG：紧急指针字段有效

ACK：确认字段有效

PSH：推送功能

RST：重置连接

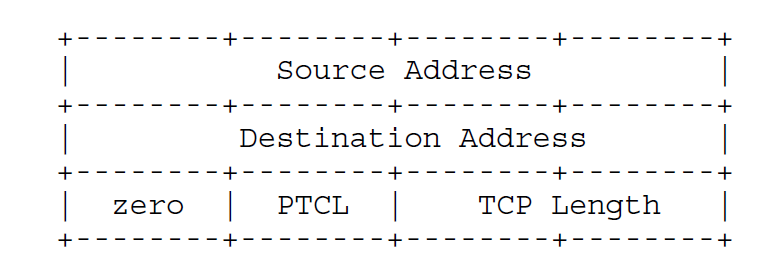
SYN：同步序列号

FIN：发送端没有更多数据

窗口：16比特，以确认字段中指示的数据字节数开始，该段的发送方愿意接收的数据字节数目。

校验和：16比特，校验和字段是标题和文本中所有16位字的一个补码和的16位补码。如果一个段包含要校验和的头和文本字节为奇数，则最后一个字节用零填充在右边，形成一个16位字用于校验和。填充值不作为段的一部分传输。在计算校验和时，校验和字段本身将替换为零。

校验和还包括一个96位伪头，在概念上作为TCP头的前缀。此伪标头包含源地址，目标地址，协议号和TCP长度。这为TCP提供了针对错误路由段的保护。此信息在Internet协议中承载，并通过TCP/IP网络接口在IP上的TCP调用的参数或结果中传输。



TCP长度是TCP报头长度加上数据的长度（这不是显式发送的数量，而是计算的），并且它不计算伪报头的12个字节。

紧急指针：16比特，该字段将紧急指针的当前值传递为该段中序列号的正偏移量。紧急指针指向紧急数据之后的字节的序列号（在RFC1122中被更正为指向紧急数据字节的序列号）。该字段仅在设置了URG控制位的段中进行解释。

选项：可变

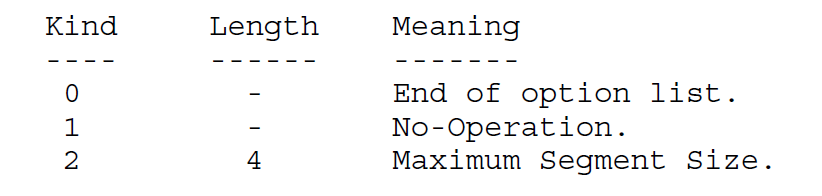
选项可能占用TCP标头末尾的空间，并且长度为8位的倍数。所有选项都包含在校验和中。选项可以从任何字节边界开始。选项格式有两种情况：

情况1：选项类型的单个字节。

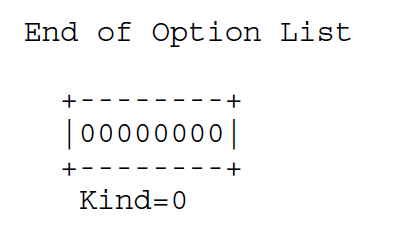
情况2：选项类型的字节，选项长度的字节和实际的选项-数据字节。

选项长度计算选项类型和选项长度的两个字节以及选项-数据字节数目。请注意，选项列表可能比数据偏移字段可能暗示的要短。超出选项结束选项的标题的内容必须是标题填充（即，零）。TCP必须实现所有选项。

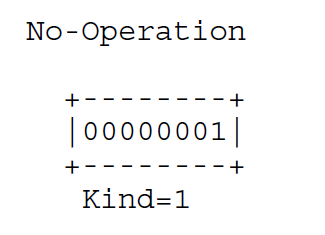
目前定义的选项包括（以字节表示的种类）：



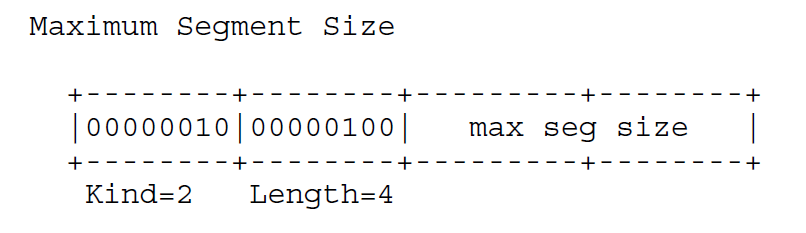
具体选项定义



此选项代码表示选项列表的结尾。根据数据偏移字段，这可能与TCP标头的末尾不一致。这在所有选项的末尾使用，而不是在每个选项的结尾处使用，并且只有在选项的末尾不与TCP头的末尾重合时才需要使用。



可以在选项之间使用该选项代码，例如，以对齐单词边界上的后续选项的开头。无法保证发件人将使用此选项，因此接收者必须准备好处理选项，即使他们不是从单词边界开始。



最大段大小选项数据：16位，如果存在此选项，则它会在发送此段的TCP上传达最大接收段大小。该字段只能在初始连接请求中发送（即，在设置了SYN控制位的段中）。如果未使用此选项，则允许任何段大小。

填充：可变，TCP头填充用于确保TCP头结束并且数据在32位边界上开始。填充由零组成。

### 3.2 术语

在我们讨论TCP的操作之前，我们需要介绍一些详细的术语。维护TCP连接需要记住几个变量。我们设想这些变量存储在称为传输控制块或TCB的连接记录中。

存储在TCB中的变量包括本地和远程套接字号，连接的安全性和优先级，指向用户的发送和接收缓冲区的指针，指向重新传输队列和当前段的指针。

另外，与发送和接收序列号有关的几个变量也存储在TCB中。

发送序列变量

SND.UNA：发送未确认的

SND.NXT：发送下一个

SND.WND：发送窗口

SND.UP：发送紧急指针

SND.WL1：用于最后一次窗口更新的段序列号

SND.WL2：用于最后一次窗口更新的段确认号

ISS：初始发送序列号

接收序列变量

RCV.NXT：接收下一个

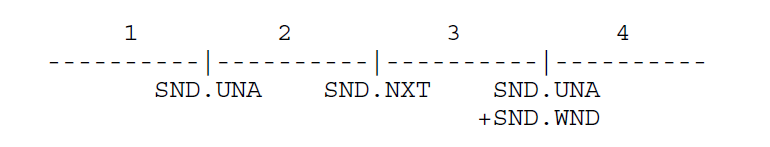
RCV.WND：接收窗口

RCV.UP：接收紧急指针

IRS：初始接收序列号

下面的图表可能有助于将这些变量中的一些关联到序列空间。

发送序列空间



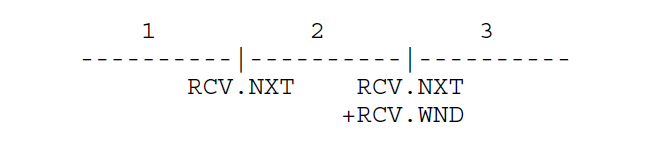
1、已确认的序列号

2、未确认数据的序列号

3、允许新数据传输的序列号

4、未来允许的序列号

接收序列空间



1、已确认的序列号

2、允许新接收的序列号

3、未来允许的序列号

在讨论中还经常使用一些变量，这些变量从当前段的字段中获取它们的值。

当前段变量

SEG.SEQ：段序列号

SEG.ACK：段确认号

SEG.LEN：段长度

SEG.WND：段窗口

SEG.UP：段紧急指针

SEG.PRC：段优先值

连接在其生命周期中通过一系列状态进行。状态是：LISTEN，SYN-SENT，SYN-RECEIVED，ESTABLISHED，FIN-WAIT-1，FIN-WAIT-2，CLOSE-WAIT，CLOSING，LAST ACK，TIME-WAIT和虚构状态CLOSED。CLOSED是虚构的，因为它表示没有TCB时的状态，因此没有连接。简言之状态的含义：

LISTEN：表示等待来自任何远程TCP和端口的连接请求。

SYN-SENT：表示在发送连接请求后等待匹配的连接请求。

SYN-RECEIVED：表示在收到并发送连接请求后等待确认连接请求确认。

ESTABLISHED：表示开放连接，接收的数据可以传递给用户。 连接的数据传输阶段的正常状态。

FIN-WAIT-1：表示等待来自远程TCP的连接终止请求，或者对先前发送的连接终止请求的确认。

FIN-WAIT-2：表示等待来自远程TCP的连接终止请求。

CLOSE-WAIT：表示等待本地用户的连接终止请求。

CLOSING：表示等待来自远程TCP的连接终止请求确认。

LAST-ACK：表示等待先前发送到远程TCP的连接终止请求的确认（其包括对其连接终止请求的确认）。

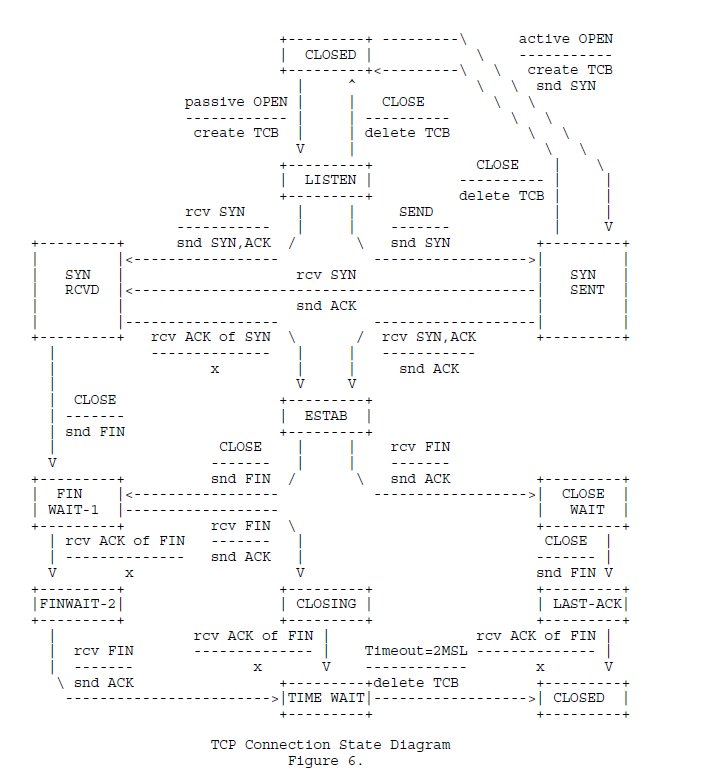
TIME-WAIT：表示等待足够的时间传递以确保远程TCP收到其连接终止请求的确认。

CLOSED：表示根本没有连接状态。

TCP连接从一个状态进展到另一个状态以响应事件。事件是用户呼叫，OPEN，SEND，RECEIVE，CLOSE，ABORT和STATUS；传入的段，特别是那些包含SYN，ACK，RST和FIN标志的段；和超时。

图6中的状态图仅说明状态变化以及导致事件和结果动作，但既不解决错误条件也不解决与状态变化无关的动作。

在后面的部分中，提供了有关TCP对事件的反应的更多细节。注意：此图仅是摘要，不得视为总规格。



### 3.3 序列号

设计中的一个基本概念是，通过TCP连接发送的每个字节(1 octet = 8 bit)数据都有一个序列号。因为每个字节都被排序，所以每个字节都可以被确认。所采用的确认机制是累积的，因此序列号X的确认表示已接收到所有不包括X的字节。这种机制允许在重发时直接进行重复检测。段内字节的编号是指紧接TCP头后面的第一个数据字节，这个字节的编号是最低的，接下来的字节是连续编号的。

必须记住，实际的序列号空间是有限的，尽管非常大。此空间的范围为0到2^32-1。由于空间是有限的，所有处理序列号的算法都必须执行模2^32。

当序列号再次从2^32-1循环到0时，此无符号算术保持序列号之间的关系。计算机模运算有一些微妙之处，因此在编制此类数值的比较程序时应格外小心。符号“=<”表示“小于或等于”（模2^32）。

TCP必须执行的序列号比较的典型类型包括：

(a)确定确认是指发送但尚未确认的某个序列号。

(b)确定一个段占用的所有序列号已被确认（例如，从重传队列中删除该段）

(c)确定输入段包含预期的序列号（即，段“重叠”接收窗口）。

为响应发送数据，TCP将收到确认。处理确认需要进行以下比较。

snd.una=最早的未确认序列号

snd.nxt=要发送的下一个序列号

seg.ack=接收TCP的确认（接收TCP预期的下一个序列号）

seg.seq=段的第一个序列号

seg.len=段中数据占用的八位字节数（计算syn和fin）

seg.seq+seg.len-1=段的最后一个序列号

新的确认（称为“可接受的确认”）是以下不平等的一种确认：

SND.UNA < SEG.ACK =< SND.NXT

如果重新传输队列上的一个段的序列号和长度之和小于或等于传入段中的确认值，则该段被完全确认。

当收到数据时，需要进行以下比较：

rcv.nxt=接收段上预期的下一个序列号，是接收窗口的左边缘或下边缘。

rcv.nxt+rcv.wnd-1=接收段上预期的最后一个序列号，是接收窗口的右边缘或上边缘。

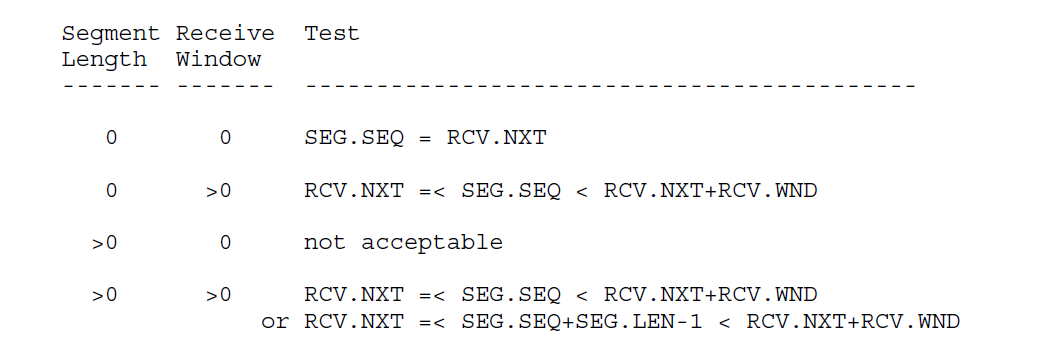
seg.seq=输入段占用的第一个序列号

seg.seq+seg.len-1=输入段占用的最后一个序列号

如果RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND或RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND，一个段被判断为占用有效接收序列空间的一部分。

测试的第一部分检查段的开始部分是否落在窗口中，测试的第二部分检查段的结束部分是否落在窗口中；如果段通过了测试的任何一部分，它就在窗口中包含数据。

实际上，它比这个稍微复杂一点。由于零窗口和零长度段，我们有四种情况可接受的传入段:



注意，当接收窗口为零时，除了ACK段之外，不应该接受任何段。因此，TCP有可能在传输数据和接收ACKS时保持零接收窗口。然而，即使接收窗口为零，TCP也必须处理所有传入段的RST和URG字段。

我们也利用编号方案来保护某些控制信息。这是通过在序列空间中隐式地包含一些控制标志来实现的，这样它们就可以被重新传输和确认而不会产生混淆(只会对控件的一个副本起作用)。控制信息没有实际地存储在段数据空间中。因此，我们必须采用隐式分配序列号给控件的规则。SYN和FIN是唯一需要这种保护的控件，这些控件仅用于打开和关闭连接。为了序列号的目的，SYN被认为发生在它所在段的第一个实际数据之前，而FIN被认为发生在它所在段的最后一个实际数据之后。段长度(seg.len)包括数据和序列占位控件。当一个SYN出现时，然后是SEG，SEQ是SYN的序号。

**初始序列号选择**

协议对反复使用的特定连接没有限制。连接由一对套接字定义。连接的新实例将被称为连接的体现。由此产生的问题是——“TCP如何识别连接以前版本中的重复段?”如果连接被连续快速地打开和关闭，或者连接由于内存丢失而中断，然后重新建立，那么这个问题就会变得很明显。

为了避免混淆，我们必须防止在网络中使用来自某个连接的一个化身的段，而来自早期化身的相同序列号可能仍然存在。我们希望确保这一点，即使TCP崩溃并丢失了它一直使用的序列号的所有知识。当创建新的连接时，将使用初始序列号(ISN)生成器来选择一个新的32位ISN。生成器绑定到一个(可能是虚构的)32位时钟，其低阶位大约每4微秒递增一次。因此，ISN大约每4.55小时循环一次。由于我们假定段在网络中停留的时间不会超过最大段生存期(MSL)，并且MSL小于4.55小时，因此我们可以合理地假定它是惟一的。

对于每个连接，都有一个发送序列号和一个接收序列号。数据发送TCP选择初始发送序列号（ISS），在建立连接过程中学习初始接收序列号（IRS）。

要建立或初始化连接，两个TCP必须在彼此的初始序列号上同步。这是在建立连接的交换中完成的，建立带有一个名为“SYN”（用于同步）的控制位和初始序列号的段。简而言之，携带SYN位的段也称为“SYNs”。因此，该解决方案需要一种合适的机制来选择初始序列号，并使用稍微复杂的握手来交换ISN。

同步要求每一方发送自己的初始序列号，并从另一方接收确认信息。每一方还必须接收另一方的初始序列号并发送确认。

1）A-->B SYN 我的序列号是X

2）A<--B ACK 你的序列号是X

3）A<--B SYN 我的序列号是Y

4）A-->B ACK 你的序列号是Y

因为步骤2和3可以组合在一条消息中，所以称为三向（或三条消息）握手。

由于序列号与网络中的全局时钟无关，TCP可能有不同的机制来选择ISN，因此需要进行三路握手。第一个SYN的接收器无法知道该段是否是旧的延迟段，除非它记住连接上使用的最后一个序列号（这并不总是可能的），因此它必须要求发送者验证该SYN。在[3]中讨论了三路握手和时钟驱动方案的优点。

**知道什么时候保持安静**

为了确保TCP不会创建一个带有序列号的段，该序列号可能被网络中剩余的旧段复制。TCP必须在启动时或者在其使用序列号的内存丢失导致崩溃恢复需要分配任何序列号之前保持最大生存时间（MSL）的安静。对于此规范，MSL为2分钟。这是一种工程选择，如果经验表明需要这样做，可能会改变。请注意，如果TCP在某种意义上重新初始化，但仍然保留其使用的序列号的内存，那么它根本不需要等待；它必须只能确保使用大于最近使用的序列号。

**TCP安静的时间概念**

该规范规定，在不保留每个活动（即非关闭）连接上发送的最后序列号的任何知识的情况下“崩溃”的主机应至少延迟发送任何TCP段，以便至少在互联网系统中达成协议的最大段寿命（MSL）。在下面的段落中，给出了对该说明书的解释。TCP实现者可能违反“安静时间”限制，但仅有可能导致某些旧数据被接受为新的或新的数据被拒绝，因为互联网系统中的某些接收器复制旧数据。

每次形成一个段并将其输入到源主机的网络输出队列中时，TCP会消耗序列号空间。TCP协议中的重复检测和排序算法依赖于段数据与序列空间的唯一绑定，以使序列号在绑定到那些序列号的段数据之前不会循环通过所有2^32值，这些序列号已经由接收者传递和确认，并且片段的所有重复副本已从互联网“耗尽”。如果没有这样的假设，可以想象两个不同的TCP段被分配相同或重叠的序列号，从而导致接收器对哪些数据是新的以及哪些是旧的混淆。请记住，每个段都绑定了与段中数据的八位字节一样多的连续序列号。

在正常情况下，TCP会跟踪要发出的下一个序列号和最旧的等待确认，以避免在第一次使用确认之前错误地使用序列号。仅这一点并不能保证从网络中排出旧的重复数据，因此序列空间已经非常大，以减少流浪复制品在到达时会造成麻烦的可能性。以2兆比特/秒。使用2 \*\* 32个八位字节的序列空间需要4.5小时。由于网络中的最大段寿命不可能超过几十秒，因此即使数据速率上升到10兆比特/秒，也可以认为这对可预见的网络提供了充足的保护。在100兆比特/秒时，循环时间为5.4分钟，这可能有点短，但仍然在合理范围内。

但是，如果源TCP没有在给定连接上使用用过的序列号的任何内存，则可以取消TCP中的基本重复检测和排序算法。例如，如果TCP要启动序列号为0的所有连接，则在崩溃并重新启动时，TCP可能会重新形成先前的连接（可能在半开连接解析之后）并发出序列号相同或重叠的数据包。这些数据包仍在网络中，这些数据包是在同一连接的早期版本中发出的。在没有关于特定连接上使用的序列号的知识的情况下，TCP规范建议在连接上发送段之前MSL秒的源延迟，以允许来自早期连接化身的段的时间从系统中消耗。

即使能够记住一天中的时间并且用它来选择初始序列号值的主机也不能免于这个问题（即使使用时间来为每个新的连接化身选择初始序列号）。例如，假设从序列号S开始打开连接。假设此连接使用不多，并且最终初始序列号函数（ISN（t））采用等于序列号的值，例如S1，此TCP在特定连接上发送的最后一个段。现在假设，在这一瞬间，主机崩溃，恢复并建立连接的新化身。选择的初始序列号是S1 = ISN（t） - 最近使用的旧版连接序列号！如果恢复足够快地发生，则S1附近的净承载序列号中的任何旧重复可以到达并且被连接的新化身的接收器视为新分组。

问题是恢复的主机可能不知道它崩溃了多长时间，也不知道系统中是否还存在来自早期连接化身的旧重复。

解决这个问题的一种方法是在从崩溃恢复后故意延迟一个MSL的发射段 - 这是“相当时间”的规范。希望避免等待的主机愿意冒险在给定目的地处可能混淆新旧分组可能选择不等待“相当时间”。实现者可以为TCP用户提供在连接基础上选择是否在崩溃之后等待的能力，或者可以非正式地实现所有连接的“相当时间”。显然，即使用户选择“等待”，在主机已经“启动”至少MSL秒之后也不需要这样做。

总结一下：发出的每个段都占用了序列空间中的一个或多个序列号，段中占用的数字是“忙”或“正在使用”，直到MSL秒已经过去，一旦时空块被崩溃占用了空间。最后一个发射段的八位字节，如果新连接过早启动并使用前一个连接化身的最后一个段的空时足迹中的任何序列号，则存在潜在的序列号重叠区域，这可能在接收器侧导致混淆。