RFC：793

传输控制协议
DARPA 互联网程序
协议规范

1981 年 9 月

委托方

（美国）国防部高级研究计划署（DARPA）
信息处理技术办公室
1400 Wilson Boulevard
Arlington, Virginia 22209

开发方

信息科学研究院
南加州大学
4676 Admiralty Way
Marina del Rey, California 90291

目录

序言

本文描述 DoD 标准传输控制协议（TCP）。ARPA TCP 规范有 9 个早期版本，该版本基于它们，且当前的正文有大量来自它们。有许多人对该工作作出贡献，包括概念方面和正文方面。本版阐明几个细节并删除信尾缓存大小调整，并重新描述信函机制为上推功能。

Jon Postel

编辑

RFC：793
代替：RFC 761
IEN：129、124、112、81、55、44、40、27、21、5

传输控制协议

DARPA 互联网程序
协议规范

1 简介

传输控制协议（TCP）打算用作在分组交换计算机通迅网络和这类网络的内部连接系统的主机间高可靠主机对主机协议。

本文描述传输控制协议将执行的功能，实现的程序，及提供给要求其服务的程序或用户的接口。

1.1 动机

计算机通迅系统在军队、政府和民用环境中扮演越来越重要的角色。本文主要集中注意在军队计算机通迅的要求，特别是存在通迅不可靠时的精力充沛及存在拥塞时的可用性，然而许多这些问题同样可在民用和政府部门发现。

当战略和战术计算机通迅网络被开发和部署时，其本质是提供内部连接的手段并提供可支持广泛应用的内部通迅协议。在预计到对这种标准的需求后，（美国）国防部研究和工程署代理副秘书长声称这里描述的传输控制协议（TCP）是 DoD 范围进程间通迅协议标准的基础。

TCP 是基于连接的，端到端可靠协议，设计来适合支持多网络应用的协议分层结构。TCP 在附加到截然不同但互相连接的计算机通迅网络的主机计算机内的进程对间提供可靠进程间通迅。对 TCP 层其下的通迅协议的可靠性作出了非常少的假设。TCP 假设它可以从下层协议获取简单的、潜在不可靠的数据报服务。原则上，TCP 应该能够运作通迅系统的广泛领域，其范围从固定有线连接到分组交换或线路交换网络。

TCP 基于首先由 Cerf 和 Kahn 在[1]中所描述的概念。TCP 适合分层协议结构，正好在基本网际协议[2]之上，它为 TCP 提供某种方式来发送和接收封装在互联网数据报“信封”内的可变长信息分段。互联网数据报提供手段来寻址不同网络中的源和目标 TCP。网际协议还处理要求完成通过多个网络和互联网关传输和递交的 TCP 分段的任何分片或重组。网际协议还挟带优先级信息，安全性证书和 TCP 分段的区分，因此该信息可以跨多个网络进行端到端

通迅。

协议分层

图 1

本文大部分是描写 TCP 实现的内容，它将与上层协议共同驻住在主机计算机中。某些计算机系统将通过掩盖 TCP 和网际协议层及网络特定软件的前端计算机连接到网络。TCP 规范描述对上层协议的接口，它似乎是可实现的，即使在前端情况下，只要实现一个适当的主机与前端协议。

1.2 范围

TCP 打算提供在多网络环境中的可靠进程对进程通迅服务。TCP 打算成为多个网络中的通用主机对主机协议。

1.3 关于本文

本文描述任何 TCP 实现所要求的行为规范，在其与上层协议的交互行为及在其与其它 TCP 终端的交互行为两个方面。本节的其它部分提供该协议的接口和操作的非常简短的见解。第 2 节总结 TCP 设置的哲理基础。第 3 节提供当各种事件发生时（新分段到达、用户呼叫、错误、等等）TCP 所要求的行为的详细描述和 TCP 分段的详细格式。

1.4 接口

TCP 接口一方面对用户或应用进程且另一方面对下层协议，如网际协议。

应用进程与 TCP 间的接口被合理仔细地描绘。该接口由呼叫集组成，便像操作系统为操控文件而提供给应用进程的呼叫。例如，有打开和关闭连接及在已建立连接上发送和接收数据的呼叫。还期望 TCP 可以与应用程序异步通迅。尽管允许 TCP 实现者有相当的自由度来设计接口，使其适合特定操作系统环境，对 TCP/用户接口的任何有效实现要求提供最小功能集。

TCP 和下层协议间的接口根本没有规定，除了假设存在某种机制使 2 层间相互可以异步

传递信息。通常，人们希望下层协议规定该接口。TCP 被设计来工作在非常普通的互连网络环境中。本文通篇假设下层协议是网际协议[2]。

1.5 操作

如上所述，TCP 的主要用途是在进程对间提供可靠、安全的逻辑线路或连接服务。为在缺少可靠性的互联网通迅系统上提供这种服务要求如下方面的机构：

基本数据传输
可靠性
流量控制
多路复用
连接
优先级和安全性

TCP 在每个这些领域中的基本操作在下面的段落中描述。

基本数据传输： TCP 能够在其用户间的每个方向上传输连接的字节流，通过分组一些数量的字节到通过互联网系统传输的分段中。通常，TCP 终端判断在它们自己方便时成块和转发数据。

某些时候用户需要确保他们提交给 TCP 的全部数据都已经传完了。因该用途，上推功能被定义。假设提交给 TCP 的数据实际上按发送方用户指示的方式传输，它应该被上推给接收方用户。上推引起 TCP 终端向接收方的位置迅速地传发和递交数据。确切的上推点可能对接收方用户不可见且上推功能不支持记录边界标记。

可靠性： TCP 必须从互联网通迅系统的损坏、丢失、重复或无序的数据中恢复过来。这可通过给每个要传输的字节分配序列号来达到，且要求 TCP 接收方的肯定确认（ACK）。若 ACK 在超时间隔内没有收到，则重传该数据。在接收方，该序列号用于正确排序可能无序到达的分段并消除重复。损坏可通过给每个传输分段增加一个校验和，在接收方检查它并丢弃已损坏分段来处理。

只要 TCP 终端继续正确运行且互联网系统不有完全分隔开，则没有传输错误会影响正确递交数据。TCP 可从互联网通迅系统错误中恢复。

流量控制： TCP 为接收方提供手段来管理由发送方发送的大量数据。这可通过在每个 ACK 中返回“窗口”来指示超过最后成功收到分段的可接受序列号的范围来完成。该窗口指出所允许发送方在收到进一步许可前可以传输的字节数。

多路复用：为允许单个主机内的多个进程同时使用 TCP 通迅工具，TCP 在每个主机中提供一组地址或端口集。与网络连接和来自互联网通迅层的主机地址，这形成一个套接口。一个套接口对唯一标识每个连接。即，一个套接口可以在多个连接中同时使用。

绑定到进程的端口由每个主机独自处理。然而，附加经常使用的进程（如，“登录器”或时间共享服务）到对公众所知的固定套接口将很有用。这时这些服务可以通过该已知地址被访问。建立和学习其它进程的端口地址将引入更动态的机制。

连接：上述可靠性和流量控制机制要求 TCP 终端为每个数据流初始化并维护某些状态信息。这些信息的组合，包括套接口、序列号和窗口大小被称为连接。每个连接由标识两端的套接口对唯一指定。

当 2 个进程希望通迅时，它们的 TCP 终端必须首先建立一个连接（在每端初始化这些状态信息）。当它们的能迅完成时，该连接终止或关闭，并释放这些资源给其它连接使用。

因连接必须在不可靠主机间通过不可靠互联网通迅系统被建立，使用基于时钟的序列号握手机制来避免连接的不正确初始化。

优先级和安全性：TCP 用户可以指出它们通迅的安全性和优先级。规定当这些特性不需要时使用缺省值。

2 哲理

2.1 互联网工程系统元素

互联网工程环境由连接到网络的主机组成，这些网络又返过来通过网关互联。这里假设这些网络可以是本地网（如，以太网）或大型网络（如，ARPA 网），但在许多情况下是基于分组交换技术。产生和消费消息的活动代理是进程。在这些网络中的各种层次协议、网关和支持进程间通迅系统的主机在进程端口间提供在逻辑连接上的双向数据流。

术语分组在这里通常用于指主机与其网络间的某个事务的数据。在网络内交换的数据块的格式通常将不我们所要关心的。

主机是附到网络上的计算机，且以通迅网络的观点，是分组的源端和目标端。进程被视作主机计算机内的活动元素(与将进程视为正在执行的程序这一非常通用的定义是一致的)。即使终端和文件或其它 I/O 设备都被视作可通过使用进程来相互间通迅。因此，全部通迅被视作进程间通迅。

因某个进程可能需要从几个通迅流中区别其自己和其它进程（或多个进程），我们认为每个进程可以有一些端口，通过它们与其它进程的端口通迅。

2.2 运行模型

进程通过调用 TCP 并传递数据缓冲区作为参数来传输数据。TCP 将数据从这些缓冲区中打包为分段并调用互联网模块来传递每个分段到目标端 TCP。接收方 TCP 从分段中将数据放入接收方用户的缓冲区中并提醒该接收方用户。TCP 终端包括控制这些分段中的信息，它们被用于确保可靠有序的数据传输。

互联网通迅模型是使用某个互联网协议模块关联到每个 TCP 上以提供接口给本地网络。该互联网模型打包 TCP 分段为互联网数据报并路由这些数据报到目标端互联网模块或中间网关。为通过本地网络传输该数据报，它被嵌入到本地网络分组中。

这些分组交换可能进一步执行打包、分片或其它操作以完成递交该本地分组到目标端互联网模块。

在网络间的网关处，互联网数据报是从其本地分组中“被展开的”且检查以判断该互联网数据报应用通过哪个网络来再次传输。该互联网数据报这时“被包装”到适合下个网络的本地分组中且路由到下个网关，或到达最终目标端。

允许某个网关将一个互联网数据报分解为多个更小的互联网分片，若这是通过下个网络传输的要求。为这样做，该网关产生一组互联网数据报；每个挟带一个分片。分片在后序网关中可以被进一步分解为更小的分片。设计互联网数据报分片格式以便目标端互联网模块可以重组分片为互联网数据报。

目标端互联网模块从数据报中（在重组为数据报后，若有必要）展开分段并传递它给目标端 TCP。

该简单运行模型通过许多细节来掩饰。一个非常重要的特性是服务类型。这提供信息给网关（或互联网模块）以指导它选择将在传到下个网络中所使用的服务参数。所包括的服务类型信息是数据报的优先级。数据报还可以挟带安全信息以允许运行于多级安全环境中的主机和网关为安全考虑而适当地隔离数据报。

2.3 主机环境

TCP 被假设为是操作系统中的一个模块。用户访问 TCP 更像他们将访问文件系统。TCP 可以调用其它操作系统功能，例如，以管理数据结构。网络的实际接口被假设为由设备驱动模块所控制。TCP 不直接调用网络设备驱动，但代之以调用互联网数据报协议模块，并可能由它代替调用该设备驱动。

TCP 机制并不排除在前端处理器中的 TCP 实现。然而，在这类实现中，某个主机到前端协议必须提供某种功能以支持在本文中描述的 TCP 用户类型接口。

2.4 接口

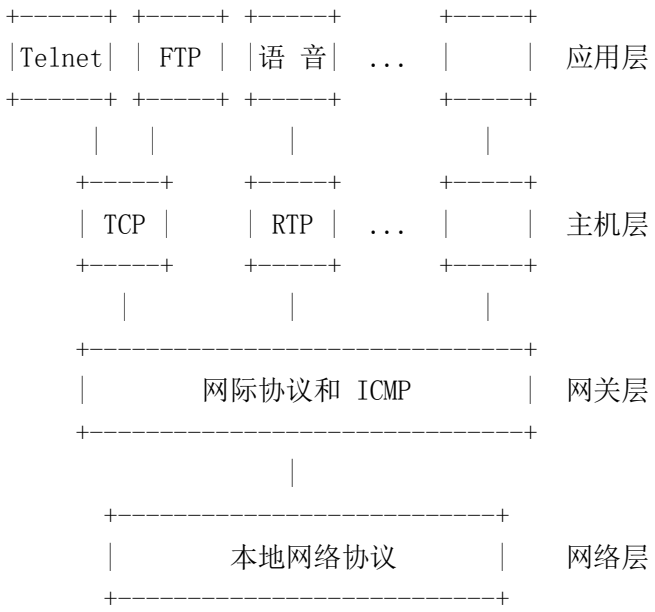
TCP/用户接口提供用户在 TCP 上所使用的调用来 OPEN 或 CLOSE 某个连接，SEND 或

RECEIVE 数据,或获取关于连接的 STATUS。这些调用很像操作系统上的用户程序的其它调用，例如，打开、读自、和关闭文件的调用。

TCP/互联网接口提供调用来发送和接收在互联网系统中的任何地方的主机中被寻址到 TCP 模块的数据报。这些调用有传递地址的参数，服务类型、优先级、安全性和其它控制信息。

2.5 相关其它协议

下图描绘 TCP 在协议层次中的位置：



协议关系

图 2

期望 TCP 将能够更有效地支持上层协议。它应该很容易连接上层协议如 ARPANET Telnet 或 AUTODIN II THP 到 TCP。

2.6 可靠通迅

TCP 连接上发送的数据流被可靠且有序地递交到目标端。通过使用序列号和确认来使传输可靠。概念上，数据的每个字节被分配一个序列号。分段中数据首字节的序列号与该分段一起传输并被称为分段序列号。分段还挟带确认号，它是反向上传输的下个期望数据字节的序列号。当 TCP 传输包含数据的分段时，它在重传队列中放置一份拷贝并开始定时；当对该数据的确认收到时，该分段从该队列删除。若在超时前该确认没有收到，则该分段被重传。

TCP 作出的确认不保证数据已经被递交到最终用户，但只是接收方 TCP 有责任这样做。

为管理 TCP 终端间的数据流，流量控制机制被部署。接收方 TCP 报告“窗口”给发送方 TCP。该窗口规定接收方 TCP 当前准备好接收的字节数，从确认号开始。

2.7 连接建立和清除

为识别 TCP 可能处理的独立数据流,TCP 提供端口标识符.因端口标识符由每个 TCP 单独选择，它们可能不是唯一的。为在每个 TCP 内提供唯一地址，我们联系标识 TCP 的互联网地址与端口标识符来创建套接口，它将在连接在一起的整个网络中是唯一的。

连接完全由终端上的套接口对来指定。本地套接口可以参与到与不同远端套接口的许多连接中。连接可以用于挟带双向数据，即，它是“全双工的”。

TCP 终端可自由关联端口与它们所自由选择的进程。然而，在任何实现中有几个必要的基本概念。必须存在 TCP 只通过某种手段与“适当的”进程所关联的众所周知的套接口。我们可想象，进程可能“拥有”端口，且该进程可能只在它们所拥有的端口上发起连接。（实现所有权的手段是本地问题，但我们想象是某种请求端口用户命令，或唯一地分配一组端口给给定进程的方法，如，通过分本端口名的高序位与给定进程。）

连接在通过本地端口和远端套接口参数的 OPEN 调用中指定。返回时，TCP 提供（简短的）本地连接名，用户可在后序列调用时引用它。有几件关于连接的事情必须提醒。为存储该信息，我们想象有称为传输控制块（TCB）的数据结构。某个实现策略将使该本地连接名是指向该连接的 TCB 的指针。OPEN 调用还规定连接建立是否将被主动追踪，或被动地等待。

被动 OPEN 请求意思是进程希望接受到达的连接请求，而不是尝式发起连接。经常是，请求被动 OPEN 的进程将接受任何呼叫者的连接请求。在该情况下，全空的远端套接口用于表示未指定套接口。未指定远端套接口只在被动 OPEN 中允许。

希望为未知其它进程提供服务的服务进程将以未指定远端套接口发出被动 OPEN 请求。这时任何进程可以发出连接以请求到该本地套接口的连接。它可能有帮助，若该本地套接口已知与该服务相关联。

众所周知套接口为优先关联某个套接口地址与某个标准服务而言是方便的机制。例如，“Telnet 服务器”进程被永久分配到特定套接口，且其它套接口被保留给文件传输、远程任务入口、文字生成器、回声器和 Sink 进程（最后 3 个正被用于测试用途）。套接口地址可能被保留以访问“查询”服务，它将返回将提供新建服务的特别套接口。众所周知套接口的概念是 TCP 规范的一部分，但为服务分配套接口超出本规范。（见[4]。）

进各可以发出被动 OPEN 并等待从其它进程的匹配主机 OPEN,且当连接已经建立时由 TCP 提醒。相互间同时发出主机 OPEN 的 2 个进程将会正确连接。该灵活性是支持组件相互间相关的异步行为的分布式计算关键点。

匹配在本地被动 OPEN 和远端主动 OPEN 间的套接口和 2 个主机情况。在第一个情况下，本地被动 OPEN 已经完全指定远端套接口。在该情况下，该匹配必须精确。在第二个情况下，

本地被动 OPEN 没有指定远端套接口。在该情况下，任何远端套接口一旦匹配本地套接口就是可接受的。其它可能性包括部分限制匹配。

若有多个挂起的被动 OPEN（记录在 TCB 中）有相同的本地套接口，则远端主动 OPEN 将与远端主动 OPEN 中规定的套接口的 TCB 相匹配，若该 TCB 存在的话，在选择没有指定远端套接口的 TCB 前。

建立连接的过程初始化同步（SYN）控制标志将引起 3 个消息的交换。该交换被称为 3 次握手[3]。

连接由包含 SYN 的到达分组与由用户 OPEN 命令各自创建的等待中 TCB 项的集合来发起。本地和远端套接口的匹配决定何时连接被发起。当序列号已经在双方都同步时，连接成为“已建立的”。

消息的清除还引起分段的交换，在该情况下挟带 FIN 控制标志。

2.8 数据通信

在连接上流动的数据可以被认为是字节流。发送方用户在每个 SEND 调用中指出该调用（和任何先决调用）中的数据是否应该通过设置 PUSH 标志立即被上推到接收方用户。

发送方 TCP 被允许从发送方用户收集数据并在其自己方便时以分段发送该数据，直到上推功能被提示，这时它必须发送全部未发送数据。当接收方 TCP 发现 PUSH 标志时，它禁止等待发送方 TCP 的更多数据，应立即将该数据传递给接收方进程。

在上推功能和分段界限间没有必然的关系。在任何特殊分段中的数据都可以引起单个 SEND 调用，全部地或部分地，或引起多个 SEND 调用。

上推功能和 PUSH 标志的用途是从发送方用户上推数据到接收方用户。它不提供记录服务。

需要组合上推功能与跨 TCP/用户接口的数据缓冲区的使用。每次 PUSH 标志与放入接收方用户缓冲区中的数据相关，该缓冲区被返回给用户来处理，即使该缓冲区没有填充。在发现 PUSH 前收到填充用户的缓冲区的数据，则该数据以缓冲区大小单位被传递给用户。

TCP 还提供手段来与数据的接收方通迅，该数据是紧急数据，在沿前数据流比接收方正读取更前面的某点处。TCP 不尝试定义在被提醒有紧急数据时用户要特别做什么，但一般看法是接收方进程将作出动作来快速处理紧急数据。

2.9 优先级和安全性

TCP 使用互联网协议服务类型域和安全选项来提供每个连接原则的优先级和安全性给 TCP 用户。不是所有 TCP 模块有必要运行在多级安全环境中；某些可能限制只有非认证用途，且其它的可能只运行在单安全级和隔离区上。因此，某些 TCP 实现和提供给用户的服务可能

被限制为多级安全情况的子集。

运行在多级安全环境中的 TCP 模块必须正确标志外出分段的安全性，隔离区及优先级。这类 TCP 模块还必须提供给它们的用户或上层协议如 Telnet 或 THP 接口来允许它们指定连接的希望安全级、隔离区和优先级。

2.10 精力充沛规则

TCP 实现将遵循普通精力充沛规则：对所做的事保守，对从其它处接收的东西自由。

3 功能性规范

3.1 头部格式

TCP 分段以互联网数据报发送。网际协议头部挟带几个信息域，包括源端和目标端主机地址[2]。TCP 头部跟着互联网头部，提供对 TCP 协议特别的信息。这种划分允许非 TCP 的主机层协议存在。

TCP 头部格式

TCP 头部格式
要注意一个记号标志代表一位位置。

图 3

源端端口：16 位

源端端口号。

目标端端口：16 位

目标端端口号。

序列号：32 位

本分段（除了当存在 SYN 时）首个数据字节的序列号。若存在 SYN 则序列号是补始序列号（ISN）且首个数据字节是 ISN+1。

确认号：32 位

若设置 ACK 控制位则该域包含分段发送方希望收到的下个序列号的值。一旦连接建立该域始终要发送。

数据偏移：4 位

TCP 头部的 32 位字数量。这指示数据从何处开始。TCP 头部（基于包含选项）长度是 32 位整数倍。

保留：6 位

保留给将来使用。必须为 0。

控制位：6 位（从左到右）

URG：重要的紧急指针域
ACK：重要的确认域
PSH：上推功能
RST：复位连接
SYN：同步序列号
FIN：发送方没有更多数据

窗口：16 位

始于该分段的发送方希望接受的在确认域中指示的数据字节数量。

校验和：16 位

校验和域是头部和文本的全部 16 位字的补码之和的 16 位补码。若分段中包含奇数个头部和文本字节用于计算校验和，则最后字节在其右填充一个 0 来形成 16 位字用于计算校验和。该填充不作为分段的一部份传输。当计算校验和时，校验和域自身置为 0。

该校验和还覆盖 96 位伪头部，该伪头部概念式地置于 TCP 头部前面。该伪头部包含源端地址、目标端地址、协议和 TCP 长度。这用于防止 TCP 分段误传。该信息被挟带于网际协议中，且跨 TCP/网络接口传输，作为 IP 上由 TCP 调用的参数或结果。

TCP 长度是 TCP 头部长度加上数据长度，以字节为单位（这不是个明确的传输数量，但是计算值），且它不计算伪头部的 12 个字节。

紧急指针：16 位

该域通知紧急指针的当前值作为从本段的序列号开始的正偏移。紧急指针指向下面的紧急数据的字节序列号。该域只在 URG 控制位设置的分段中被解释。

选项：可变

选项可占据 TCP 头部的后面的空间，且长度是 8 位的整数倍。全部选项都被包括在校验和中。选项可从任何字节边界开始。选项的格式有 2 种情况：

情况 1：选项类型单字节

情况 2：选项类型字节，选项长度字节和实际选项数据字节。

选项长度计算选项类型和选项长度的 2 个字节，同样包括选项数据字节。

要注意，选项列表可能比数据偏移域暗示的更短。头部内容超出结束选项的选项必须用头部填充（如，0）。

TCP 必须实现全部选项。

当前已定义选项包括（类型指示为八进制数）：

类型	长度	含义

0	-	结束选项列表。
1	-	无操作。
2	4	最大分段长度。

特别选项定义

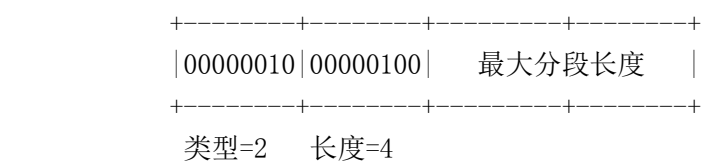
结束选项列表

该选项代码指出结束选项列表。这可能不与依据数据偏移域的 TCP 头部结尾一致。这用在所有选项的结尾处，而非每个选项的结尾处，且只需要在选项的结尾不会另外与 TCP 头部的结尾一致时使用。

无操作

该选项代码可以用于选项之间，例如，为了对齐后序选项的开始于字边界处。不保证发送方将使用该选项，后以接收方必须准备处理选项，即使它们不从字边界处开始。

最大分段长度



最大分段长度选项数据：16 位

若存在该选项，这时它通知该分段的 TCP 发送方的最大接收分段长度。该域必须只在于初始连接请求中（如，SYN 控制位设置的分段中）。若该选项没有使用，则任何分段长度都是允许的。

填充：可变

TCP 头部填充用于确保 TCP 头部结束于 32 位边界且数据开始于 32 位边界上。该填

充由 0 组成。

3.2 术语

在我们可以非常我地讨论 TCP 的操作前，我们需要引入一些详细术语。TCP 连接的维护要求记住几个变量。我们构思这些变量正存储在称为传输控制块或 TCB 的连接记录中。在这些存在于 TCB 的变量之中有本地和远端套接口数量，连接的安全性和优先级，用户发送和接收缓冲区的指针，重传队列和当前分段的指针。此外还有几个与发送和接收序列号相关的变量存储在 TCB 中。

发送次序变量

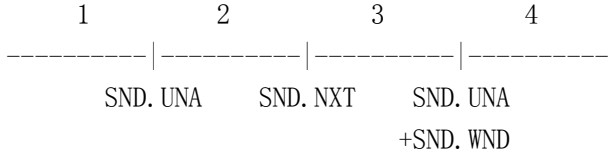
SND.UNA - 发送未确认
SND.NXT - 发送下个
SND.WND - 发送窗口
SND.UP - 发送紧急指针
SND.WL1 - 用于最后窗口更新的分段序列号
SND.WL2 - 用于最后窗口更新的分段确认号
ISS - 初始发送序列号

接收次序变量

RCV.NXT - 接收下个
RCV.WND - 接收窗口
RCV.UP - 接收紧急指针
IRS - 初始接收序列号

下图可帮助理清某些变量与次序空间的关系。

发送次序空间



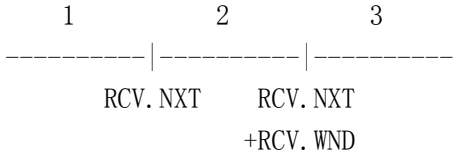
1 - 已经确认的旧序列号
2 - 未确认数据的序列号
3 - 允许新数据传输的序列号
4 - 还不允许的未来序列号

发送次序空间

图 4

发送窗口是图 4 标注为 3 的次序空间的部分。

接收次序空间



1 - 已确认旧序列号
2 - 允许新接收的序列号
3 - 还不允许的未来序列号

接收次序空间

图 5

接收窗口是图 5 中标注为 2 的次序空间的部分。

还有一些变量经常用于讨论中，它们的值作为当前分段的域。

当前分段变量

SEG.SEQ - 分段序列号
SEG.ACK - 分段确认号
SEG.LEN - 分段长度
SEG.WND - 分段窗口
SEG.UP - 分段紧急指针
SEG.PRC - 分段优选值

连接过程在其生存期内有一系列状态。这些状态是：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，因为它表示不存在 TCB 时的状态，且因此，没有连接。这些状态的简要含义是：

LISTEN - 表示从任何远端 TCP 和端口等待连接请求。

SYN-SENT - 表示在已发送连接请求后等待匹配的连接请求。

SYN-RECEIVED - 表示在已经接收和发送连接请求后等待确认连接请求确认。

ESTABLISHED - 表示已打开连接，接收数据可交给用户。连接数据传输阶段的正常状态。

FIN-WAIT-1 - 表示等待远端 TCP 的连接终止请求，或先前发送的连接终止请求的确认。

FIN-WAIT-2 - 表示等待远端 TCP 的连接终止请求。

CLOSE-WAIT - 表示等等本地用户的连接终止请求。

CLOSING - 表示等待远端 TCP 的连接终止请求确认。

LAST-ACK - 表示等待先前发送给远端 TCP 的连接终止请求的确认（包括其连接终止请求的确认）。

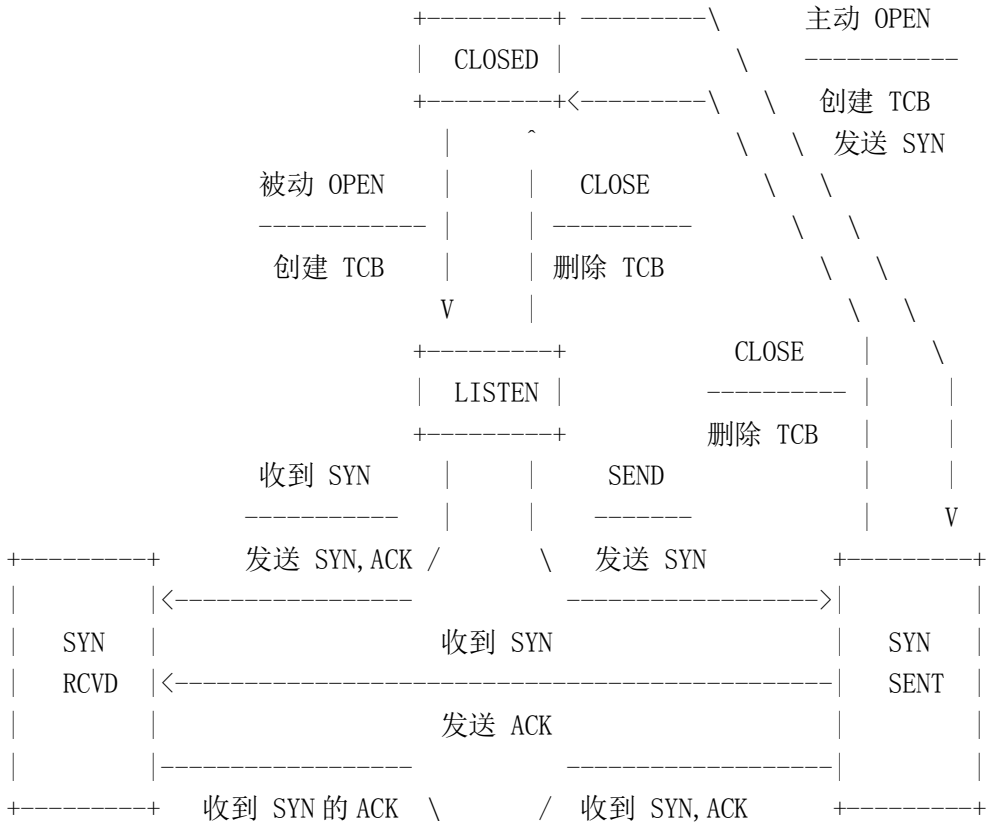
TIME-WAIT - 表示等待经过足够长时间以确保远端 TCP 收到其连接终止请求。

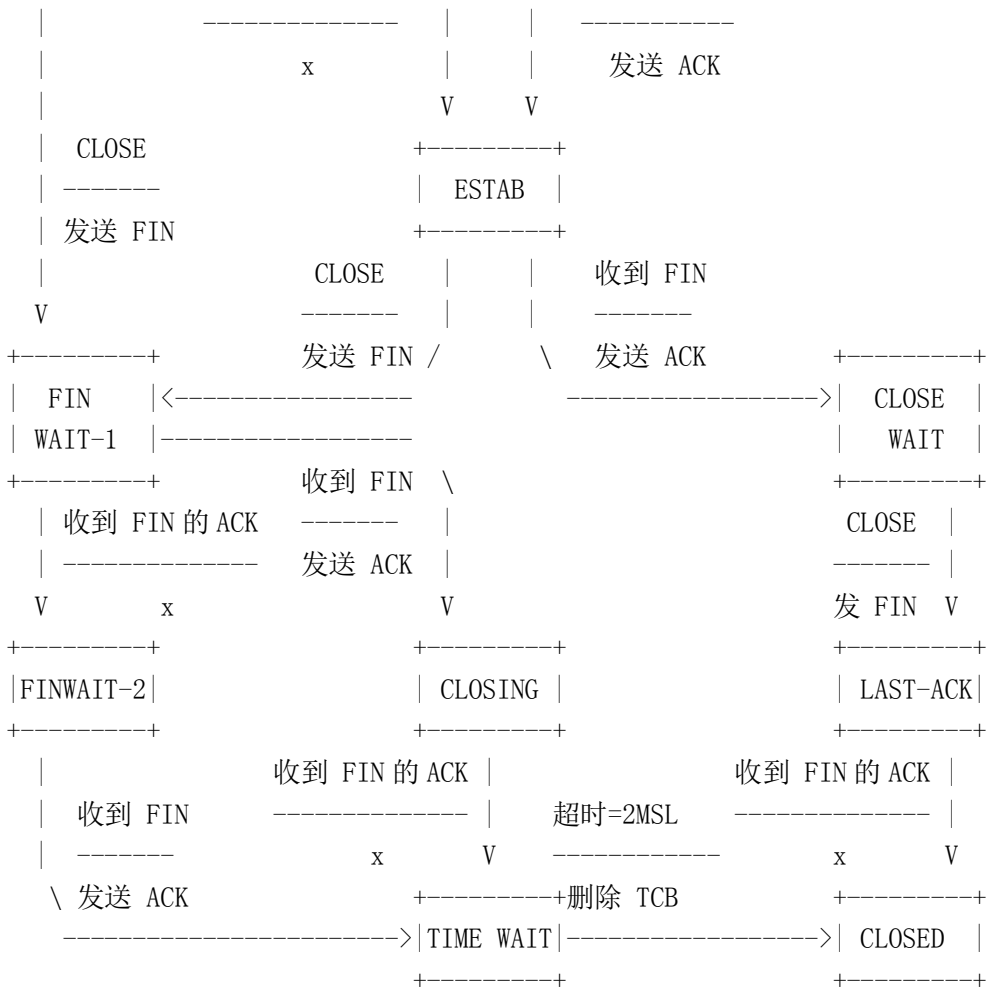
CLOSED - 表示根本无连接状态。

为响应事件，TCP 连接过程从一个状态到另一个。这些事件是用户调用，OPEN、SEND、RECEIVE、CLOSE、ABORT 和 STATUS；向内分段，特别是这些包含 SYN、ACK、RST 和 FIN 标志的；和超时。

图 6 中的状态图只描绘状态改变，同时引起的事件及结果行为，但没有说明不与状态改变相关的错误状态或行为。在后面章节中，提供有关 TCP 对事件的反应的更多细节。

注意祷告：该图只是个概述，且禁止作为整个规范。





TCP 连接状态图
图 6

3.3 序列号

在设计中的基本概念是通过 TCP 连接发送的每个字节都有一个序列号。因为每个字节被编号，所以它们每个都可以被确认。部署的确认机制是累积的，因此，序列号 X 的确认指出直到但不包括 X 的所有字节都已经收到。该机制允许在有重传时的直接转发重复检测。在分段中的字节数是紧跟着头部的首个数据字节的最小数，且下面字节接着编号。

要记住的实质是实际序列号空间是有限的，尽管非常大。该空间范围从 0 到 2**32-1。因该空间是有限的，对序列号的全部算术必须执行模 2**32。该无符号算术保持序列号的关系当它们再次从 2**32-1 到 0 循环时。计算模算术有些微妙，因此在编程比较这些值时应该非常小心。符号“=<”意思是“小于或等于”（模 2**32）。

TCP 必须执行的序列号比较的一般类型包括：

（a）判断确认表示已发送但未确认的序列号。

（b）判断分段占用的全部序列号已经确认（如，要从重传队列吕删除该分段）。

（c）判断向内分段包含所期望的序列号（如，该分段“覆盖”接收窗口）。

作为对发送数据的响应，TCP 将收到确认。下列比较需要用于处理确认。

SND.UNA = 旧的未确认序列号

SND.NXT = 下个要发送序列号

SEG.ACK = 接收 TCP 的确认（接收 TCP 期望的下个序列号）

SEG.SEQ = 分段的首个序列号

SEG.LEN = 分段中的数据所占据的字节数（计算 SYN 和 FIN）

SEG.SEQ+SEG.LEN-1 = 分段的最后一个序列号

新确认（称为“可接受确认”），是满足如下不等式的：

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

重传队列上的分段是完全确认的，若它的序列号的和和长度小于或等于到达分段的确认值。

当收到数据时，需要下列比较：

RCV.NXT = 期望到达分段的下个序列号，且是接收窗口的左边或最低边沿

RCV.NXT+RCV.WND-1 = 期望到达分段的最后序列号，且是接收窗口的右边或最高边沿

SEG.SEQ = 到达分段占据的首个序列号

SEG.SEQ+SEG.LEN-1 = 到达分段占据的最后序列号

判断分段占据部分有效接收序列空间，若

RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND

或

RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

该测试的第一部分检测该分段的开始是否落入窗口中，该测试的第二部分检查该分段的

结尾是否落入窗口中；若该分段该测试的通过任何部分，则它包含数据在窗口中。

实际情况比这还要复杂点。因 0 窗口和 0 长度分段，我们对到达分段的可接受性有 4 种情况：

分段长度	接收长度	测试窗口
0	0	SEG.SEQ = RCV.NXT
0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND
>0	0	不可接受
>0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND 或 RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

要注意，当接收窗口是 0 时，没有分段应该被接受除了 ACK 分段。因此，TCP 可能维护 0 接收窗口同时传输数据并接收 ACK。然而，即使当接收窗口是 0，TCP 必须处理所有到达分段的 RST 和 URG 域。

我们获得了用同样的数字方案来保护某些控制信息的好处。这通过在序列空间中暗含某些控制标志来做到，因此它们可以被传输且不会混淆确认（如，该控制的一个且仅有一个拷贝将被执行）。控制信息不是在分段数据空间本身中挟带的。因此，我们必须采用规则来暗中分配序列号给控制。SYN 和 FIN 是唯一需要这种保护的控制（标志），且这些控制（标志）只用在连接打开和关闭时。对序列号用途，SYN 被认为占据在分段的首个实际数据字节所占据的（序列号）之前，同时 FIN 被认为占据在分段的最后实际数据数据字节所占据的（序列号）之后。分段长度（SEG.LEN）包括数据和控制（标志）占据的序列号空间。当 SYN 存在时，这时 SEG.SEQ 是 SYN 的序列号。

初始序列号选择

本协议不限制特殊连接被多次使用。连接通过套接口对来定义。连接的新实例将被作为该连接的化身。这将会引起的问题是“TCP 如何识别该连接以前化身的重复分段？”该问题变得很明显，若连接很快地打开和关闭，或如果连接因内存丢失而中断且这时重新建立。

为避免混淆，我们必须阻止来自某个某个连接的一个化身的分段被使用，当来自较早化身的相同序列号可能依然存在于网络中。我们希望确保，即使 TCP 崩溃且丢失它已经使用的序列号的全部信息。当创建新连接时，初始序列号（ISN）生成器被部署来选择新的 32 位 ISN。该生成器被绑定到（可能是虚构的）32 位时钟，其低序位大概每隔 4 微秒就会增加。因此，ISN 周期大约每 4.55 小时。因为我们假设分段将在网络中停留不超过最大分段生存时间（MSL）且该 MSL 小于 4.55 小时，我们可以可靠地假设该 ISN 将是唯一的。

对于每个连接，都有一个发送序列号和一个接收序列号。初始发送序列号（ISS）由数

据发送 TCP 选择，且初始接收序列号（IRS）在连接建立过程期间获得。

对于将建立或发起的连接，2 个 TCP 终端必须同步双方的初始序列号。这可通过交换连接建立分段，并在其中挟带称为“SYN”（为了同步）的控制位和初始序列号。作为速记符号，挟带 SYN 位的分段也称为“SYN 分段”。因此，该方案要求适合的机制来选择初始序列号和有点复杂的握手来交换 ISN。

这种同步化要求每方发送其自己的初始序列号从另一方并接收对它的确认。每方还必须接收其它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 可以组合为单个消息，这被称为 3 次（或 3 消息）握手。

三次握手是必须的，因为序列号没有约束到网络中的全局时钟，且 TCP 终端可以有不同的机制还选择 ISN。首个 SYN 的接收方没办法知道该分段是否是旧的延迟分段，除非它知道用于该连接的最后序列号（而这不是始终可能的），且因此它必须询问发送方验证该 SYN。三次握手和时钟驱动机制的优势在[3]中有讨论。

知道何时保持静默

确保 TCP 不创建挟带有可能由停留在网络中的旧分段所重复的序列号的分段，TCP 必须在初始或从内存中正使用序列号丢失的崩溃中恢复时分配任何序列号之前保持静默一个最大分段存活期（MSL）。本规范规定 MSL 是 2 分钟。这是个工程选择，且可能会修改，若经验指出希望这样做。要注意，若 TCP 因某种原因重初始化，尽管保留其使用的序列号内存，这时它不需要等待；它仅需确保使用序列号比最近所使用的大。

TCP 静默时间概念

本规范保证，“崩溃”的主机不需要保留在每个活动（如，非关闭）连接上的最后所传输序列号的任何知识，且应该在该主机是一部分的互联网系统中延迟发出任何 TCP 分段至少该一致的最大分段存活期（MSL）。在下面的段落中，给出了对该规定的解释。TCP 实现者可能违反“静默时间”约束，但仅冒引起某些旧数据被接受为新的或新的数据作为被互联网系统中的某些接收方重复的旧的而被拒绝的风险。

TCP 终端每次在源主机处转发分段并进入网络输出队列时消费序列号。TCP 协议中的该重复检测和编号算法依靠分段数据统一绑定到扩展该序列号在绑定到该序列号的分段数据已经被递交并由接收方确认且该分段的全部重复拷贝已经从互联网中“耗尽”前将不会循环直到全部 2**32 值的序列空间。离开这种假设，2 个不同的 TCP 分段可能确信被分配相同或重叠序列号，引起接收方混淆哪个数据是新的且哪个是旧的。要记住，每个分段如该分段中的数据字节一样被绑定到尽可能连续的序列号。

在正常条件下，TCP 终端保持跟踪下个嵌入和最后等待确认的序列号，以避免在其首次使用被确认前错误使用重叠的序列号。仅这点不能保证旧重复数据从网络中耗尽，因此序列号空间被定义得很大以减少徘徊重复将引起接收问题的可能性。用 2Mb/s 速率,它将花 4.5 小时来用尽序列空间的 2**32 个字节.由于网络中的最大分段存活期不会超过几十秒，可认为这是种对可预测网络的充分保护，甚至如果数据率逐惭升高到数十 Mb/s。当速率为 100Mb/s 时，循环时间是 5.4 分钟，它可能有点短，但依然可能。

TCP 中的基本重复检测和编号算法可能被击败，然而，若源 TCP 没有记忆在某个给定连接上它最后使用的任何序列号。例如，若 TCP 以序列号 0 开始全部连接，这时因崩溃和重启，TCP 可能重组较早的连接（可能在半开连接决定后）并嵌入与依然在网络中的分组相同的或覆盖的序列号分组，后面的分组嵌入到同个连接的较早化身上。在缺少关于在特定连接上所使用的序列号的信息时，TCP 规范推荐源端在连接上发出分段前延迟 MSL 秒，以允许较早连接上的分段化身有足够时间从系统中耗尽。

致使主机可以记住时间并使用它来选择初始序列号值，它也不能避免该问题（如，即使时间用于为每个新连接化身选择初始序列号）。

假设，例如，该连接以序列号 S 开始打开。假设该连接不经常使用且该初始序列号功能（ISN（t））偶然地使用与序列号，称为 S1，相同的值，S1 是该 TCP 在特定连接上发送的最后分段。现在假设，在本例中，该主机崩溃，恢复并建立该连接的新化身。所选择的初始序列号是 S1=ISN（t）——在旧连接化身上最后所使用的序列号！若恢复发生足够快，网络中在 S1 邻居中相关序列号的任何旧复本可能到达并被该连接的新化身的接收方作为新分组。

问题是恢复主机不可能知道它崩溃了多长也不知道系统中是否依然存在较早连接化身的旧复本。

解决该问题的一个方法是在从崩溃恢复后故意延迟发送分段 1 个 MSL——这就是“静默”规范。选择避免等待的主机愿意冒可能在给定目的端上混淆旧和新分组，它可以选择不等待“静默时间”。实现者可以向 TCP 用户提供可以在连接上选择，通过选择在崩溃后是否等待的连接根据，或者可以为全部连接非正式实现“静默时间”。明显在，甚至在用户选择“等待”的地方，在该主机已经“上线”至少 MSL 秒后这也不是必要的。

结论：每个发出的分段占用在序列号空间中的一个或多个序列号，被分段所占用的数字是“忙的”或“使用中”，直到已经过 MSL 秒，因空时的崩溃块被最后发出的分段的字节所占用，若新连接开始得太快且使用以前连接化身的最后分段的空时足迹中的任何序列号，这里有潜在序列号覆盖区域，这可能引起接收方混淆。

3.4 建立连接

“三次握手”用于建立连接的过程。该过程正常由某方 TCP 发起并由另一方 TCP 响应。该过程也有效，若两方 TCP 同时发起该过程。当同时尝试时，每端 TCP 接收到“SYN”分段，它没有在已经发送“SYN”后挟带确认。当然，旧的重复“SYN”分段的到达可能潜在使它发生，给接收方，即同时连接发起正在处理。正确使用“复置”分段可以消除这些问题。

几个连接发起例子如下。尽管这些例子不使用数据挟带分段来显示连接同步，这是完全合法的，以至于接收方 TCP 不递效数据给用户，直到它明确该数据是有效的（如，该数据必须在接收方缓存，直到该连接进入 ESTABLISHED 状态）。三次握手降低连接失败的可能性。为该检测提供信息的内存和消息由实现约定。

最简单的三次握手在下面的图 7 中显示。该图应该以如下方式解释。每行为引用目的编号。右箭头(-->)指出 TCP 分段从 TCP A 出发到 TCPB，或 B 处的分段从 A 到达。左箭头(<--)，指出相反含义。省略号（...）指出依然在网络中的分段（延迟了）。“XXX”指出丢失或被拒绝的分段。注释出现在括号中。TCP 状态表示分段（其内容在每行中间显示）出发或到达的 AFTER 状态。分段内存显示为简短形式，有序列号、控制标志、和 ACK 域。其它域，如窗口、地址、长度和文本已经没有明显的兴趣。

```
      TCP A                                                TCP B

1.  CLOSED                                               LISTEN

2.  SYN-SENT    --> <SEQ=100><CTL=SYN>               --> SYN-RECEIVED

3.  ESTABLISHED <-- <SEQ=300><ACK=101><CTL=SYN,ACK>  <-- SYN-RECEIVED

4.  ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK>      --> ESTABLISHED

5.  ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK><DATA> --> ESTABLISHED
```

连接同步的基本三次握手

图 7

在图 7 中的第 2 行，TCP A 开始发送 SYN 分段，指出它将使用序列号 100 开始。在第 3 行中，TCP B 发送 SYN 和它从 TCP A 接收到的确认 SYN。要注意确认域指出 TCP B 现在希望听到序列号 101，确认 SYN 占用序列号 100。

在第 4 行，TCP A 响应空分段，包含对 TCP B 的 SYN 的 ACK；且在第 5 行中，TCP A 发送一些数据。要注意，在第 5 行中的分段序列号与第 4 行中的相同，因 ACK 不占用序列号空间（若它这样做，我们将会触发确认 ACK 的!）。

同步初始化只是有些更复杂，如图 8 中所示。每个 TCP 循环从 CLOSED 到 SYN-SENT 到 SYN-RECEIVED 到 ESTABLISHED。

```
      TCP A                                                TCP B

1.  CLOSED                                               CLOSED
```

```
2.  SYN-SENT      --> <SEQ=100><CTL=SYN>                ...

3.  SYN-RECEIVED <-- <SEQ=300><CTL=SYN>                <-- SYN-SENT

4.                ... <SEQ=100><CTL=SYN>                --> SYN-RECEIVED

5.  SYN-RECEIVED --> <SEQ=100><ACK=301><CTL=SYN,ACK> ...

6.  ESTABLISHED  <-- <SEQ=300><ACK=101><CTL=SYN,ACK> <-- SYN-RECEIVED

7.                ... <SEQ=101><ACK=301><CTL=ACK>      --> ESTABLISHED
```

同时连接同步

图 8

三次握手的原则因素将阻止旧重复连接初始化引起混淆。为解决它，特殊控制消息，重置，已经被发明。若接收 TCP 在非同步状态（如，SYN-SENT、SYN-RECEIVED），它接收到可接收重置时返回到 LISTEN。若 TCP 在某种同步状态下（ESTABLISHED、FIN-WAIT-1、FIN-WAIT-2、CLOSE-WAIT、CLOSING、LAST-ACK、TIME-WAIT），它断开连接并提醒其用户。我们在后面讨论该情况，以下面“半开”连接时。

```
    TCP A                                                TCP B

1.  CLOSED                                               LISTEN

2.  SYN-SENT     --> <SEQ=100><CTL=SYN>                ...

3.  (duplicate) ... <SEQ=90><CTL=SYN>                --> SYN-RECEIVED

4.  SYN-SENT     <-- <SEQ=300><ACK=91><CTL=SYN,ACK>  <-- SYN-RECEIVED

5.  SYN-SENT     --> <SEQ=91><CTL=RST>                --> LISTEN

6.                ... <SEQ=100><CTL=SYN>                --> SYN-RECEIVED

7.  SYN-SENT     <-- <SEQ=400><ACK=101><CTL=SYN,ACK>  <-- SYN-RECEIVED

8.  ESTABLISHED --> <SEQ=101><ACK=401><CTL=ACK>       --> ESTABLISHED
```

从旧重复 SYN 中恢复

图 9

作为从旧复本中恢复的一个简单例子，考虑图 9。在第 3 行中，旧重复 SYN 到达 TCP B。TCP B 不能说出这是旧复本，因此它正常响应（第 4 行）。TCP A 检测到 ACK 域是不正确的并返回 RST（重置）且其所选择的 SEQ 域使该分段可信任。TCP B，在收到 RST 时，返回到 LISTEN 状态。在 RST 之前，RST 在双向上发送的更复杂交换可能已经发生。

半开连接和其它异常

已建立连接称为“半开”，若 TCP 某方已经在其这端关闭或中断该连接而不管另一方，或者若连接的两端已经成为非同步将引起崩溃，其结果是内存丢失。该连接将自动开始重置，若作出尝试以在双向上发送数据。然而，半开连接预期是非正常的，且恢复过程被适度引入。

若在地点 A 该连接不在存在，这时在地点 B 的用户作出在其上尝试发送任何数据将引起地点 B 的 TCP 收到重置控制消息。该消息指出地点 B 的 TCP 有些问题，且它希望中止该连接。

假设 2 个用户进程 A 和 B 与另一个进行通迅，当崩溃的发生引起 A 的 TCP 的内存丢失。取决于支持 A 的 TCP 的操作系统，它似乎是存在某些错误恢复机制。当 TCP 再次重启时，A 似乎将再次从头或从恢复点开始。作为结果，将可能试图再次 OPEN 该连接或尝式在它认为是打开的连接上 SEND。在后种情况，它从本地（A 的）TCP 收到错误消息“连接未打开”。在尝式建立连接时，A 的 TCP 将发送包含 SYN 的分段。这种情形引出图 10 中显示的例子。在 TCP A 崩溃后，用户尝试重打开该连接。TCP B，在其间，认为该连接是打开的。

```
     TCP A                                          TCP B

1.   (CRASH)                                  (发送 300，接收 100)

2.   CLOSED                                         ESTABLISHED

3.   SYN-SENT --> <SEQ=400><CTL=SYN>                --> (??)

4.   (!!)      <-- <SEQ=300><ACK=100><CTL=ACK>      <-- ESTABLISHED

5.   SYN-SENT --> <SEQ=100><CTL=RST>                --> (Abort!!)

6.   SYN-SENT                                       CLOSED

7.   SYN-SENT --> <SEQ=400><CTL=SYN>                -->
```

半开连接发现

图 10

当 SYN 到达第 3 行时，TCP B，正处在同步状态，且输入分段超出窗口，以某个确认响应指出它希望收到的下个序列号（ACK 100）。TCP A 发现该分段不确认它发送的任何东西且，正是非同步的，发送重置（RST）因为它已经检测到半开连接。TCP B 中止于第 5 行。TCP A

将继续尝式建立该连接；现在的问题是减少图 7 的基本 3 次握手。

所关心的另一种情况发生在 TCP A 崩溃且 TCP B 尝试在它所认为是同步的连接上发送数据时。这在图 11 中描述。在这种情况下，从 TCP B（第 2 行）到达 TCP A 数据是不可接受的，因不存在这种连接，因此 TCP A 发送 RST。该 RST 是可接受的，以使 TCP B 处理它并中止该连接。

```
      TCP A                                                TCP B

1.  (CRASH)                                          (发送 300,接收 100)

2.  (??)    <-- <SEQ=300><ACK=100><DATA=10><CTL=ACK> <-- ESTABLISHED

3.          --> <SEQ=100><CTL=RST>                   --> (ABORT!!)
```

活动端引直半开连接发现

图 11

在图 12 中，我们发现 2 端 TCP A 和 B 都使用被动连接来等待 SYN。旧复本到达 TCP B（第 2 行）引起 B 活动。SYN-ACK 被返回（第 3 行）且引起 TCP A 生成 RST（在第 3 行中的 ACK 不可接受）。TCP B 接受该重置并返回它的被动 LISTEN 状态。

```
      TCP A                                                TCP B

1.  LISTEN                                               LISTEN

2.        ... <SEQ=Z><CTL=SYN>                  -->  SYN-RECEIVED

3.  (??) <-- <SEQ=X><ACK=Z+1><CTL=SYN,ACK>      <--  SYN-RECEIVED

4.        --> <SEQ=Z+1><CTL=RST>                -->  (return to LISTEN!)

5.  LISTEN                                               LISTEN
```

旧重复 SYN 在 2 个被动套接口上发出重置

图 12

多种其它情况是可能的，它们全部遵循如下关于 RST 生成和处理的规则。

重置生成

作为通用规则，重置（RST）必须被发送，无论何时分段到达，即使它显然不是当前连

接所希望的。重置禁止被发送若该情况下明显的话。

这里有 3 组状态：

1、若该连接不存在（CLOSED）这时重置被发送以响应除另外重置之外的任何输入分段。特别是，SYN 寻址到不存在连接在这种情况下会被拒绝。

若办输入分段有 ACK 域，则重置将使用该分段的 ACK 域中的序列号，否则重置使用序列号 0 且 ACK 域设置为序列号与输入分段的分段长度的和。该连接保持为 CLOSED 状态。

2、若该连接进入任何非同步状态（LISTEN、SYN-SENT、SYN-RECEIVED），且输入分段确认某些还未发送的（该分段挟带不可接受 ACK），或者若输入分段有安全级别或隔离间，它不精确匹配连接所请求的级别和隔离间，则重置被发送。

若我们的 SYN 未经确认且输入分段的优先级高于被请求的优先级，这时都会提升本地优先级（若被用户和系统所允许）或发送重置；或者若输入分段的优先级低于所请求的优先级，这时继续如同该优先级精确匹配（若远端 TCP 不能提升优先级以匹配我们的，这将在它发送下个分段中被检测到，且连接这时将会中断）。若我们的 SYN 已经被确认（可能在该输入分段中），输入分段的优先级必须精确匹配本地优先级，若它不必发送重置。

若输入分段有 ACK 域，重置从该分段的 ACK 域中取出它的序列号，否则重置使用序列号 0 且 ACK 域设置为序列号与输入分段的分段长度的和。该连接维持相同的状态。

3、若连接处于同步状态（ESTABLISHED、FIN-WAIT-1、FIN-WAIT2、CLOSE-WAIT、CLOSING、LAST-ACK、TIME-WAIT），任何不可接受分段（超出窗口序列号或不可接受确认号）必须只引出空确认分段包含当前发送序列号和指出希望接收的下个序列号的确认，并且该连接维持相同的状态。

若输入分段有安全级，或者隔离区，或者不精确匹配的优先级，和隔离区，和连接所请求的优先级，重置被发送且连接变为 CLOSED 状态。重地从输入分段的 ACK 域中使用其序列号。

重置处理

在除 SYN-SENT 以外的全部状态中，所有重置（RST）分段都是有效的，通过检查其 SEQ 域。重置是有效的，若其序列号在窗口中。在 SYN-SENT 状态（RST 被收到以响应初始 SYN）中，RST 是可接受的，若 ACK 域确认该 SYN。

RST 的接收方首先难它，这里修改状态。若接收方处于 LISTEN 状态，它乎略它。若接收方处于 SYN-RECEIVED 状态且之前处于 LISTEN 状态，这时接收方返回到 LISTEN 状态，否则接收方中断该连接并进入 CLOSED 状态。若接收方处于其它状态，它中止连接并告知用户并进入 CLOSED 状态。

3.5、关闭连接

CLOSE 是个操作，意思是“我没有更多要发送的”。关闭全双工连接的术语是有歧义的，当然，因它可能不明确如何对待连接的接收端。我们已经先择视 CLOSE 为单工方式。CLOSE 的用户可以继续 RECEIVE，直到他被告诉另一端也已经 CLOSED，且这时继续 RECEIVE 直到被信号中断，即 RECEIVE 失败因另一端已经 CLOSED。我们假设 TCP 将给用户发送信号，即使没有 RECEIVE 是明显的，即另一端已经关闭，因此用户可以有好地中止它这端。TCP 将可靠地在连接变为 CLOSED 前递交全部 SENT 缓存，因此用户希望不返回数据只需要等待接收连接成功 CLOSED 就知道其全部数据都被目的 TCP 所接收。用户必须保持读取它们发送关闭的连接直到 TCP 说没有更多数据。

有 3 种本质不同的情况：

1）用户通过告诉 TCP CLOSE 连接来发起

2）远端 TCP 通过发送 FIN 控制信号来发起

3）双方用户同时 CLOSE

情况 1：本地用户发出关闭

在该情况下，FIN 分段可以被构造且放在外出分段队列中。没有更多从用户 SEND 的将会被 TCP 所接受，且它进入 FIN-WAIT-1 状态。RECEIVE 允许这种状态。之前并包括 FIN 的全部分段将被重传直到被确认。当另一方 TCP 有确认的 FIN 并发送它自己的 FIN，首个 TCP 可以 ACK 该 FIN。要注意，接收 FIN 的 TCP 将 ACK，但不发送其自己的 FIN，直到其用户也 CLOSED 该连接。

情况 2：TCP 从网络收到 FIN

若未请求的 FIN 从网络到达，则接收方 TCP 可以 ACK 它并告诉用户该连接正关闭。用户将响应以 CLOSE，基于 TCP 可以在发送任何剩余数据后发送 FIN 给其它 TCP。TCP 这时等待直到其自己 FIN 被确认因此它删除该连接。若 ACK 不来临，在用户超时后该连接被中止且用户被告知。

情况 3：双方用户同时关闭

在连接两端的用户同时 CLOSE 会引起 FIN 分段被交换。当先于 FIN 的全部分段已经被处理并确认时，每端 TCP 可以 ACK 它已经接收的 FIN。双方将，基于接收这些 ACK，删除该连接。

	TCP A	TCP B
1.	ESTABLISHED	ESTABLISHED
2.	(关闭)	

```
      FIN-WAIT-1  --> <SEQ=100><ACK=300><CTL=FIN,ACK>  --> CLOSE-WAIT

3.    FIN-WAIT-2  <-- <SEQ=300><ACK=101><CTL=ACK>      <-- CLOSE-WAIT

4.                                                        (关闭)
      TIME-WAIT   <-- <SEQ=300><ACK=101><CTL=FIN,ACK>  <-- LAST-ACK

5.    TIME-WAIT   --> <SEQ=101><ACK=301><CTL=ACK>      --> CLOSED

6.    (2 MSL)
      CLOSED
```

正常关闭顺序

图 13

```
      TCP A                                                TCP B

1.    ESTABLISHED                                          ESTABLISHED

2.    (关闭)                                               (关闭)
      FIN-WAIT-1  --> <SEQ=100><ACK=300><CTL=FIN,ACK>  ... FIN-WAIT-1
                  <-- <SEQ=300><ACK=100><CTL=FIN,ACK>  <--
                  ... <SEQ=100><ACK=300><CTL=FIN,ACK>  -->

3.    CLOSING     --> <SEQ=101><ACK=301><CTL=ACK>      ... CLOSING
                  <-- <SEQ=301><ACK=101><CTL=ACK>      <--
                  ... <SEQ=101><ACK=301><CTL=ACK>      -->

4.    TIME-WAIT                                            TIME-WAIT
      (2 MSL)                                              (2 MSL)
      CLOSED                                               CLOSED
```

同时关闭顺序

图 14

3.6、优先级和安全性

其意图是连接只在与确定相同的安全性和隔离区值操作的端口间被允许且处于被双方端口所请求更高的优先级。

TCP 中使用的优先级和安全性参数如同在网际协议（IP）[2]中所定义的。通篇 TCP 规范的术语“安全性/隔离区”用于指出在 IP 中所使用的安全性参数包括安全性、隔离区、用户组、

和处理约束。

连接尝试失配安全性/隔离区值或更低优先级值必须以发送重置来拒绝。拒绝因太低优先级的连接只发生在 SYN 的确认已经被收到后。

要注意 TCP 模块只运行于缺省的优先级值，将依然必须检查输入分段的优先级且可能提升它们在该连接上所使用的优先级。

安全性参数可参被使用，即使在非安全环境中（该值将指出非认证数据），因此在非安全环境中的主机必须准备接收安全性参数，尽管它们不需要发送它。

3.7、数据通迅

一旦连接被连接，数据通过交换分段来通迅。因分段可能因某种问题（检验和测试失败）而丢失，或者网络拥塞，TCP 使用重传（在超时后）来确保递交每个分段。重复分段可能到达，因网络或 TCP 重传。如关于序列号的章节中的讨论的，TCP 对分段中的序列号和确认号执行一定的测试来验证其可接受性。

数据的发送方在变量 SNT.NXT 中持续跟踪将使用的下个序列号。数据的接收方在变量 RCV.NXT 中保持跟踪希望的下个序列号。数据的发送方在变量 SND.UNA 中保持跟踪最近未确认序列号。若数据流即刻空闲且全部发送的数据被确认，这时这 3 个变量将相等。

当发送方创建分段并传输它发送方求助于 SND.NXT。当接收方收到分段时它求助于 RCV.NXT 并发送确认。当数据发送方收到确认时它求助于 SND.UNA。扩展到与这些变量不同的值是在通迅中延迟的尺度。所求助的变量的总数是分段中数据的长度。要注意，一旦在 ESTABLISHED 状态下，全部分段必须挟带当前确认信息。

CLOSE 用户呼叫实现上推功能，如同对输入分段中的 FIN 控制标志所做的。

重传超时

因为组成互联网系统的网络的可变性和 TCP 连接的大范围使用，重传超时必须动态地判断。判断重传超时的一个过程在这里给出描述。

一个重传超时过程的例子

测量以特定序列号发送数据字节和接收确认间的流逝时间，该确认覆盖该序列号（所发送分段不必匹配所收到的分段）。该测量流逝时间是往返时间（RTT）。下步计算平滑往返时间（SRTT）如：

SRTT = (ALPHA * SRTT) + ((1-ALPHA) * RTT)

且基于这点，计算重传超时（RTO）如：

RTO = min[UBOUND,max[LBOUND,(BETA*SRTT)]]

这里的 UBOUND 是超时的上边界（如，1 分钟），LBOUND 是超时的下边界（如，1 秒钟），ALPHA 是个平滑因子（如，.8 到.9），且 BETA 延迟变化因子（如，1.3 到 2.0）。

紧急信息的通迅

TCP 紧急机制的目标是允许发送用户刺激接收用户接受某些紧急数据并允许接收 TCP 指出接收用户，当全部当前已知紧急数据已经被用户所接收。

该机制允许数据流中的某点被指定为紧急信息的终端。无论何时该点将在接收方 TCP 提前接收序列号（RCV.NXT），该 TCP 必须告诉用户进入“紧急模式”；当接收序列号到达紧急指针时，TCP 必须告诉用户进入“正常模式”。若紧急指针被更新，同时用户处于“紧急模式”时，更新将对用户不可见。

该方法使用紧急域，它挟带在全部传输分段中。URG 控制标志指出该紧急域是有意义的且必须被增加到分段序列号以产生紧急指针。缺少该标志指出不存在明显的紧急数据。

发送紧急指示，用户还必须发送至少 1 个数据字节。若发送用户还指出上推，及时递交紧急信息给目标进程是增强的。

管理窗口

在每个分段中发送的窗口指出发送方窗口（数据接收方）的序列号范围当前被准备以接受。这里还假设这与对该连接有效的当前可用数据缓存空间相关。

指出大窗口鼓励传输。若比能够接受的更多数据到达，它将被丢弃。这将引起额外的传输，增加对网络和 TCP 的非必要负载。指出小窗口可能限制数据传输以指出在每个新传输分段间引入往返延迟。

提供该机制允许 TCP 通告大窗口并后序通告更小窗口，而无需接受更多数据。这，因此被称为“收缩窗口”，强烈不推荐。精力充沛原则指示 TCP 将不收缩窗口自身，但将在某些其它 TCP 上准备这种行为。

发送方 TCP 必须准备从用户接受并发送至少 1 字节新数据，即使发送窗口是 0。发送方 TCP 必须有规律地重传给接收方 TCP，即使当窗口是 0 时。2 分钟是推荐的重传间隔，当窗口为 0 时。该重传将本质上保证当任何 TCP 有 0 窗口时，重打开窗口将被可靠地报告给另一方。

当接收方 TCP 有 0 窗口且到达它的分段必须依然发送确认显示其下个被希望序列号和当前窗口（0）。

发送方 TCP 打包数据到被传输的分段中，它适合当前窗口，且可能在重传队列中重打包分段。这种重打包是不必要的，但可能有帮助。

在单向数据流的连接中，窗口信息将在确认分段中挟带，它全部有相同的序列号因此将没有办法来重排序它们，若它们无序地到达。这不是个严重的问题，但它将允许窗口信息有临时机会来基于来自数据接收方的旧报告。避免该问题的精致安排将作为来自分段的窗口信息，它挟带更高的确认号（即是该分段的确认号与以所接收的更高的相等或更大）。

窗口管理过程已经明显影响通迅性能。如下注释是对实现者的建议。

窗口管理建议

分配非常小的窗口将引起数据被重传为许多分段，当更佳性能可使用更少大分段来达到。

对避免小窗口的一个建议是接收方区别更新窗口，直到该连接的额外可分配最大可分配的至少 X 百分比（这里 X 可能是 20 到 60）。

另一个建议是发送方避免发送小分段，通过等待直到窗口在发送数据前足够大。若用户通知上推功能，这里数据必须被发送即使它是小分段。

要注意该确认不该被延迟或不必要重传将引起。一个策略将是发送确认当小分段到达时（不需要更新窗口信息），且当窗口更大时以新窗口信息来发送另一个确认。

发送以探测 0 窗口 的分段还可以开始制止重传数据为更小的和更小的分段。若包含单个数据字节被发送以探测 0 窗口的分段被接受，它消耗可用窗口的一个字节。若发送方 TCP 尽可能简单发送尽量多，无论何时该窗口是非 0 时，被传输数据将被分割为交错的大和小分段。如时间继续，偶尔暂停使窗口分配有效的接收方将引起分割大分段为小的且非完全如此大的对。且在一段时间后数据传输将更小的分段。

这里的建议是 TCP 实现需要活动地尝试组合小窗口分配为大窗口，既然管理窗口的机制趋向于引起许多小窗口为最简单有想法的实现。

3.8、接口

这里当然涉及到 2 个接口：用户/TCP 接口和 TCP/低层接口。我们有相当精心制作的用户/TCP 接口模型，但低层协议模块的接口这里还没有规定，因它将由低层协议规范详细地规定。作为低层是 IP 的情况，我们要注意 TCP 可能使用某些参数值。

用户/TCP 接口

下述功能描述 TCP 的用户命令是，最佳的，编造的，因每个操作系统将有不同的工具。因此，我们必须警告读者，不同的 TCP 实现可能有不同的用户接口。然而，所有 TCP 必须提供相当少的服务集来保证全部 TCP 实现可以支持相同的协议层次。本节规定全部 TCP 实现要求的功能性接口。

TCP 用户命令

下节功能性地表现用户/TCP 接口。所使用的术语与在高级语言中的大多数过程或功能调用相似，但该语法不意味着规定陷井类服务调用（如，SVC、UUO、EMT）。

下面描述的指定 TCP 的基本功能的用户命令必须执行支持进程间通迅。独立实现必须定义它们自己的精确格式，且可能在单个调用中提供基本功能的组合或子集。特别是，某些实现可能希望在用户为给定连接首次引入 OPEN 或 RECEIVE 时自动 OPEN。

在提供进程间通迅工具时，TCP 禁止只接受命令，但还必须返回信息来处理其服务。后面的组成有：

（a）关于连接的一般信息（如，中断、远程关闭、非指定远端套接口绑定）。

（b）依靠指定用户命令来表示成功或各种类型的失败。

打开

格式：OPEN（本地端口，远端套接口，主动/被动[，超时][，优先级][，安全性/隔离区][，选项]）-> 本地连接名

我们假设，本地 TCP 清楚它所服务的进程的标识且将检查使用指定连接的进程的权限。取决于 TCP 的实现，本地网络和源地址的 TCP 标识符将被 TCP 或低层协议（如，IP）之一所支持。这些考虑是关心安全性，直到没有 TCP 能够化装为另一个，等等的结果。相似地，没有进程能够离开 TCP 共谋而化装为另一个。

若主动/被动标识设置为被动，这时称为 LISTEN 输入连接。被动打开可能有完全规定远端套接口来等待特定连接或非指定远端套接口来等待任何调用。完全指定被动调用可以通过 SEND 的后序列执行来作为主动的。

传输控制块（TCB）被创建并部分填充来自 OPEN 命令参数的数据。

作为主动 OPEN 命令，TCP 将开始立即同步化（如，建立）连接的进程。

超时时，若存在，允许调用者设置提交给 TCP 的全部数据的超时。若数据在超时周期内递交到目标不成功，TCP 将中止连接。存在全局缺省值是 5 分钟。

TCP 或某些操作系统组件将验证用户权限以用指定优先级或安全性/隔离区打开连接。在 OPEN 调用中缺少指出缺省值的优先级或安全性/隔离区规定必须被使用。

TCP 将因匹配而接受输入请求，仅当安全性/隔离区信息几乎相同时，且仅当该优先级与 OPEN 调用所请求的优先级相同或更高时。

连接的优先级是 OPEN 调用中所请求的和从输入请求接收的更高值，并在连接生存期内

固定到某个值。实现者可能希望给出该优先级协商的用户控制。例如，用户可能被允许规定该优先级必须精确匹配，或者尝试提升优先级任何行为应该由用户确认。

本地连接名将由 TCP 返回给用户。本地连接名这时能够用作通过<本地套接口、远端套接口>对定义连接的短期连接。

发送

格式：SEND（本地连接名，缓存地址，字节计数，PUSH 标志，URGENT 标志[，超时]）

该调用引起包含在指示用户缓存中的数据被发送到指出的连接上。若该连接还未打开，SEND 被认识有错误。某些实现可能允许用户首先 SEND；在该情况下，自动 OPEN 将完成。若调用进程无权限使用该连接，则返回错误。

若设置 PUSH 标志，数据必须迅速地传输给接收方，且 PUSH 位将在从缓存中创建的最后 TCP 分段中设置。若没有设置 PUSH 标导，数据可能因传输效率与后序列 SEND 的数据相组合。

若设置 URGENT 标志，发送给目的 TCP 的分段将有紧急指针设置。接收方 TCP 将向接收方进程发送紧急条件信号，若紧急指针指出该紧急指针前的数据没有被接收方进程消费。紧急的目的是刺激接收方处理紧急数据并指示接收方何时全部当前已知紧急数据已经被收到。发送用户的紧急 TCP 信号的次数没有必要与接收方用户的次数相等，将不会注意到紧急数据的存在。

若 OPEN 中没有指定远端套接口，但连接被建立（如，因 LISTEN 连接已经开始指定，因远端分段到达本地套接口），这时指定缓存被发送到暗指的远端套接口。使用未指定远端套接口的 OPEN 用户可能使用 SEND，而无需曾经明确知道远端套接口地址。

然而，若 SEND 在远端套接口规定前被尝试，将返回错误。用户能够使用 STATUS 调用来判断连接的状态。在某些实现中，TCP 可以提醒用户何时绑定未指定套接口。

若规定超时，该连接的当前用户超时修改为新的。

在最简单的实现中，SEND 将不返回控制到发送进程，直到传输完成或超时已经超过。然而，该简单方法是既造成死锁（例如，连接的双方可能在进行任何 RECEIVE 前尝试进行 SEND）又提供差的性能，因此它不被建议。更复杂的实现将立即返回以允许进程与网络 I/O 并发运行，且，进而，允许以后多次 SEND。多次 SEND 服务首次到达的，首次服务顺序，因此 TCP 将排队这些它不能立即服务的。

我们暗中假设异步用户接口，其中 SEND 后来引出某种类型的 SIGNAL 或来自服务 TCP 的伪中断。另一种方法是立即返回响应。例如，SEND 可能立即返回本地确认，即使发送的分段未经远程 TCP 确认。我们可能乐观地假设最终成功。若我们错了，该连接总之将因超时而关闭。在这种类型（同步）的实现中，将依然存在某些异步信号，但这些将由连接自己处理，且不由指定分段或缓存。

为了使进程区别不同 SEND 的错误或成功指示，返回缓存地址与编码响应给 SEND 请求可能是适当的。TCP 到用户信号在下面讨论，指出应该返回给调用进程的信息。

接收

格式：RECEIVE（本地连接名，缓存地址，字节计数）->字节计数，紧急标志，上推标志

本命令分配与指定连接相关的接收缓存。若该命令之前的 OPEN 或调用进程没有权限使用该连接，将返回错误。

在最简单的实现中，控制将不返回到调用进程，直到缓存被填充，或者某些错误发生，但该方案高度造成死锁。更复杂实现将允许几们 RECEIVE 立即显著。这些将因分段到达而填充。该策略允许增加生产力，代价是更精心制作的方案（可能异步的）以提醒调用程序，已发现 PUSH 或缓存已被填充。

若在发现 PUSH 前有足够的数据到达并填充缓存，则 PUSH 标志将不被设置以响应 RECEIVE。该缓存将被填充它能够拥有的尽量多的数据。若 PUSH 在缓存填充前被发现，则缓存会将部分填充和 PUSH 指示返回。

若有紧急数据，用户将在其到达后通过 TCP 到用户信号尽量快速地提醒。接收方用户应该因此进入“紧急模式”。若设置 URGENT 标志，额外紧急数据将保留。若没有设置 URGENT 标志，则该 RECEIVE 调用将返回全部紧急数据，且用户现在可能离开“紧急模式”。要注意，接着紧急指针（非紧急数据）的数据不能递交给用户的紧急数据前的相同缓存，除非边界对用户清楚标注。

为区别几个显著 RECEIVE 并关心该情况，即缓存未完全填充，返回代码将陪伴缓存指针与指出所接收数据的实际长度的字节计数。

RECEIVE 的替代实现可能使用 TCP 分配缓存存储，或者 TCP 可能共享用户的环形缓存。

关闭

格式：CLOSE（本地连接名）

该命令将引起指定连接被关闭。若该连接未被打开或调用进程没有权限使用该连接，将返回错误。关闭连接用于友好操作，意义是显著 SEND 将被传输（和重传），如流控所允许，直到全部已经服务。因此，它应该可接受作出几个 SEND 调用，接着是一个 CLOSE，且希望全部数据都发送到目的地。它还应该清楚，用户应该在 CLOSE 连接后继续 RECEIVE，因另一方可能正试图传输其最后的数据。因此，CLOSE 意味着“我没有更多要发送了”，但不意味着“我将不接收任何更多的了”。可能发生（若用户层协议经过良好思考）关闭端不能在超时前摆脱其全部数据。在该情况下，CLOSE 变为 ABORT，且正关闭 TCP 将放弃。

用户可以在任何时间由其自己主动，或者响应来自 TCP 的多种提示（如，远端关闭被执

行，传输超时，目的端不可访问）而 CLOSE 连接。

因关闭连接要求与远端 TCP 进行通迅，连接可能短时间维持关闭状态。在 TCP 回复 CLOSE 命令前尝试重打开连接将造成错误响应。

关闭还暗指上推功能。

状态

格式：STATUS（本地连接名）->状态数据

这是个依赖用户使用的实现，且可能没有通告效果。返回的信息可能一般来自连接相关的 TCB。

该命令返回的数据块包含如下信息：

本地套接口，
远端套接口，
本地连接名，
接收窗口，
发送窗口，
连接状态，
等待确认的缓存数，
接收挂起的缓存数，
紧急状态，
优先级，
安全性/隔离区，
和传输超时。

取决于连接状态，或者实现自己，某些信息可能不可用或没有意义。若调用进程没有权限使用该连接，将返回错误。这将阻止非认证进程收集关于该连接的信息。

中止

格式：ABORT（本地连接名）

本命令引起所有挂直的 SEND 和 RECEIVE 被中止，TCB 将被删除，且特别 RESET 消息将发送到 TCP 连接的另一端。取决于实现，用户可以为每个显著 SEND 或 RECEIVE 接收中止指示，或者可能简单地接收 ABORT 确认。

TCP 到用户消息

假设操作系统环境为 TCP 提供手段来异步发信号给用户程序。当 TCP 发信号给用户程序时，某些信息被传递给用户。在规范信息中经常是错误消息。在其它情况下，将会有与处理

SEND 或 RECEIVE 或其它用户调用的完成相关的信息。

提供如下信息：

本地连接名	经常
响应串	经常
缓存地址	发送和接收
字节计数（接收字节计数）	接收
上推标志	接收
紧急标志	接收

TCP/低层接口

TCP 调用低层协议模块来实际通过网络发送和接收信息。一种情况是，ARPA 互联网络系统中低层模块是网际协议（IP）[2]。

若低层协议是 IP，它提供服务类型和存活时间参数。TCP 使用这些参数的如下设置：

服务类型=优先级：路由，延迟：正常，吞吐量：正常，可靠性：正常；或者 00000000。

存活时间=1 分钟，或者 00111100。

要注意，假设最大分段存活时间是 2 分钟。这里我们明确地告诉，若不能在互联网系统中在一分钟内被递交，分段将被存消灭。

若低层是 IP（或者其它提供该特性的协议）且使用源路由，接收必须允许通迅路由信息。这是特别重要的，因此 TCP 中使用的源和目的地址校验和将是源始源和最终目的地。阻止返回路由来回答连接请求也是重要的。

任何低层协议将必须提供源地址，目的地址，和协议域，和某些方法来判断“TCP 长度”，既提供与 IP 功能性等价的服务，又用于 TCP 检验和中。

3.9、事件处理

本节中描述的处理是一种可能实现的一个例子。其它实现可能有轻微不同的处理顺序，但它们应该与本节中的这些仅在细节上不同，而非实质上的。

TCP 的活动性可能表现为对事件的响应。发生的事件可能投入三类：用户调用，到达分段和超时。本节描述 TCP 在响应每个事件时的处理。在许多情况下，所要求的处理取决于连接的状态。

发生的事件：

用户调用

OPEN
SEND
RECEIVE
CLOSE
ABORT
STATUS

到达分段

SEGMENT ARRIVES

超时

USER TIMEOUT
RETRANSMISSION TIMEOUT
TIME-WAIT TIMEOUT

TCP/用户接口模型是用户命令通过事件或伪中断接收到立即返回和可能有延迟的响应。在下面描述中，术语“信号”意思是引起延迟的响应。

错误响应以字符串给出。例如，引用不存在连接的用户命令收到“错误：连接未打开”。

请注意，在下面关于序列号、确认号、窗口，等等的全部算法，是模序列号空间的大小 2**32。还要注意，“<=”意思是小于或等于（模 2**32）。

思考关于处理输入分段的自然方式是想象它们首次测试正确序列号（如，它们的内容位于序列号空间中的希望“接收窗口”的范围内）且这时它们通常排队并以序列号顺序处理。

当分段覆盖其它已接收分段时，我们重组只包含新数据的分段，并调整需一致的头部域。

要注意，若没有提供状态改变，TCP 维持相同的状态。

OPEN 调用

CLOSED STATE（如，TCB 不存在）

创建新的传输控制块（TCB）以保持连接状态信息。填充在本地套接口标识符、远端套接口、优先级、安全性/隔离区、和用户超时信息。要注意，远端套接口的某些部分可能在被动 OPEN 中未指定且以输入 SYN 分段的参数来填充。验证所请求的安全性和优先级对该用户是被允许的，若不返回“错误：优先级不允许”或“错误：安全性/隔离区不允许”。若被动进入 LISTEN 状态并返回。若主动且远端套接口未指定，返回“错误：远端套接口未指定”；若主动和远端套接口指定，则发出 SYN 分段。初始发送序列号（ISS）被选择。形式<SEQ=ISS><CTL=SYN>的 SYN 分段被发送。设置 SND.UNA 为 ISS，SND.NXT 为 ISS+1，进入

SYN-SENT 状态，并返回。

若呼叫者不访问本地规定套接口，返回“错误：该进程连接非法”。若没有空间创建新连接，返回“错误：无充足资源”。

LISTEN STATE

若指定活动和远端套接口，这时将连接从被动变为主动，选择 ISS。发送 SYN 分段，设置 SND.UNA 为 ISS，SND.NXT 为 ISS+1。进入 SYN-SENT 状态。与 SEND 相关的数据将在进入 ESTABLISHED 状态后以 SYN 分段发送或为传输排队。若在命令中所请求的紧急位必须以数据分段发送以作为该命令的结果。若没有空间排队该请求，响应以“错误：无足够资源”。若远端套接口未指定，这时返回“错误：远端套接口未指定”。

SYN-SENT STATE
SYN-RECEIVED STATE
ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

返回“错误：连接已存在”

SEND 调用

CLOSED STATE（如，TCB 不存在）

若用户无权限访问该连接，这时返回“错误：连接对该进程非法”。

否则，返回“错误：连接不存在”。

LISTEN STATE

若远端套接口被指定，这时改变连接从被动到主动，选择一个 ISS。发送 SYN 分段，设置 SND.UNA 为 ISS，SND.NXT 为 ISS+1。进入 SYN-SENT 状态。与 SEND 相关的数据可能在进入 ESTABLISHED 状态后以 SYN 分段发送或者为传输排队。若在命令中请求的紧急位必须以数据分段发送，作为该命令的结果发送。若没有空间排队该请求，响应以“错误：无足够资源”。若远端套接口未指定，这时返回“错误：远端套接口未指定”。

SYN-SENT STATE
SYN-RECEIVED STATE

在进入 ESTABLISHED 状态后排队传输数据。若无空间排队，响应以“错误：无足够资源”。

ESTABLISHED STATE
CLOSE-WAIT STATE

分段缓存并以捎带确认发送它（确认值=RCV.NXT）。若没有足够空间记住该缓存，简单地返回“错误：无足够资源”。

若设置紧急标志，这时 SND.UP<-SND.NXT-1 并在外出分段中设置紧急指针。

FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

返回“错误：连接关闭”且不服务请求。

RECEIVE 调用

CLOSED STATE（如，TCB 不存在）

若用户无权利访问该连接，返回“错误：连接对该进程非法”。

否则返回“错误：连接不存在”。

LISTEN STATE
SYN-SENT STATE
SYN-RECEIVED STATE

在进入 ESTABLISHED 状态后的处理排队。若无空间排队该请求，响应以“错误：无足够资源”。

ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE

若无足够输入分段被排队以满足请求，则排队该请求。若无排队空间来记住 RECEIVE，响应以“错误：无足够资源”。

重组已排队向内人分段到接收缓存中并返回给用户。标记“发现上推”（PUSH）若是该情况的话。

若 RCV.UP 先于当前正传递给用户的数据，提示用户存在紧急数据。

当 TCP 有责任递交数据给用户时，事实必须是通过确认与发送方通迅。这类确认的格式在下面讨论处理到达分段中描述。

CLOSE-WAIT STATE

因远端已经发送 FIN，RECEIVE 必须以已经在手的文本来满足，但还未递交给用户。若没有文本等待递交，RECEIVE 将获得“错误：连接关闭”响应。否则，任何剩余文本可能被用于满足 RECEIVE。

CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

返回“错误：连接关闭”。

CLOSE 调用

CLOSED STATE（如，TCB 不存在）

若用户无权限访问该连接，返回“错误：连接对该进程非法”。

否则，返回“错误：连接不存在”。

LISTEN STATE

任何显著 RECEIVE 都返回以“错误：关闭响应”。删除 TCB，进入 CLOSED 状态，并返回。

SYN-SENT STATE

删除 TCB 并返回“错误：关闭”对任何排队 SEND，或 RECEIVE 的响应。

SYN-RECEIVED STATE

若没有引入 SEND 且没有挂起要发送的数据，这时形成 FIN 分段并发送它，然后进入 FIN-WAIT-1 状态；否则在进入 ESTABLISHED 状态后排队处理。

ESTABLISHED STATE

排队它直到全部之前的 SEND 已经被分段，这时形成 FIN 分段并发送它。在任何情况下，进入 FIN-WAIT-1 状态。

FIN-WAIT-1 STATE
FIN-WAIT-2 STATE

确切地说，这是个错误且应该接收“错误：连接关闭”响应。“ok”响应该可接受，也，在没有发出第二个 FIN 时（首次 FIN 虽然可能被重传）。

CLOSE-WAIT STATE

排队该请求直到全部之前的 SEND 已经被分段，这时发送 FIN 分段，进入 CLOSING 状态。

CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

响应以“错误：连接关闭”。

ABORT 调用

CLOSED STATE（如，TCB 不存在）

若用户不该有限制访问该连接，返回“错误：连接对该进程非法”。

否则返回“错误：连接不存在”。

LISTEN STATE

任何显著 RECEIVE 应该返回以“错误：连接重置”响应。删除 TCB，进入 CLOSED 状态，并返回。

SYN-SENT STATE

全部排队 SEND 和 RECEIVE 应该给出“连接重置”提示，删除 TCB，进入 CLOSED 状态，并返回。

SYN-RECEIVED STATE
ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSE-WAIT STATE

发送重置分段：

<SEQ=SND.NXT><CTL=RST>

全部排队 SEND 和 RECEIVE 应该给出“连接重置”提示；全部分段排队传输（除非上面形成的 RST）或者重传应该被更新，删除 TCB，进入 CLOSED 状态，并返回。

CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

响应以“ok”并删除 TCB，进入 CLOSED 状态，并返回。

STATUS 调用

CLOSED STATE（若，TCB 不存在）

若用户不该有权限访问该连接，返回“错误：连接对该进程非法”。

否则返回“错误：连接不存在”。

LISTEN STATE

返回“状态=LISTEN”，和 TCB 指针。

SYN-SENT STATE

返回“状态=SYN-SENT”，和 TCB 指针。

SYN-RECEIVED STATE

返回“状态=SYN-RECEIVED”，和 TCB 指针。

ESTABLISHED STATE

返回“状态=ESTABLISHED”，和 TCB 指针。

FIN-WAIT-1 STATE

返回“状态=FIN-WAIT-1”，和 TCB 指针。

FIN-WAIT-2 STATE

返回“状态=FIN-WAIT-2”，和 TCB 指针。

CLOSE-WAIT STATE

返回“状态=CLOSE-WAIT”，和 TCB 指针。

CLOSING STATE

返回“状态=CLOSING”，和 TCB 指针。

LAST-ACK STATE

返回“状态=LAST-ACK”，和 TCB 指针。

TIME-WAIT STATE

返回“状态=TIME-WAIT”，和 TCB 指针。

SEGMENT ARRIVES

若状态是 CLOSED（如，TCB 不存在）这时到达分段中的全部数据被丢弃。包含 RST 的到达分段被丢弃。不包含 RST 的到达分段引起 RST 被发送以响应。确认和序列域值被选择使重置序列对发送讨厌分段的 TCP 可接受。

若 ACK 位关闭，则使用序列号 0。

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

若 ACK 位打开，

<SEQ=SEG.ACK><CTL=RST>

返回。

若状态是 LISTEN 这时

首先检查 RST

达到 RST 应该被乎略。返回。

第二步检查 ACK

任何到达依然是 LISTEN 状态连接的确认是坏的。可接受重置分段应该形成任何到达 ACK 方面分段。RST 应该格式化如下：

<SEQ=SEG.ACK><CTL=RST>

返回。

第三步检查 SYN

若 SYN 位设置，检查安全性。若到达分段上的安全性/隔离区与 TCB 中的安全性/隔离区不精确匹配，这时发送重置并返回。

<SEQ=SEG.ACK><CTL=RST>

若 SEG.PRC 大于 TCB.PRC，这时若用户允许且系统设置 TCB.PRC<-SEG.PRC，若不允许发送重置并返回。

<SEQ=SEG.ACK><CTL=RST>

若 SEG.PRC 小于 TCB.PRC，这时继续。

设置 RCV.NXT 为 SEQ.SEQ+1，IRS 设置为 SEG.SEQ 且任何其它控制或文本应该排队之后的处理。ISS 应该被选择且 SYN 分段发送形式是：

<SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>

SND.NXT 设置为 ISS+1 且 SND.UNA 为 ISS。连接状态应该改变为 SYN-RECEIVED。要注意，任何其它到达控制或数据（与 SYN 组合）将在 SYN-RECEIVED 状态中处理，但 SYN 和 ACK 的处理应该是非重复的。若监听未完全指定（如，远端套接口非完全指定），这时非指定域应该以现值来填充。

第四步其它文本或控制

任何其它控制或文本方面的分段（不包含 SYN）必须有 ACK 且因此应该被 ACK 处理所丢弃。到达 RST 分段可能是无效的，因它可能没有被发送以响应连接的该化身发送的任何东西。因此你不可能到达这里，但若你这样做，放弃该分段，并返回。

若状态是 SYN-SENT 这时

首先检查 ACK 位

若 ACK 位设置

若 SEG.ACK =< ISS，或者 SEG.ACK > SND.NXT，发送重置（除非 RST 位设置，若因此放弃该分段并返回）。

<SEQ=SEG.ACK><CTL=RST>

并丢弃该分段。返回。

若 SND.UNA =< SEG.ACK =< SND.NXT 这时 ACK 是可接受的。

第二步检查 RST 位

若 RST 位设置

若 ACK 是可接受的，这时发信号给用户“错误：连接重置”，丢弃该分段，进入 CLOSED 状态，删除 TCB，并返回。否则（无 ACK）丢弃该分段并返回。

第三步检查安全性和优先级

若在分段中的安全性/隔离区与 TCB 中的安全性/隔离区不精确匹配，发送重置。

若是 ACK

<SEQ=SEG.ACK><CTL=RST>

否则

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

若是 ACK

分段中的优先级必须匹配 TCB 中的优先级，若没有，发送重置。

<SEQ=SEG.ACK><CTL=RST>

若没有 ACK

若分段中的优先级高于 TCB 中的优先级这时若用户允许且系统提升 TCB 中的优先级为该分段中的，若不允许提升优先级这时发送重置。

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

若分段中的优先级低于 TCB 中的优先级，则继续。

若重置被发送，丢弃该分段并返回。

第四步检查 SYN 位

该步应该仅当 ACK 是好的时执行，或者不存在 ACK，且分段不包含 RST。

若 SYN 位打开且安全性/隔离区和优先级是可接受的这时，RCV.NXT 被设置为 SEQ.SEQ+1，IRS 设置为 SEG.SEQ。SND.UNA 应该是优先与 SEG.ACK 相等（若存在 ACK），且重传队列上待待确认的任何分段应该被删除。

若 SND.UNA > ISS（我们 SYN 已经 ACK 了），修改连接状态为 ESTABLISHED，形成 ACK

分段。

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

并发送它。排队等待传输的数据或控制可能被包括。若分段中有其它的控制或文本这时继续在下面第 6 步处理，这里 URG 位被检查，否则返回。

否则进入 SYN-RECEIVED，形成 SYN，ACK 分段

<SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>

并发送它。若分段中有其它控制或文本，在到达 ESTABLISHED 状态后排队它们以处理，返回。

第 6 步，若 SYN 和 RST 位都没有设置这时放弃该分段并返回。

否则，

首先检查序列号

SYN-RECEIVED STATE
ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

分段按顺序处理。初始到达测试用于丢弃旧复本，但进一步处理以 SEG.SEQ 顺序完成。若分段的内容跨在旧的和新的边界间，只有新部分应该被处理。

对到达分段的可接受性测试有 4 种情况：

分段长度	接收长度	窗口
0	0	SEG.SEQ = RCV.NXT
0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND
>0	0	可接受
>0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND

或者 RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

若 RCV.WND 是 0，没有分段是可接受的，但特别许可应该作出以接受有有效 ACK、URG 和 RST。

若到达分段不可接受，确认应该被发送以响应（除非 RST 位设置，若这样丢弃该分段并返回）：

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

在发送确认后，丢弃不可接受分段并返回。

在下面，假设分段是理想化的分段，它开始于 RCV.NXT 且不超出窗口。可以裁剪实际分段以适合该假设，通过去除超出窗口的任何部分（包括 SYN 和 FIN），且只进一步处理，若分段这里开始于 RCV.NXT。用更高初始序列号的分段可能为后序处理拥有。

第二步检查 RST 位，

SYN-RECEIVED STATE

若 RST 位设置

若该连接以被动 OPEN（如，从 LISTEN 状态开始）发起，这时返回该连接到 LISTEN 状态并返回。用户不需要被提醒。若该连接以主动 OPEN（如，从 SYN-SENT 状态），这时连接被拒绝，提醒用户“连接被拒绝”。在任何情况下，重传队列上所有分段应该被删除。且在主动 OPEN 情况下，进入 CLOSED 状态并删除 TCB，然而返回。

ESTABLISHED
FIN-WAIT-1
FIN-WAIT-2
CLOSE-WAIT

若 RST 位设置这时，任何显著 RECEIVE 和 SEND 应该接收“重置”响应。全部分段队列应该被刷新。用户还应该接收主动的一般“连接重置”信号。进入 CLOSED 状态，删除 TCB，并返回。

CLOSING STATE
LAST-ACK STATE
TIME-WAIT

若 RST 位设置，这时进入 CLOSED 状态，删除 TCB，并返回。

第 3 步检查安全性和优先级

SYN-RECEIVED

若分段中的安全性/隔离区和优先级与 TCB 中的安全性/隔离区和优先级精确匹配，这时发送重置，并返回。

ESTABLISHED STATE

若分段中的安全性/隔离区和优先级与 TCB 中的安全性/隔离区和优先级不精确匹配，这时发送重置，任何显著 RECEIVE 和 SEND 应该接收“重置”响应。全部分段队列应该被刷新。用户还应该接收主动一般“连接重置”信号。进入 CLOSED 状态，删除 TCB，并返回。

要注意，该检查放在序列号检查之后以阻止分段来自有不同安全性的这些端口间的旧连接或来自引起当前连接中止的优先级。

第 4 步，检查 SYN 位，

SYN-RECEIVED
ESTABLISHED STATE
FIN-WAIT STATE-1
FIN-WAIT STATE-2
CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

若 SYN 在窗口中，它是错误的，发送重置，任何显著 RECEIVE 和 SEND 应该接收“重置”响应，所有分段队列应该被刷新，用户还应该接收主动一般“连接重置”信号，进入 CLOSED 状态，删除 TCB，并返回。

若 SYN 不在窗口中，这步将不会到达且确认应该已经在首步中被发送（序列号检查）。

第 4 步检查 ACK 域，

若 ACK 域关闭丢弃该分段并返回

若 ACK 位打开

SYN-RECEIVED STATE

若 SND.UNA =< SEG.ACK =< SND.NXT 这时进入 ESTABLISHED 状态并继续处理。

若分段确认不可接受，形成重置分段，

<SEQ=SEG.ACK><CTL=RST>

并发送它。

ESTABLISHED STATE

若 SND.UNA < SEG.ACK =< SND.NXT 这时，设置 SND.UNA <- SEG.ACK。重传队列上的任何分段因此完全确认的应该被删除。用户应该接收缓存的肯定确认，它已经被 SENT 并完全确认（如，SEND 缓存应该返回“ok”响应）。若 ACK 是重复的（SEG.ACK < SND.UNA），它可能被乎略。若 ACK 要求某些还未发送的（SEG.ACK > SND.NXT）这时发送 ACK，丢弃该分段，并返回。

若 SND.UNA < SEG.ACK =< SND.NXT，发送窗口应该被更新。若（SND.WL1 < SEG.SEQ 或者(SND.WL1 = SEG.SEQ 且 SND.WL2 =< SEG.ACK)），设置 SND.WND <- SEG.WND，设置 SND.WL1 <- SEG.SEQ，并设置 SND.WL2 <- SEG.ACK。

要注意，SND.WND 是从 SND.UNA 的偏移，即 SND.WL1 记录最后用于更新 SND.WND 的分段的序列号，且该 SND.WL2 记录最后用于更新 SND.WND 的确认号。这里的检查阻止使用旧分段更新窗口。

FIN-WAIT-1 STATE

作为对处理 ESTABLISHED 状态的补充，若我们的 FIN 现在被确认，这时进入 FIN-WAIT-2 并继续在该状态中处理。

FIN-WAIT-2 STATE

作为对 ESTABLISHED 状态的补充，若重传队列是空的，用户的 CLOSE 可能被确认(“ok”)但不删除 TCB。

CLOSE-WAIT STATE

执行对 ESTABLISHED 状态相同的处理。

CLOSING STATE

作为对 ESTABLISHED 状态处理的补充，若 ACK 确认我们的 FIN，这时进入 TIME-WAIT 状态，否则乎略该分段。

LAST-ACK STATE

可能达到该状态的唯一事件是我们的 FIN 的确认。若我们的 FIN 现在被确认，删除 TCB，进入 CLOSED 状态，并返回。

TIME-WAIT STATE

可能达到该状态的唯一事件是远端 FIN 的重传。确认它，且重启 2MSL 超时。

第 6 步，检查 URG 位，

ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE

若 URG 位设置，RCV.UP <- max(RCV.UP,SEG.UP)，且发信号给用户，即远端有紧急数据，若紧急指针（RCV.UP）在数据之前消费。若用户已经因持续紧急数据而被递送信号（或者依然处于“紧急模式”），不要再次给用户发送信号。

CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT

这不该发生，因 FIN 已经从远端收到。乎略 URG。

第 7 步，处理分段文本，

ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE

一旦进入 ESTABLISHED 状态，递交分段文本给用户的 RECEIVE 缓存是可能的。分段中的文本可能被移动到缓存中，直到任何缓存满或分段是空的止。若分段清空且挟带 PUSH 标志，这时用户被提醒，当缓存返回时，PUSH 已经被接收。

当 TCP 有责任递交数据给用户时，它还必须确认接收的数据。

一旦 TCP 有责任在所接受数据之上增加数据的 RCV.NXT，且适当调整 RCV.WND 为当前可用缓存。RCV.NXT 与 RCV.WND 的总和不该减小。

请注意 3.7 小节中的窗口管理建议。

发送确认的形式：

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

该确认应该是捎带在正要发送的分段中，若可能且不招致不必要的延迟。

CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

这不该发生，因 FIN 已经从远端收到。乎略分段文本。

第 8 步，检查 FIN 位，

不处理 FIN，若状态是 CLOSED，LISTEN 或 SYN-SENT，因 SEG.SEQ 不可能被认证；丢弃分段并返回。

若 FIN 位设置，发“连接关闭”信号给用户并以相同的消息返回任何挂起的 RECEIVE，在 FIN 之上增加 RCV.NXT，且发送 FIN 的确认。要注意，FIN 潜指 PUSH 任何未递交分段文本给用户。

SYN-RECEIVED STATE
ESTABLISHED STATE

进入 CLOSE-WAIT 状态。

FIN-WAIT-1 STATE

若我们的 FIN 已经被 ACK（可能在该分段中），这时进入 TIME-WAIT，开始时间等待计时器，关闭其它计时器；否则进入 CLOSING 状态。

FIN-WAIT-2 STATE

进入 TIME-WAIT 状态。开始时间等待计时器，关闭其它时间器。

CLOSE-WAIT STATE

维持 CLOSE-WAIT 状态。

CLOSING STATE

维持 CLOSING 状态。

LAST-ACK STATE

维持 LAST-ACK 状态。

TIME-WAIT STATE

维持 TIME-WAIT 状态。重启 2MSL 时间等待超时。

并返回。

USER TIMEOUT

对于任何状态，若用户超时截止，刷新全部队列，一般发信号“错误：连接因用户超时而中止”给用户且对于任何显著调用，删除 TCB，进入 CLOSED 状态并返回。

RETRANSMISSION TIMEOUT

对于任何状态，若重传队列中的某个分段上的重传超时截止，再次在重传队列最前面发送该分段，重启重传计时器，并返回。

TIME-WAIT TIMEOUT

若连接上的时间等待超时截止，删除 TCB，进入 CLOSED 状态并返回。

术语表

1822
BBN 报告 1822，“主机和 IMP 互连规范”。主机和 ARPANET 间的接口规范。

ACK
控制位（确认）不占用任何序列空间，它指出分段的确认域规定该分段的发送方正期望接收的下个序列号，因此确认全部以前所接收的序列号。

ARPANET 消息
ARPANET 中主机与 IMP 间的传输单元。最大长度大约 1012 个字节（8096 位）。

ARPANET 分组
在 ARPANET 的 IMP 间内部使用的传输单元。最大长度大约 126 个字节（1008 位）。

连接
由套接口对标识的逻辑通迅路径。

数据报
在交换计算机通迅网络的分组中发送的消息。

目的地址
目的地址，一般的网络和主机标识符。

FIN

占用一个序列号的控制位（结尾），它指出发送方将不再发送更多占用序列空间的数据或者控制。

分片

数据逻辑单元的一部分，特殊的互联网分片是互联网数据报的一部分。

FTP

文件传输协议。

头部

消息，分段，分片，数据分组或块的开头的控制信息。

主机

计算机。以通迅网络观点的消息的特殊源或目的地。

识别号

一个网际协议域。该标识值由发送方为辅助数据报分片重组而分配。

IMP

接口消息处理器，PARANET 的分组交换。

互联网地址

主机层特别的源或目的地址。

互联网数据报

在互联网模块和与互联网头部一起的更高层协议间交换的数据单元。

互联网分片

有互联网头部的互联网数据报的数据的一部分。

IP

网际协议。

IRS

初始接收序列号。发送方在连接上所使用的首个序列号。

ISN

初始序列号。连接上所使用的首个序列号，（是 ISS 或 IRS 之一）。以基于时钟的过程来选择。

ISS

初始发送序列号。发送方在连接上所使用的首个序列号。

前导

数据的消息或块的开始处的控制信息。特别是，在 ARPANET 中，在 ARPANET 消息的主机 IMP 接口上的控制信息。

左序列

由数据接收 TCP 确认的下个序列号（或者最低当前未确认序列号）且某些时候作为发送窗口的左边界。

本地分组

在本地网络内的传输单。

模块

实现，通常是软件，协议或其它过程的。

MSL

最大分段存活期，TCP 分段能够在互联网系统中存在的时间。直接定义为 2 分钟。

字节

8 位字节。

选项

选项域可能包含几个选项，且每个选项可能是几个字节长。选项主要用于测试环境；例如，挟带时间戳。网际协议和 TCP 都提供选项域。

分组

有头部的数据分组，它可能或不可能是逻辑完全的。更通常是数据的物理分组而非逻辑分组。

端口

套接口的一部分，它规定进程与数据相关的逻辑输入或输出通道。

进程

执行的程序。以 TCP 或其它主机到主机协议的视步的数据的源或目的地。

PUSH

不占用任何序列空间的控制位，指出该分段包含必须直接上推到接收用户的数据。

RCV.NXT

接收下个序列号

RCV.UP

接收紧急指针

RCV.WND

接收窗口

接收下个序列号

逻辑 TCP 希望接收的下个序列号。

接收窗口

这代表逻辑（接收）TCP 愿意接收的序列号组。因此，逻辑 TCP 认为覆盖范围 RCV.NXT 到 RCV.NXT + RCV.WND - 1 间的分段挟带可接受数据或控制。包含完全在该范围之外的序列号的分段被认为是重复的并被丢弃。

RST

控制位（重置），不占用序列空间，指出接收方应该删除连接而无需进一步互动。接收方可以判断，基于到达分段的序列号和确认域，它是否应该遵守重置命令或乎略它。包含 RST 的分段的接收方决不该在其响应是提升 RST。

RTP

实时协议：时间悠关信息通迅的主机到主机协议。

SEG.ACK

分段确认

SEG.LEN

分段长度

SEG.PRC

分段优先级值

SEG.SEQ

分段序列

SEG.UP

分段紧急指针域

SEG.WND

分段窗口域

分段

数据逻辑单元，在特殊 TCP 分段中是在 TCP 模块对间传输的数据单元。

分段确认

到达分段的确认域中的序列号。

分段长度

分段占用的序列号空间总和，包括占用序列空间的任何控制。

分段序列
达到分段的序列域的数字。

发送序列
这是本地（发送）TCP 将在连接上使用的下个序列号。它开始从初始序列号（ISN）选择，且在每个数据字节或序列控制传输后增加。

发送窗口
这表示远端（接收）TCP 愿意接收的序列号。它是来自远端（数据接收）TCP 的分段中规定的窗口域的值。新的序列号范围可能被 TCP 发出，位于 SND.NXT 和 SND.UNA + SND.WND - 1 之间。（重传 SND.UNA 和 SND.NXT 间的序列号是期望的，当然。）

SND.NXT
发送序列

SND.UNA
左序列

SND.UP
发送紧急指针

SND.WL1
最近窗口更新时的分段序列号

SND.WL2
最近窗口更新时的分段确认号

SND.WND
发送窗口

套接口
特别包括端口标识符的地址，即，互联网地址与 TCP 端口的串联。

源地址
源地址，一般的网络和主机标识符。

SYN
向内分段中的控制位，占用一个序列号，在连接发起时使用，指出序列号将从哪里开始。

TCB
传输控制块，记录连接状态的数据结构。

TCB.PRC
连接的优先级。

TCP

传输控制协议：互联网环境中可靠通迅的主机到主机协议。

TOS

服务类型，一个网际协议域。

服务类型

一个网际协议域，指出该互联网分片的服务类型。

URG

一个控制位（紧急），不占用序列空间，用于指出接收用户应该注意进行紧急处理，只要有数据消费比紧急指针中指出的值小的序列号。

紧急指针

一个控制域，仅当 URG 位设置时有意义。该域通迅紧急指针值，它指出与发送用户的紧急调用相关的数据字节。

参考资料

[1] Cerf，V.，和 R. Kahn，“A Protocol for Packet Network Intercommunication”，IEEE 通迅事务，卷号 COM-22，编号 5，pp 637-648，1974 年 5 月。

[2] Postel，J.（ed.），“Internet Protocol - DARPA Internet Program Protocol Specification”，RFC 791，USC/信息科学学院，1981 年 9 月。

[3] Dalal，Y.和 C. Sunshine，“Connection Management in Transport Protocols”，计算机网络，卷号 2，编号 6，pp. 454-473，1978 年 12 月。

[4] Postel，J.，“Assigned Numbers”，RFC 790，USC/信息科学学院，1981 年 9 月。