第五章 运输层

运输层引入了一个端口，而端口则是进程可以使用的输入/输出。在一大堆的IP协议背后，我们很容易会忽视IP层的共同特点，就是IP层在负责内容按照什么路径传送到对方手里，而没有很关注如何保证内容传送到位和送到对方主机的哪个位置，而TCP就负责这个一个事情。

在本章中你将会了解的是：

- UDP协议的特点。
- TCP协议的特点。
- TCP如何保证可靠传输。
- TCP的流量控制、拥塞控制、连接管理。

5.1 运输层协议概述

5.1.1 进程之间的通信

运输层向上面的应用层提供通信服务。计算机进行通信通常使用“两台主机进行通信”这样的表述，但实际上，真正在通信的是某计算机的一个进程和另一计算机的一个进程在交换数据，这就是计算机的端到端通信。

运输层有个很重要的功能：让不同的进程**共用**一个运输层协议，此之谓复用，而分用指的是运输层在东西到对面了以后解包交给正确的应用进程。看上去这种通信实验是水平方向直接传送数据，但事实上两个运输层之间并没有一条水平方向的物理连接。数据的传送是一层层解包和加包的逻辑来的。

IP提供了无连接的服务，因此运输层至少需要提供一种可靠的信道实现，当然也可以由额外的不可靠信道实现，前者的经典是TCP，后者是UDP。而这一层需要向高层用户屏蔽下面网络核心的细节。

5.1.2 运输层的两个主要协议

在OSI术语中，两个对等运输实体在通信时传送的数据单位叫做运输协议数据单元（TPDU）。运输层的两个主要协议就是TCP和UDP，分别对应了TCP报文段和UDP用户数据报。相对而言，TCP面向连接，可靠一些，有额外损耗，UDP不需要先建立连接，不提供可靠交付，额外损耗小的一批。

在应用层协议上，SMTP（电子邮件），TELNET（远程终端接入），HTTP，FTP用的是TCP，其他大部分用的是UDP。

5.1.3 运输层的端口

在单个计算机中，进程是使用一个进程标识符来表示的，但是在互联网环境下这个不可行，因为计算机的操作系统不通，对应的进程标识符也不一样，这会导致对方无法了解自己可以连接到哪里去。于是TCP/IP体系必须自定义一个东西用来标志进程，其结果就是协议端口号。这种端口号是软件接口，即应用层各种协议与运输实体进行层间交互的一种地址，长度16位。

对于端口号，为了方便，分成三类：

- 熟知端口号（系统端口号）数值为0-1023，在www.iana.org可以查到，里面都是大名鼎鼎的协议端口号，也都是TCP/IP的一些重要程序。比如FTP为21，TELNET为23，SMTP为25，SSH为22，DNS为53，TFTP为69，HTTP为80，SNTP为161，HTTPS为443。
- 登记端口号为1024-49151，一般为没有熟知端口号的应用程序使用的，这类端口号也在IANA的手续有等级，以防止重复。
- 短暂端口号为49152-65535，通信结束后这类端口号就不复存在。

5.2用户数据报协议UDP

5.2.1 UDP概述

UDP是非常重要的协议，在IP上只是增加了复用、分用、差错检测的功能。其主要特点是：

- UDP**无连接**，减少开销和发送数据的时延。
- UDP尽最大努力交付，即**不可靠交付**，没有复杂的链接状态表，但是也会有丢失。
- UDP是**面向报文**的。其含义是一次发送一个报文，不管多长多短都是一样的，对于效率问题则是应用层自己处理。
- UDP**没有拥塞控制**。可以网络拥塞时不会降低发送速率。
- UDP支持一对一、一对多、多对一、多对多的交互通信。
- UDP的首部开销非常小，只有8个字节。

通常来说，可靠性是以额外内容作为代价的

5.2 UDP的首部格式

UDP的首部仅有八个字节，四个字段，每个两个字节。

0-15	16-31	32-47	48-64
源端口	目的端口	长度	校验和

如果拿到UDP报文之后，发现目的端口号不存在，就丢弃该报文，并由ICMP发送端口不可达差错报文回去。

在首部校验和的计算中，需要考虑加入一个临时的**伪首部**（但是**不加入传输**）。

0-31	32-63	64-71	72-79	80-95
源IP	目的IP	只写0	写入0x11	UDP报文长度

这里的UDP报文长度不包括伪首部，但是包括UDP首部。按照4字节为单位、二进制反码和作为计算方法计算检验和写入校验和，检验时把校验和纳入考虑计算为0即可。**如果UDP数据部分不满足四个字节的倍数，就尾巴填0。**

5.3 传输控制协议TCP概述

5.3.1 TCP最主要的特点

TCP本身是一个非常复杂的协议。

- TCP是**面向连接**的的传输层协议。在使用TCP协议之前，必须先建立TCP连接。在传输完毕后，必须先释放已经建立的TCP连接。
- TCP是**点对点**的。这就是不把这个东西放IP的最大要点。
- TCP提供**可靠交付的服务**。通过TCP传送的数据，无差错、不丢失、不重复、按序到达。
- TCP提供全双工通信。TCP允许任何时候都可以发送数据，两端都有单独的发送和接收缓存，应用只需要交给TCP缓存之后就可以做自己的事情。
- 面向**字节流**。

TCP和UDP发送报文时的应对方式完全不同。TCP不关心应用进程一次把多长的报文发送到TCP缓存中，而是根据已有的网络情况自行决定一次发多长的报文。

5.4 可靠传输的工作原理

5.4.1 停止等待协议

停止等待协议的机制：

- 按照顺序发送内容。每一次发送的前置条件是收到对方关于上一段的确认，如果没有收到则等待更多时间。
- 如果到了一个**稍长于RTT**的时间还没有收到，默认对方没拿到信息，进行超时重传。
- 如果确认报文迟到了，那么双方啥都不做。

这种方法便是自动重传请求（ARQ），重传是自动进行的，接收方不需要请求发送方重传某个出错的分组。在这样的写一下，信道的利用率非常低，因为大部分数据报都是比较短的，此时RTT会占用大部分的时间。于是产生了流水线传输

5.4.2 连续ARQ协议

TCP中的滑动窗口协议比较复杂，首先来一个简单的。假设有一个不断往前的起始点，固定的区间长度用于划分出一个区间，这个区间内的所有分组都可以发送出去，而不需要等待对方的确认，这样可以提升信道利用率。只要这个区间的第一个分组得到了确认，就可以把发送窗口推进，直到这个区间的第一个分组没有得到确认。

接收方一般采用**累计确认**的方式，这意味着每收到几个分组，会对按序到达的最后一个分组发送确认。这样存在缺点也有优点，容易实现，即便确认丢失也不需要重传，但是无法反映出接收方所有已经收到的所有分组的信息。如果中间存在断掉的，会从这个地方开始后面都发一遍。

5.5 TCP报文段的首部格式

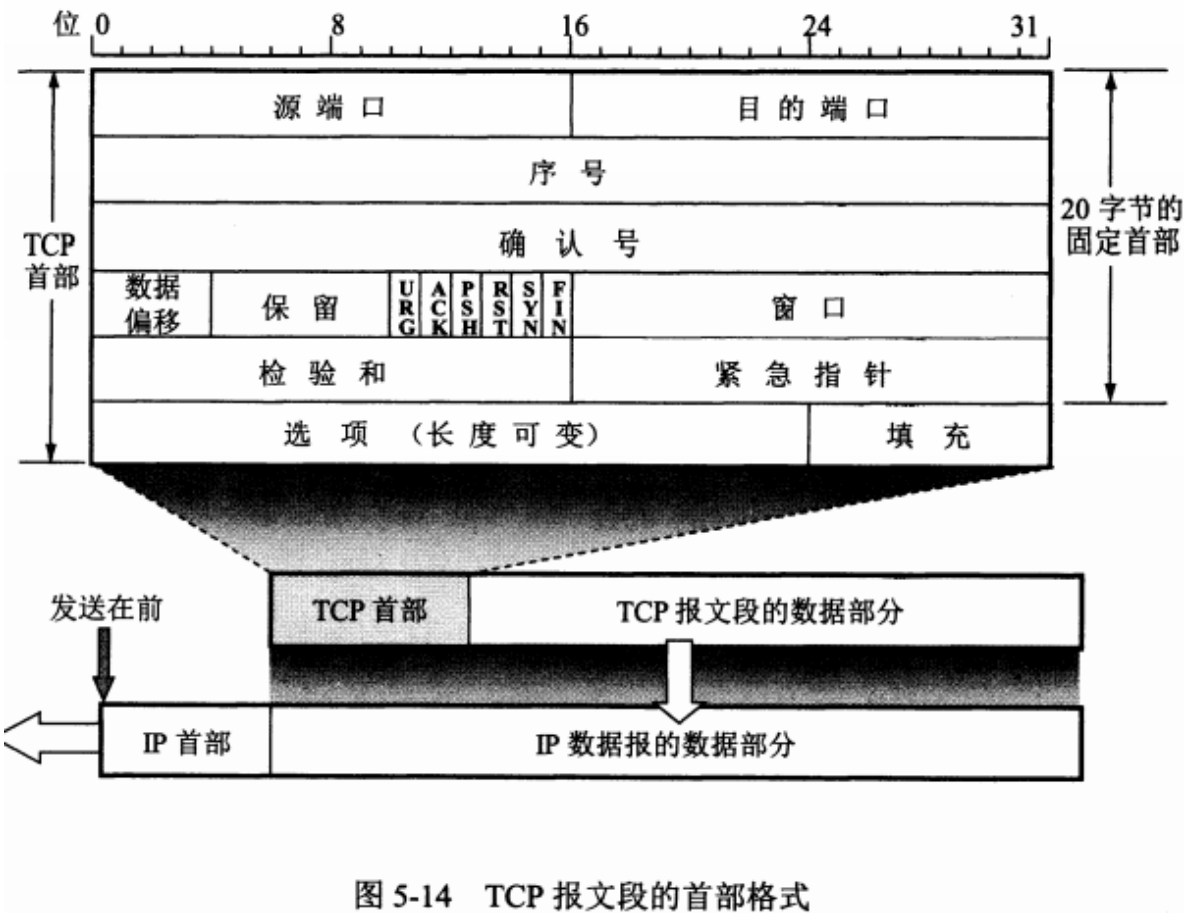

图 5-14　TCP 报文段的首部格式

序号：四个字节，一共2^{32}个序号，如果到头了，下一个会变成0，这个序号指的是第一个字节的序号，之后每个字节序号+1。

确认号：四个字节，希望收到对方下一个报文段的第一个字节的序号。需要注意的是，如果确认号为N，那么到序号N-1的而所有数据都已正确收到。4GB虽然看上去很小，但是序号可以重复使用的时候，旧序号早就通过网络到终点了。

数据偏移：占4位，以四字节为单位，意义是TCP首部的长度。这意味着最大首部长度60字节。

保留：6位，目前设置为0，用于标志位。

紧急URG：当这个字段为1时，表明紧急指针字段有效，系统报文段有紧急数据，需要尽快传送。在一个发送了很长一段程序要在远地的主机上运行，但是后来发现了问题，需要Ctrl+C掉，这个就是紧急报文，如果按照顺序传送，那么会浪费很多时间。关于这个最好看看RFC 6093。

确认ACK：**仅ACK=1时确认号字段有效。**在连接建立后所有传送的报文段都必须ACK=1。

推送PSH：如果这东西为1，那么会立刻整理缓冲区交付。用的很少。

复位RST：如果这东西为1，等着TCP断掉之后重建吧。**这是因为TCP连接出现严重差错。**

同步SYN：连接建立时产生同步信号。**当SYN=1而ACK=0时，表明这是一个连接请求报文段。如果对方同意连接，那么会送一个SYN=1和ACK=1的报文段，表示连接接受。**

终止FIN：如果这东西为1，说明东西传完了，要求**释放连接。**

窗口：2字节，告知对方自己现在还能接收多少字节。这是因为接收方的缓存空间是有限的。

检验和：检验和检验的内容包括首部和数据两个部分。在计算之前，要把UDP一样的伪首部也纳入计算，计算方法也是类似的。对于不同的协议，伪首部可能不一样（比如v4和v6）。

紧急指针：2字节，仅URG=1才有意义。它指出本报文段的紧急数据的字节数，即数据开始之后到紧急数据结束的字节数。数据处理完成之后TCP恢复正常操作。窗口为0也可以发送紧急数据。

5.6 TCP可靠传输的实现

5.6.1 以字节为单位的滑动窗口

所谓滑动窗口$[l,r)$，就是两个只能前移的指针框定的范围，这个范围之前的已经发送并收到确认，之后的还来不及处理，不允许发送。中间的所有数据在确认没有问题之前都不能删除，以备超时重传。如果收到的分组出现了差错就要丢弃。

l发生的变化只有一种，即收到新的确认，l前移；$r=l+$新的窗口，但是强烈不赞成r变小，因为变小的这一段有可能对方已经接收到了，这会导致一些神奇的问题。如果l字节的数据没法接收到，这意味滑动窗口永远不可能前进。

如果A发送了，B也收到了，并且滑动窗口前移，但是回送的内容被滞留在网络中。此时A的滑动窗口无法迁移，并在**超时计时器归零**之后重新发送，直到拿到了B的确认。

显然，发送缓存应当存储应用程序准备发送的数据和TCP已经发送但是没有确认的数据。接收缓存用来存放按序到达但是还没被拿走的和未按序到达的数据。

不过整个过程中需要注意的是：

- 发送方和接收方的窗口大小不一定一样，这个是存在滞后性的。而且对方发过来的窗口大小不一定能完全决定自己的窗口大小，如果网络不行那么还要继续缩小窗口。
- 如果数据不按序到达，接收方不要一律丢弃，最好把这些数据先**临时存放**起来。但是具体怎么执行，TCP没有明确规定。
- TCP接收方必须有累计确认的功能。接收方可以在合适的时候发确认，也可以把数据捎带上，但是不应过分推迟，因为这会导致发送方不必要的重传。TCP标准规定，这个时间不超过0.5s。如果收到一连串具有最大长度的报文串，就必须每隔一个报文段就发送一个确认。其次大多数应用程序很少同时在两个方向上传输数据。

5.6.2 超时重传时间的选择

超时重传的时间选择是TCP的最大问题之一，虽然概念极其简单。在这里TCP采取了一个自适应算法，这取决于过去的样本和现在的样本，而且都是RTT。

$$RTT_{S,new} = (1-\alpha)RTT_{S,old} + \alpha * RTT \quad (1)$$

$$RTT_{D,new} = (1-\beta)RTT_{D,old} + \beta * |RTT_S - RTT| \quad (2)$$

$$RTO = RTT_S + 4RTT_D \quad (3)$$

其中(1)(2)都是迭代，(3)恒定满足，推荐的参数是$\alpha = \frac{1}{8}, \beta = \frac{1}{4}$，要求。$RTT_S$初值就是第一次的$RTT$，$RTT_D$初值是$RTT$的一半。这里的$RTT$指的是**最新样本的往返时间。**

对于重传确认的报文，不论是按照发送报文还是重传报文，都会有一定的误差。因此Karn提出：只要发生了重传报文，不采用对应的RTT。这样会有另一个问题：就是如果报文的时延突然增大了很多，回来的报文超过了重传时间，这样RTT永远都的得不到更新。这个问题的解决办法是每次重传RTO时间**加倍。**

5.6.3 选择确认SACK

在已经了解的TCP协议中，有一个不影响传输但是比较尴尬的事情：如果仅仅使用确认号，那么有很多收到的断断续续的段落都无法得到确认，这会让发送方觉得没有收到。这种情况的避免交给了选择确认，作为TCP的扩展选项，这个东西不能太多，否则碰到了40字节的红线不是好玩的。

在TCP的40扩展字节中，一个指针需要4字节，而框定一个左右区间需要两个指针，因此SACK只能框定四个区间。

为了开启这个操作，需要双方的TCP打开SACK选项，这个在TCP建立时就要确定。SACK文档并没有指定发送方应当怎么响应SACK，因此大多数的实现还是重传所有。

5.7 TCP的流量控制

5.7.1利用滑动窗口实现流量控制

流量控制的本质是不让网络卡住。如果出现了零窗口报文段，那么启动一个**持续计时器**，如果计时器归零，就发送一个零窗口探测报文段，对方需要确认并且给出现在的窗口值，如果为0就重置重新倒计时。

5.7.2 TCP的传输效率

接下来考虑一个情况：如果要发送的内容非常少，而且频次非常高，这个时候TCP和IP的20字节报文头占了大头，会把网络质量拉下去。这样的问题会在两种情况发生：

- 远程命令行，发送一个字符会产生发送和回显两段请求，这时发送的内容则是非常多的。这种情况的解决方法是Nagle算法，**先发送一个字符，确认了以后把剩下的全部发出去**，并且只有在前一个报文段确认后才继续发送下一个报文段。当然如果数据太多，一样可以直接打包立刻发送。
- TCP缓存已满，不断地发送只有一个字节的窗口报文，这个时候最好让接收方**等待一段时间**，空出一个最长报文段或者一半空闲的空间即可。

5.8 TCP的拥塞控制

5.8.1 拥塞控制的一般原理

对于网络拥塞的定义，就是**网络对于某一资源的需求超过了该资源能提供的可用部分，网络的性能就要变坏**，发生这种情况时，我们会说发生了拥塞。

网络拥塞不能通过**简单的增强硬件性能**来解决。

- 单独过分增加缓存会导致缓存队列太长，处理器来不及发出去，于是重传继续塞满队列。
- 单独增加处理器性能除了提升成本，那么木桶的短板会转移到其他地方。
- 网络拥塞常常趋于恶化。如果发生了网络拥塞，东西发不出去，处理机会丢弃新到的分组，于是重传，继续加剧拥塞。
- 如果发现了网络拥塞，则更需要小心，因为**你发送的检测拥塞的报文可能会加剧网络拥塞。**

拥塞控制不是流量控制，但是二者有关，考虑的点则是不一样的。流量控制考虑的是两个通信的主机之间如何控制发送速率使得对方可以接收，而拥塞控制则是网络中的路由器和链路不要过载。

进行拥塞控制必须要付出代价，这正是在负载较低时实际拥塞控制的吞吐量小于没有拥塞控制的原因，但是拥塞到头了，如果没有控制就会发生**死锁**，这个是所有人都不希望看到的。

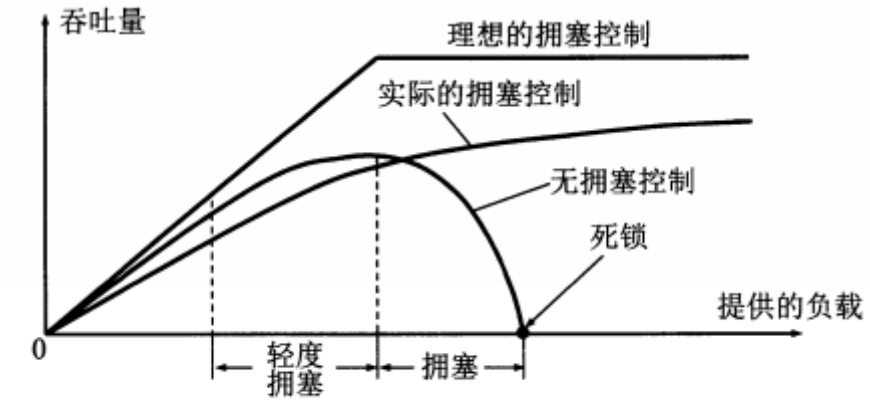

图 5-23 拥塞控制所起的作用

控制拥塞从大的方面来看有两种方法：**开环控制**和**闭环控制**。开环控制指设计时把有关拥塞的因素都考虑到，力求网络在工作时不发生拥塞，但是一旦系统运行起来，就不能再改了（除非关机）。闭环控制基于反馈回路，主要有以下措施：

- 检测网络系统以便检测到拥塞在何时、何处发生。
- 把拥塞发生的信息传送到可采取行动的地方。
- 调整网络系统的运行以解决出现的问题。

对于具体的行为，可以考虑周期性探测或者专门预留一段检测拥塞的字段。前者可能会加剧拥塞，后者拉低一点空间效率。但是一定要注意的是：过于频繁的控制会导致系统产生不稳定的震荡，但是过于迟缓则没有意义。

5.8.2 TCP的拥塞控制方法

TCP的拥塞控制主要由四个部分：慢开始、拥塞避免、快重传、快恢复。对于其中不理解的，这两张图值得一看。这两个图里面，cwnd是拥塞窗口，在假设条件下，就是发送窗口，并且接收方的接收缓冲是无穷大的；ssthresh是慢开始门限。

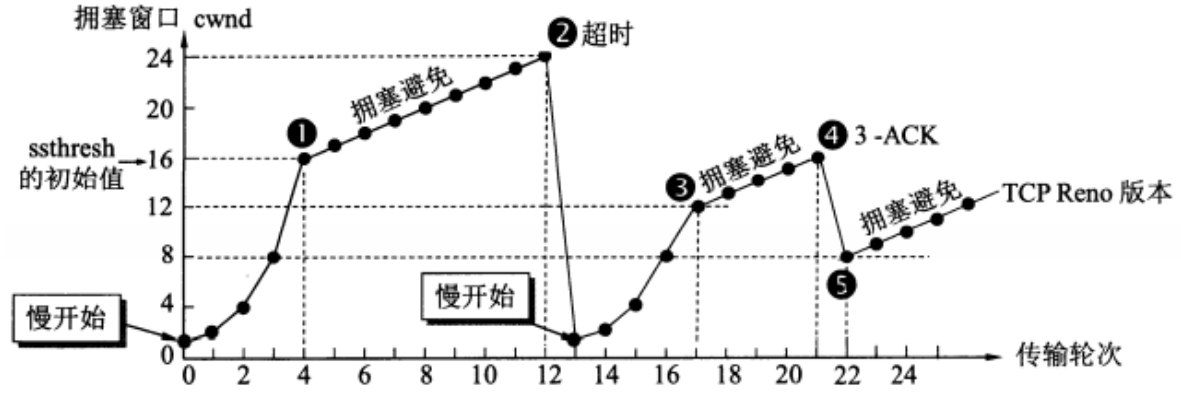

图 5-25　TCP 拥塞窗口 cwnd 在拥塞控制时的变化情况

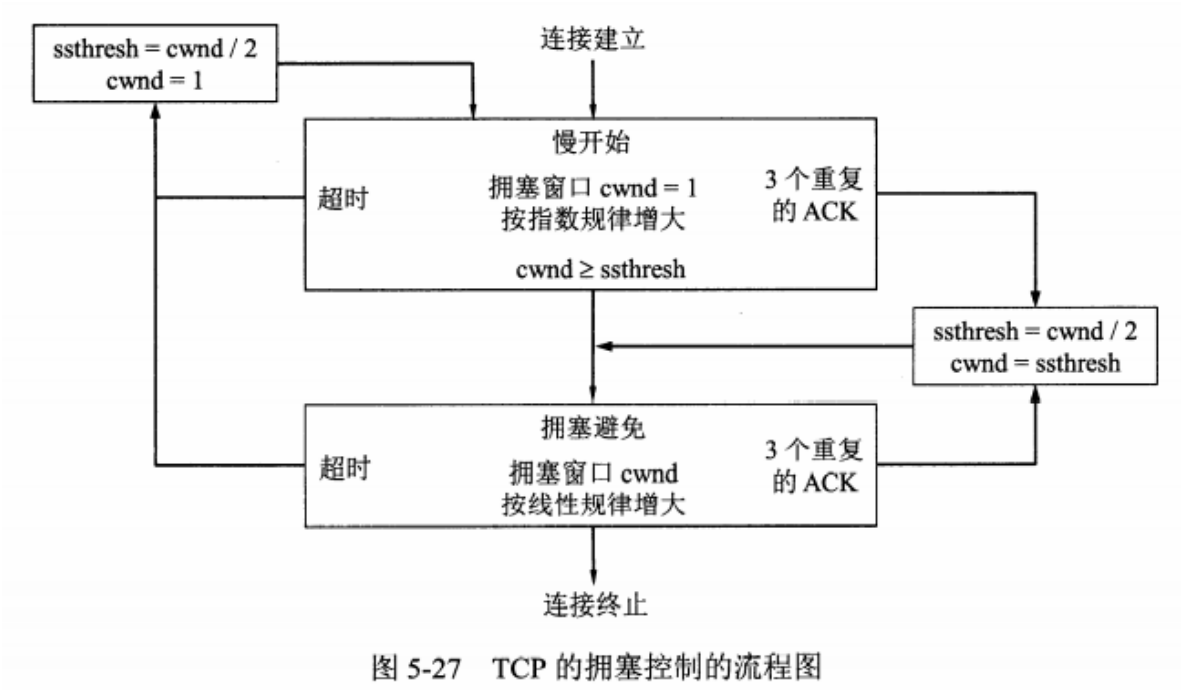

图 5-27　TCP 的拥塞控制的流程图

这里是对于为什么会转变算法的解释：

- 一开始如果是线性规律增长可能会过慢，采用不断加倍可以快速逼近网络上限，减少时间。
- 到了慢开始门限之后，如果继续快速增加可能会把网络快速弄崩，所以 只能缓慢增长使网络比较不容易出现拥塞。
- 一旦发现超时，说明拥塞产生，门限值修改并重新从头开始。这也给网络一定的缓冲时间。
- 如果发现连续三次确认号一样，说明对方的报文接收出现了断层，应当立即进行重传，但这时不是网络拥塞（因为有东西回来），所以没有必要从头开始。
- 这里的cwnd就是滑动窗口的大小，在里面的处理和5.6的实现是一样的。

5.8.3 主动队列管理AQM

这玩意主要是为了发送队列不爆仓，极力减缓的方法：其核心是两个门限。如果在两者之间，那么以一定概率丢弃报文；如果大于高点，那么全部丢弃；如果低于低点，那么全盘接收。

这便是RED算法的内容，其中最难处理的是这个丢弃概率，虽然思路不错，但是实际效果不是很好，在2015年这个算法已经不再推荐，但是还没有一种算法能够成为IETF的标准。

5.9 TCP的运输连接管理

5.9.1 TCP的连接建立

TCP的连接建立要解决以下三个问题：

- 对方知道自己的存在。
- 双方协商参数。
- 能够对运输实体资源进行分配。

通常来说，主动发起连接的乘坐客户，被动的叫做服务器。

TCP的连接建立是三报文握手。形象地说明就是这个图：

1. A和B一开始都要先创建好TCB（传输控制块），B进入监听状态。
2. A发送一个**SYN=1**的报文，消耗掉一个序号x。
3. B拿到后如果同意连接，就发送**SYN=1，ACK=1**的报文，并自己也定义一个初始序号y，并确认x已经收到。
4. 之后A发一个**ACK=1**的报文，x++，并且确认对方序号y。

这便是三报文握手。B会送给A**可以拆成两步**，一步ACK=1，ack=x+1，一步SYN=1，seq=y，便是四报文握手。

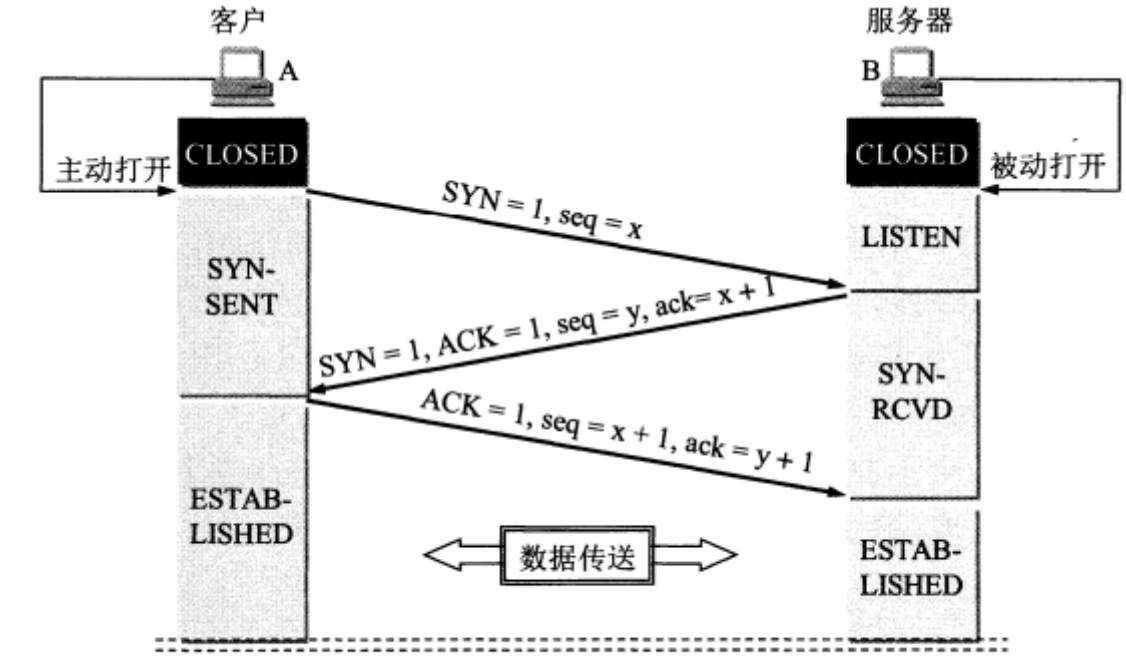

图 5-28 用三报文握手建立 TCP 连接

为什么要三次握手？假定A发送给B的第一条报文滞留在网络中，时间太长导致失效，那么A会认为这个连接没法建立。过一段时间B收到了，并且回送报文，如果只有两次握手，那么B此时已经是准备连接。A拿到报文时会直接丢弃，所以A也不会往B发送数据。

5.9.2 TCP的连接释放

直接看图：

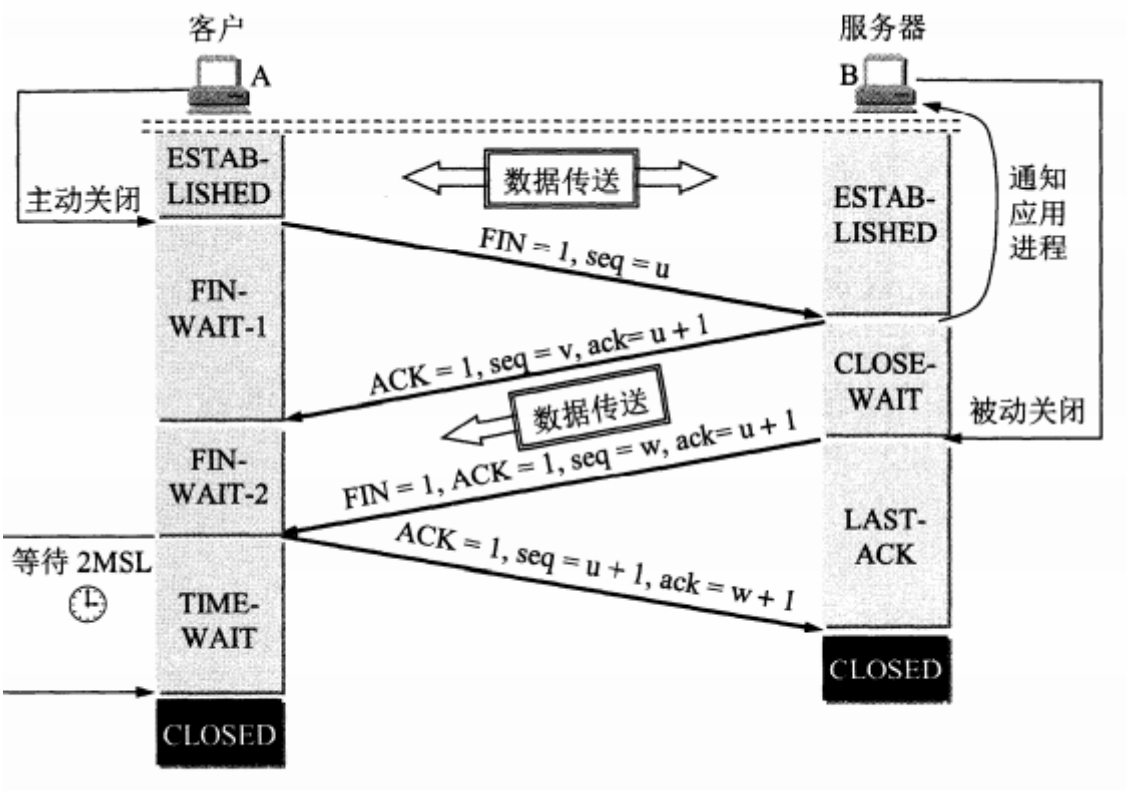

图 5-29　TCP 连接释放的过程

需要注意的是：

- 一开始的序列号u不是自己制定的，同样这个v和w也不是。
- FIN不携带数据但是也要消耗一个序号。关闭是双向关闭，一遍FIN只能告知自己不再发送数据。
- 前两次让服务器知道客户已经传完了。但是如果服务器要给用户传数据用户还是得接着。此时的TCP还是半关闭状态。
- MSL（最长报文段寿命），一开始是2分钟，不过现在可能有点长了，可以缩短一点。如果B到A的报文没有回复，那么B会重新传一遍seq=w，ack=u+1这个报文，以告知A这个回复丢失了或者根本没送到。除此以外，2MSL的时间足够AB消化完各自还没传送完的报文，即让所有报文从网络上消失。

除了**时间等待计时器、坚持（零窗口）计时器、重传计时器以外，还有一个保活计时器。**如果A有很长一段时间（比如两个小时）没有发送数据，B则发送探测报文段，没过75s发送一次，若一连十个报文段都没有回复，那么只好推断A出现故障，关闭连接。

5.9.3 TCP的有限状态机

别鲨了别鲨了，一张图解决掉就行了。。。

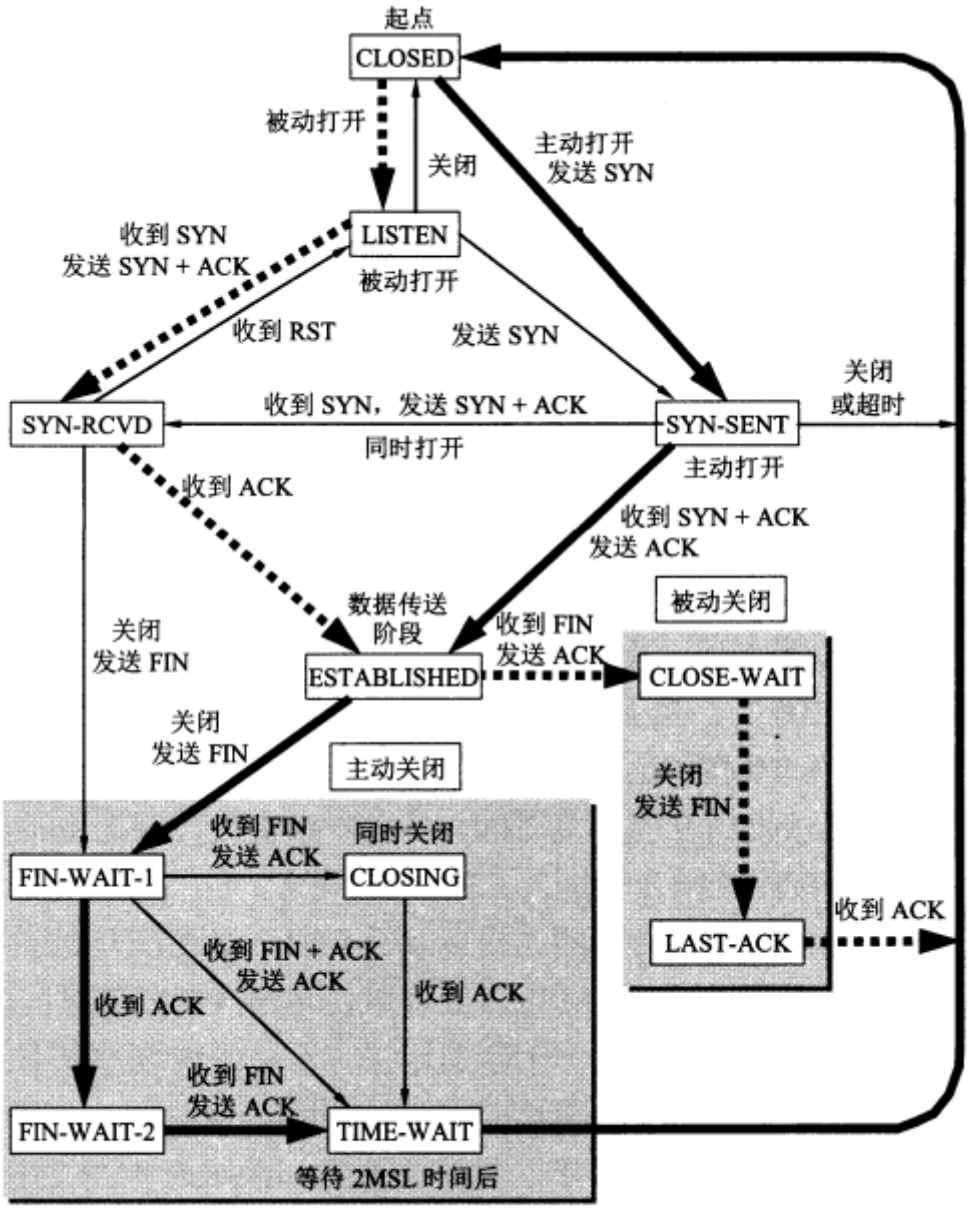

图 5-30　TCP 的有限状态机