TCP 协议确保传输数据的顺序正确与内容完整。

TCP 首部格式

TCP 的首部格式如图 3.2 所示。各个字段的具体作用详见后文。这种形式的首部说明图一般是以 32 位为单位绘制的。究其原因，是首部格式的设计考虑到了 32 位计算机。不过这只是一个表达形式，因此即使现在 64 位系统成了主流，大家也无须改变观念。在实际处理中，所有字段的数据都被看作一维数组，是按照行序直接处理的。

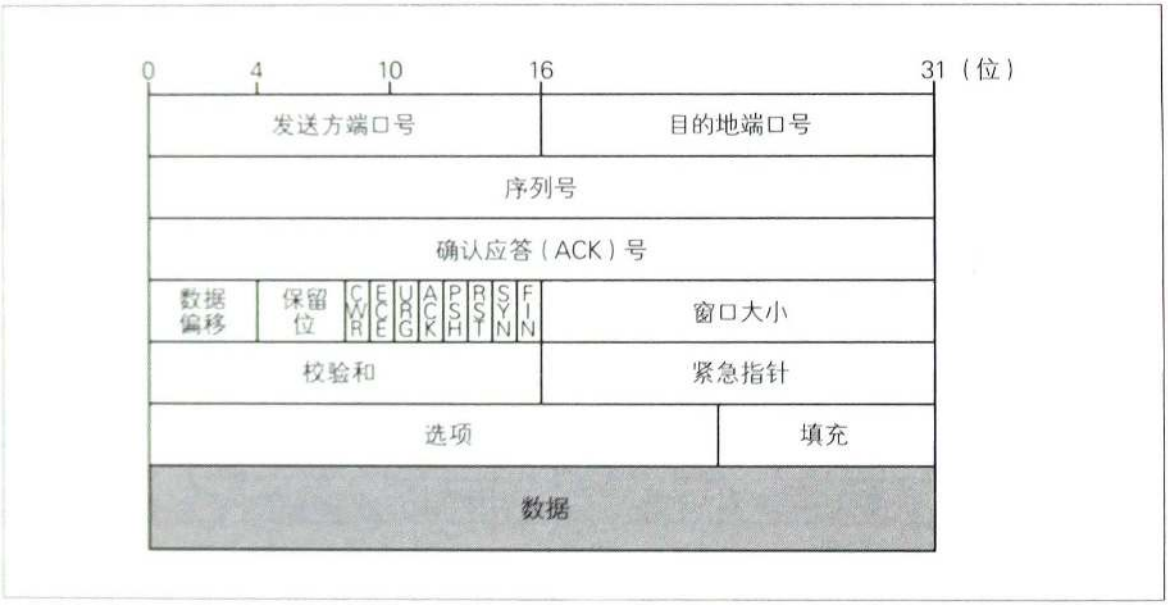

图 3.2 TCP 首部格式

- **发送方端口号**（source port）
  16 位字段，代表发送方的端口号。
- **目的地端口号**（destination port）
  16 位字段，代表目的地的端口号。
- **序列号**（sequence number）
  32 位字段，代表序列号。已发送数据的位置通过此字段来确定，接收方根据此字段来进行顺序控制。当序列号达到最大值时，重新循环回最小值继续使用。

- **确认应答号（acknowledgement number）**

  32 位字段，存储的是与已收到的序列号对应的确认应答（ACK）号。请注意，准确来说，存储的是“接下来期望对方发送的序列号”。换句话说，这一消息代表了“我已经完整无误地收到了这个序列号以前的数据”（在理解 3.4 节以后讲解的详细流程时，这一点很重要）。

- **窗口大小（window）**

  16 位字段，用于通知[①]发送方“接收方最大能接收的数据大小”，单位是字节。发送方无法发送大于这个窗口大小的 TCP 报文段数据。发送方虽然会采用拥塞控制算法（3.4 节）调整窗口大小（拥塞窗口），但是这个字段通知的窗口大小优先级更高。详见 3.3 节的“流量控制”部分。

- **数据偏移（data offset）**

  TCP 首部由于包含选项部分，所以是可变长的，长度为 20 到 60 字节。因此，需要有信息来指示 TCP 首部之后的数据部分究竟从哪里开始，所以才会有这个数据偏移字段。换句话说，这个字段也代表了 TCP 首部的长度。字段长度总共有 4 位（bit），即取 0 到 15 的值，将其乘以 4 就可以得到最终的 TCP 首部长度。例如，TCP 首部长度为 20 字节，那么这个字段的值就是 5（0101）。

- **保留（reserved）**

  为了今后扩充使用而保留的字段。TCP 标准文档（RFC 768）定义了后文介绍的从 URG 控制位到 FIN 控制位的字段。2001 年的 RFC 3268 文档追加定义了 CWR、ECE 等字段（详见后文），以实现更先进的拥塞控制。

如下文所述，各个控制位（control flag）合起来构成 8 位字段，当对应的位设置为 1 时，这个控制位就变为有效位，指示设备完成对应的操作。

① 也可以说是“广而告之”。

- URG（Urgent Pointer field significant，**紧急指针字段标志**）
  表示本数据包含需要紧急处理的数据。紧急数据的位置由紧急指针字段表示。
- ACK（Acknowledgement field significant，**确认应答字段标志**）
  表示确认应答号字段有效。除了连接建立时的第一条TCP报文段以外，其他报文段的此字段必为1。
- PSH（Push function，**推送功能**）
  代表需要将收到的数据立即交由上一层即应用程序处理。如果为0，则代表允许存储到缓冲区，过一段时间后再进行处理。
- RST（Reset the connection，**重置连接**）
  用于强制切断连接的字段。当检测到出现异常时，就对数据包内的这个控制位进行置位并将数据包发送出去。例如，当TCP发现一个想要与未使用的端口号进行通信的请求时，由于很显然在这种情况下无法完成通信，所以TCP就会对此控制位进行置位，然后将数据包发送回去，以便强制终止连接。
- SYN（Synchoronize sequence numbers，**同步序列号**）
  在连接建立时使用。在连接建立时，通信开始时的序列号字段会被设置为初始值。
- FIN（no more data from sender，**发送方无更多数据**）
  代表当前的TCP报文段是通信过程的最后一个报文段。在断开连接时使用。
- CWR（Congestion Window Reduced，**拥塞窗口减小**）
  用于通知拥塞窗口大小的减小。它与下面的ECE字段一起使用。
- ECE（ECN-Echo，**显式拥塞通知回应**）
  用于通知网络拥塞的发生。当IP层检测到丢包之后，IP首部的ECN（显式拥塞通知）控制位就被激活。在把数据包交给上一层时，TCP首部的ECE控制位也被激活，并通知到发送方。也就是说，检测数据丢包的是网络层，但是负责通知的是传输层，两者分担不同的任务。跨层处理在一般情况下不会进行，但是有个例外的

情况，那就是在不同层之间进行数据传递时是可以的。具体的例子就是 ECE，通过在首部附加信息便可以实现更为灵活的处理。

- **校验和（checksum）**

  16 位字段，用于确认收到的数据是否正确无误。TCP 将 IP 首部的一部分信息[①]（发送方 IP 地址、目的地 IP 地址、协议号和 TCP 包长度）结合起来，按照一定的规则算出一个校验数据，通过确认这个校验数据便可以判断数据是否出现损坏。此功能在 TCP 中是必不可少的。通信链路中的噪声、经过的通信设备中的程序 bug 等问题是导致数据损坏的原因。噪声引起的错误通常可以通过下面的数据链路层的补偿进行处理，因此在传输层，设备故障和程序 bug 是导致数据损坏的主要原因。如果数据损坏，只要发送重传请求，就能收到正确的数据。

- **紧急指针（urgent pointer）**

  只在控制位 URG 字段被设置为 1 时才会使用。这里存储的是数据字段中紧急数据的起始位置。准确地说，紧急指针是以字节为单位的长度值，序列号代表紧急数据的起始，两者组合起来就表示了紧急数据的范围。紧急指针一般会在通信或处理中断时使用。不过具体如何处理，可以交由应用程序决定。

- **选项（options）**

  选项字段是最大可达 40 字节的可变长字段，用于扩展 TCP 的功能。*MSS* 的值在连接建立（详见 3.2 节）时确定，此时便会用到这个选项字段。此外还有一个窗口扩大选项，它用于改善 TCP 的吞吐量。在通常情况下，窗口大小字段如前文所述只有 16 位，因此单次最大传送量为 2^{16}=64 KB。通过此选项，它便可以扩展到最大 1 GB，实现在 *RTT* 较大或宽带环境下更高的 TCP 吞吐量。此外，

① IP 首部的这部分信息和 TCP 首部组合在一起，被称为“TCP 伪首部”。发送方 IP 地址、目的地 IP 地址、协议号、发送方端口号和目的地端口号合在一起便是连接的“身份证”，用于识别具体的连接。为了确保这些信息准确无误，这里会基于伪首部计算校验和。IPv4 和 IPv6 采用了相同的设计思路，IPv6 的 IP 地址有 128 位，因此对应的伪首部的长度也更长。

SACK、时间戳等功能（详见1.6节）也需要通过选项字段实现。

- **填充（padding）**
  为了将TCP首部的长度扩展到32位的整数倍，需要以0作为空数据进行填充。
- **数据（data）**
  TCP有效载荷，它存储包含上一层的首部在内的数据，长度小于等于*MSS*。

UDP首部格式

UDP的首部格式如图3.3所示。与图3.2的TCP首部相比，我们可以看出UDP首部更加简单，其中只有保存首部与数据的总长度的16位字段、发送方与目的地端口号，以及校验和。

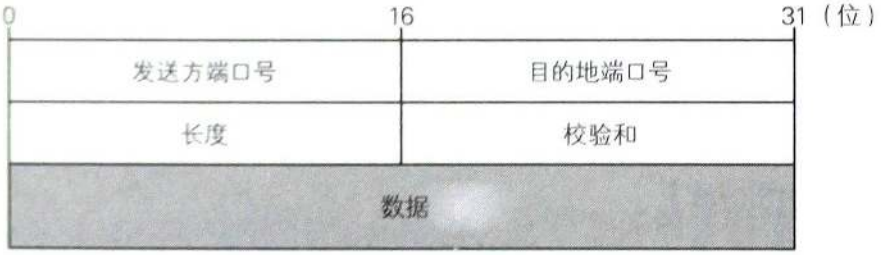

图3.3 UDP首部格式

接下来介绍一下各个字段的作用。

- **发送方端口号（source port）**
  16位字段，表示发送UDP数据报一方的端口号。
- **目的地端口号（destination port）**
  16位字段，表示接收UDP数据报一方的端口号。
- **长度（length）**
  16位字段，表示UDP首部与数据部分的总长度。

- **校验和（checksum）**

  与 TCP 首部的校验和一样，用于确认全部 UDP 数据报（UDP 首部、IP 地址和端口号）的数据是否损坏。UDP 中校验功能是可选功能。不使用校验功能，就无须计算校验和，因此可以实现更高速的数据传输，但相对地，数据可靠性也会降低。如果要关闭校验，将此字段全部设为 0 即可。

3.2 连接管理

3 次握手

TCP 协议建立一对一的通信连接，并管理连接的建立与断开。连接管理本身也是用于确保可靠性的功能之一。本节将介绍 TCP 中的连接管理方法。

建立连接 3 次握手

TCP 协议在开始通信之前，会与通信目标建立连接。TCP 是支持全双工通信的协议，因此收发数据双方都需要发送建立连接的请求。连接建立的流程如下所述。

❶ 发送方发送一个设置了 SYN（请求建立连接）的 TCP 包。

❷ 接收方发送一个设置了 ACK 的 TCP 包，这个包同时设置了用于请求建立连接的 SYN。

❸ 发送方针对接收方发过来的 SYN，返回设置了 ACK 的 TCP 包。

如上所述，因为是通过 3 次发送数据建立了连接，所以这个过程称为 **3 次握手**（three-way handshake）。具体流程如图 3.4 所示。其中，首先发送 SYN 包并开始建立连接的过程称为主动打开（active open）；与之相对，收到 SYN 包并开始建立连接的过程称为被动打开（passive open）。如前文

所述，SYN 和 ACK 都在 TCP 首部的控制位字段部分，因此步骤❷利用这一优势同时发送 ACK 和 SYN 以提高效率。

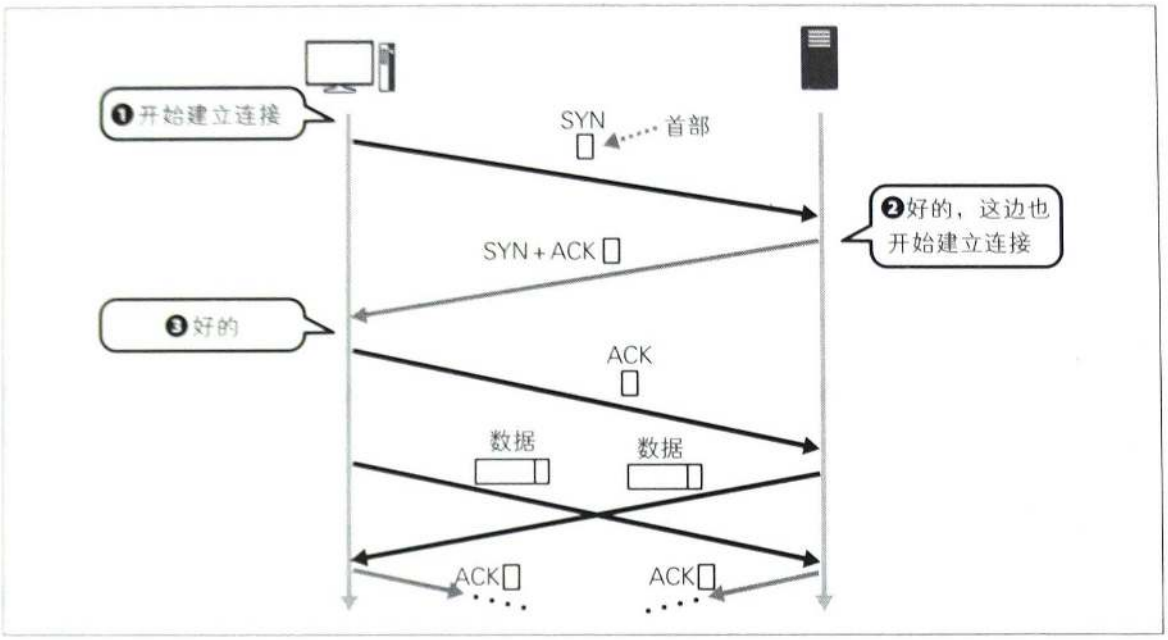

图3.4 3次握手

断开连接 半关闭

断开连接需要通信双方分别进行处理，这种方式称为半关闭（half-close）。这是因为，TCP 是全双工的数据传输。也就是说，这种数据传输是一个双向的过程，即使其中一方停止发送数据，另一方也可能仍在进行数据传输。因此，要想通过半关闭的方式彻底断开连接，需要有 4 次来回发包的过程。接收方需要在收到与自己发送的 FIN 所对应的最后一个 ACK 之后断开连接；发送方则在发送最后一个 ACK 之后，先等待一段时间再断开连接。这样做的意义是确认发送方发送的 FIN 没有进行重传，换句话说，就是确认 ACK 正确无误地传回给了发送方[1]。当双方都断开连接时，TCP 的连接就断开了。

1 原文的这个描述有误。对照图 3.5 和前文描述，发送方指客户端，接收方指服务器。因此前面的“等待一段时间”应该是为了确认接收方（服务器）不再进行 FIN 重传，也就是确认发送给接收方的最后一个 ACK 没有丢失。——译者注

❶ 发送方首先发送 FIN（请求断开连接）。

❷ 接收方发送 ACK，然后发送 FIN[①]。

❸ 发送方收到 FIN 之后，发送最后一个 ACK，并在等待一段时间后断开连接。

具体过程如图 3.5 所示。首先发送 FIN 包的一方的关闭过程称为主动关闭，而另一方的关闭过程称为被动关闭。

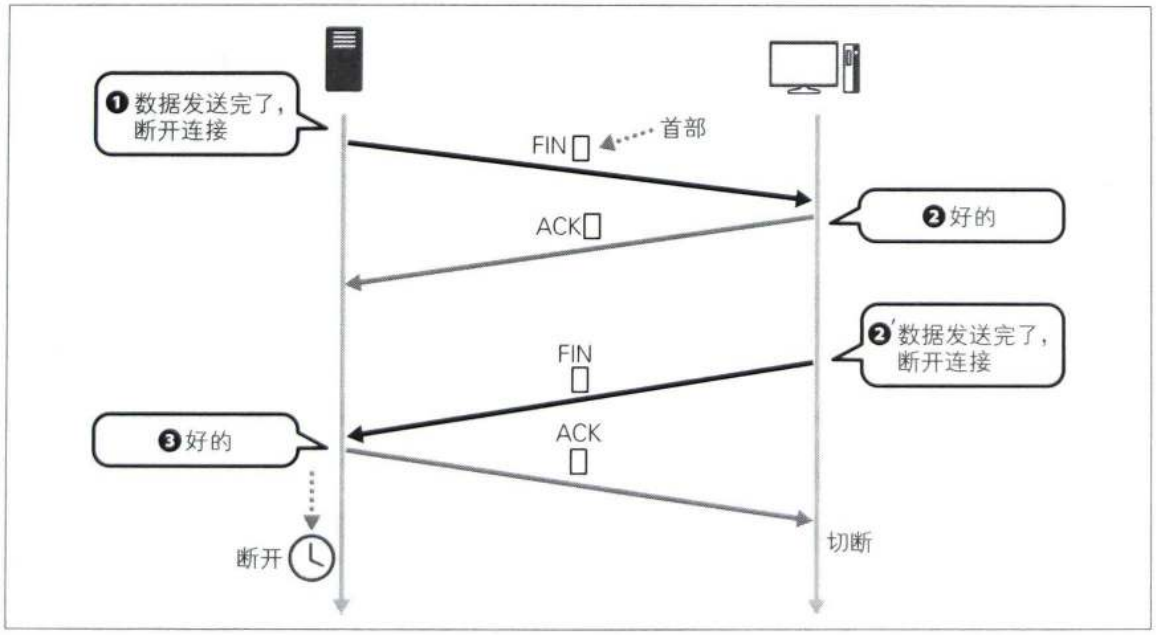

图 3.5 半关闭

端口与连接

应用程序的区分是通过传输层中代表着地址、被称为端口的序号来实现的。第 1 章曾提到，端口号是与应用层协议相对应的。此外，端口号还与网络层的 IP 地址、协议号（TCP 或 UDP）一起用于区分不同的通信。发送方和目的地都有 IP 地址和端口号，换句话说，下页这 5 项的组合便可以确定唯一的一个通信。如果某个通信的其中任意一项有所不同，便可

① 在发送 FIN 之前，理论上发送方（客户端）仍可以接收数据，接收方是否继续发送数据则取决于上层。——译者注

以认为它是另外一个通信。

- 发送方 IP 地址
- 目的地 IP 地址
- 协议号
- 发送方端口号
- 目的地端口号

例如，即使发送方 / 目的地 IP 地址、发送方 / 目的地端口号一样，只要协议号不同，对应的就也是不同的连接。此外，哪怕发送方 / 目的地 IP 地址、协议号、目的地端口号都一样，但如果发送方端口号不同（也就是说是不同的应用程序），那么也是不同的连接。

3.3 流量控制与窗口控制

不宜多也不宜少，适当的发送量与适当大小的接收方缓冲区

本节将介绍一下 TCP 数据传输的基本思路。传输的数据量不宜过多，也不宜过少，而是需要根据实际情况适当地进行调节。那么，TCP 究竟需要基于什么信息来确认传输的数据量呢？本节和下一节将具体介绍。

流量控制 窗口与窗口大小

接收方设备通常具有临时存储数据的“缓冲区”，用来处理数据。不同的设备，缓冲区的大小也各有不同。接收方首先将收到的数据暂存到接收缓冲区中，然后再交给上层应用程序。接收方如果收到比接收缓冲区更大的数据，就会“无法消化”，导致数据丢失，进而导致发送方进行无意义的重传。

TCP 在处理网络拥塞之前，首先必须掌握通信设备的容量上限。因此，接收方需要告诉发送方自己能接收的数据量大小，以实现调节发送

量。这就是**流量控制**。

TCP 在数据传输中引入了“窗口”这一概念。发送方在发送 TCP 报文段时，发送数据的大小只要在窗口大小范围内，就可以直接发送数据，无须等待 ACK 返回。在 TCP 的首部中有用于通知窗口大小的字段，接收方会将自己最大能接收并缓存的数据量放在这个字段中，和 ACK 一起发送回去（图 3.6）。这就是 1.5 节提到的接收窗口大小 *rwnd*。

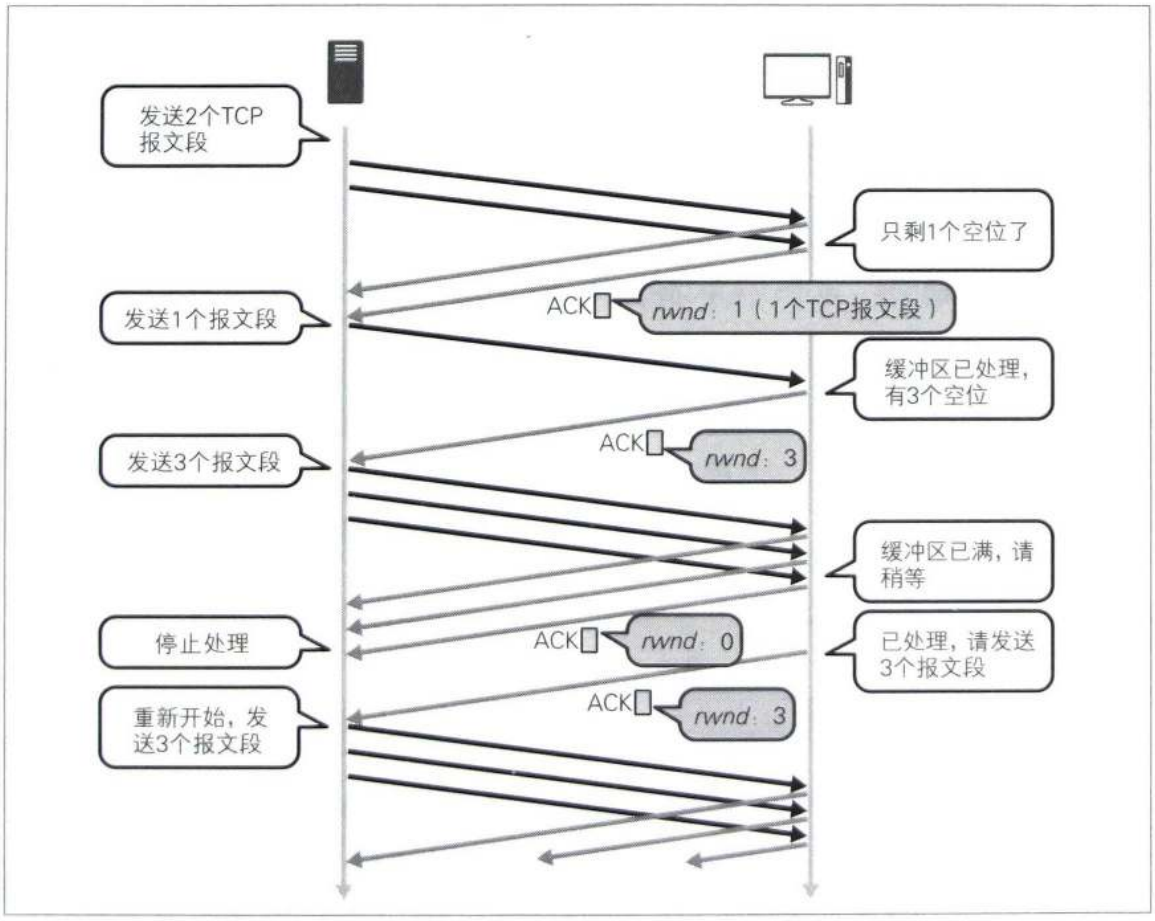

图 3.6 流量控制

缓存与时延

当接收方收到超过接收缓冲区大小的数据，或者缓冲区太小时，就会发送接收窗口大小 *rwnd* 为 0 的通知。如前面的图 3.6 所示，在这种情况下，发送方会停止发送数据。当接收方有能力继续处理数据时，就会重新

发送ACK包，通知发送方可以重新开始发送数据了。

然而，如果频繁停止发送数据，就会导致**时延**增大，进而导致传输效率下降。要想保持高传输效率，最好尽可能增大接收窗口大小，同时保证缓冲区存储的数据量大小不超过系统最大的处理能力。

窗口控制 滑动窗口

窗口控制指的是发送方一边调整代表了单次可发送数据量的参数，即发送窗口大小*swnd*，一边传输待发送的数据。具体的控制方法称为滑动窗口，图3.7对其进行了简单展示。凡是在窗口内的TCP报文段都会被直接发送出去，无须等待ACK。

在图3.7❶中，4个TCP报文段（3、4、5、6）被一次性发送出去，进入等待ACK的状态。到了图3.7❷，发送方收到3号TCP报文段对应的ACK之后，窗口往右滑动1格，并且扩大1格，随后发送待传输的7号和8号TCP报文段。也就是说，发送方收到ACK之后，就发送新的TCP报文段。

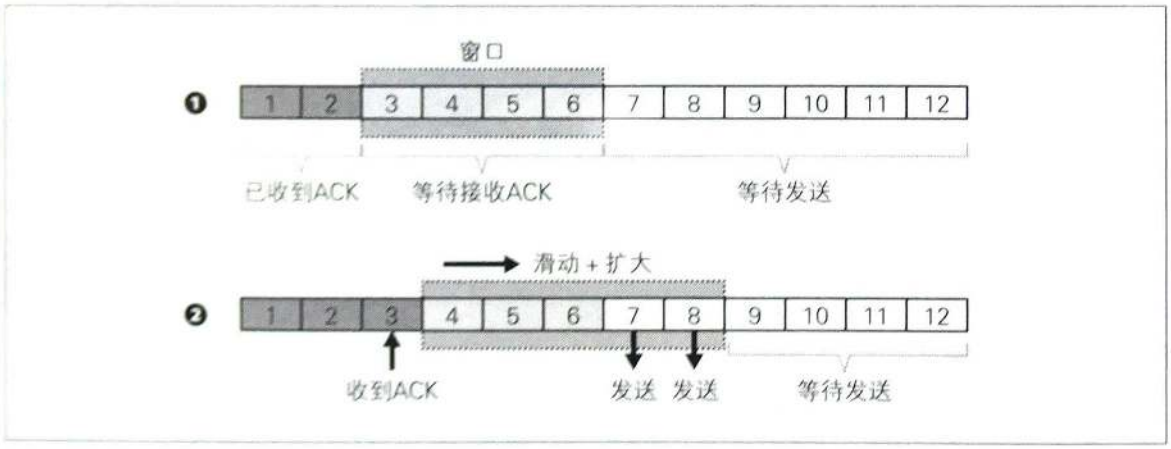

图3.7 滑动窗口

在滑动窗口时增减的窗口容量（窗口大小），是根据拥塞情况由各种算法计算出来的。简单来说，这个窗口大小就是拥塞窗口大小*cwnd*，它是由下一节将介绍的拥塞控制算法主动确定的。但是，如果被告知的接收方缓冲区大小*rwnd*的值比*cwnd*小，则优先使用*rwnd*。

复习：流量控制、窗口控制和拥塞控制

至此，本书出现了多种方法，包括流量控制、窗口控制和拥塞控制。接下来，我们整理一下这些方法之间的关系，详见图 3.8。

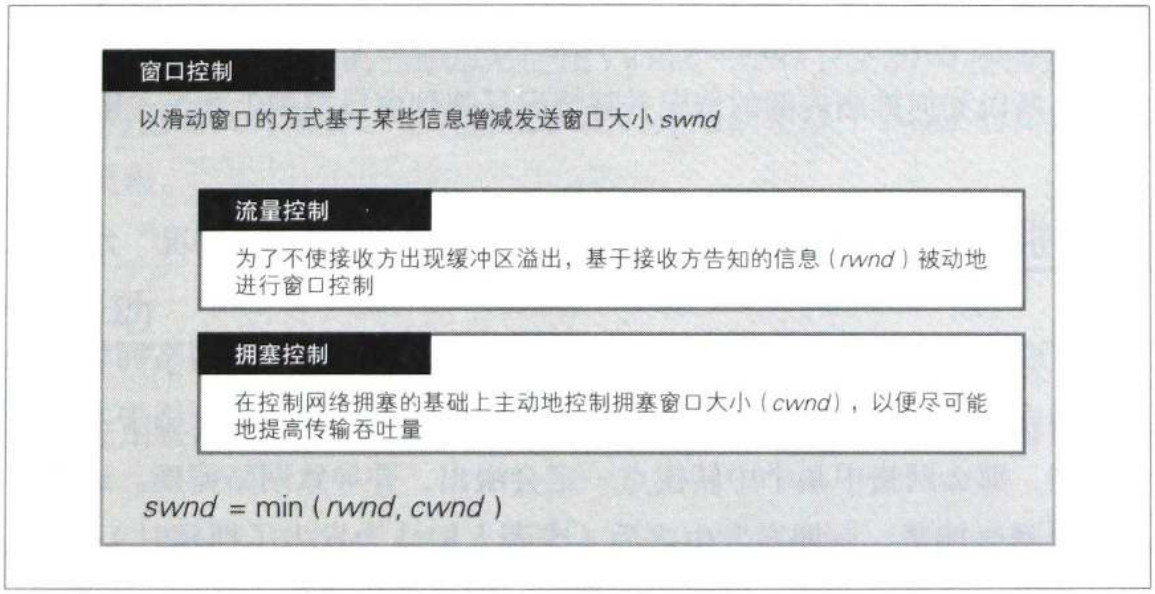

图 3.8 窗口控制、流量控制和拥塞控制之间的关系

首先，窗口控制是上层的概念，核心思路是基于滑动窗口技术传输数据。

其次，确定发送窗口大小的方法有流量控制和拥塞控制两种。流量控制指的是根据接收方告知的可接收和处理的最大窗口大小 *rwnd*，计算出发送窗口大小 *swnd*。显而易见，这是一种被动控制。与之相对，拥塞控制则是指调整拥塞窗口大小，以在确保不出现网络拥塞的前提下，尽可能高效率地传输数据。显然，这是一种主动控制。

最后，比较 *rwnd* 和 *cwnd* 的大小，选择其中较小的值用作 *swnd*。

3.4 拥塞控制

预测传输量，预测自律运行且内部宛如黑盒的网络的内部情况

我们从前文可以看出，“根据接收方缓冲区大小调节数据传输量”是大前提，接下来需要考虑的就是“网络状况”。但是，我们无法看见网络内部，所以如何推测内部的情况并将情况反映到窗口控制上才是关键。

TCP拥塞控制的基本概念 以“完全不清楚网络内部情况”为前提

互联网这种自律性网络，其中的通信数据总量有多大、网络拥堵情况如何是完全无法预测的。但是，我们不难想象，一旦持续向网络中发送大量数据，那么网络中某个中转接点一定会溢出，并导致网络瘫痪。这种情况称为网络拥塞。在拥塞发生之后（或者人们认为发生了拥塞时），发送方如果进行适当的调整，网络拥塞情况一定能有所改善。从这一点来说，拥塞控制十分重要。

TCP的拥塞控制算法是以“完全不清楚网络内部情况”为前提，并基于特定参数调整传输量来实现的。

最典型的算法是基于丢包的算法，也就是以TCP报文段的丢失为契机调整控制方法。其基本的操作是，如果没有TCP报文段丢失，就认为网络比较空闲，可以提高数据传输量；与之相对，如果有TCP报文段丢失，就认为网络比较拥堵，需要减少数据传输量。基于丢包的算法，通过反复增减传输数据量完成通信。只要没有TCP报文段丢失，就一直增加数据传输量，然后根据TCP报文段丢失的出现来判断通信链路的处理极限。

其他的拥塞控制算法包括采用RTT的基于延迟的算法，还有把基于延迟和基于丢包结合在一起的混合型控制方法。第4章将分别介绍这些算法。

接下来介绍几个基本的拥塞控制算法，它们的思路是通过灵活运用以

下几种算法，对拥塞窗口大小 *cwnd* 进行控制。

- 慢启动
- 拥塞避免
- 快速恢复

下面我们分别介绍这几个算法的流程。如前文所述，如果接收窗口大小 *rwnd* 小于 *cwnd*，则优先使用 *rwnd*。但是下文的所有描述都以 *rwnd* 的值更大为前提，也就是说，我们介绍的是基于 *cwnd* 的拥塞控制算法流程。

慢启动

应用程序如果在通信之初就发送大量数据，很有可能导致网络拥塞出现。

为了预防这种情况出现，在通信开始时，应用程序遵循名为**慢启动**（slow start）的算法开始发送数据。发送方首先将拥塞窗口大小 *cwnd* 设置为 1 个 TCP 报文段，然后发送数据。接着，在收到对应的 ACK 后，让 *cwnd* 增大 1 个 TCP 报文段的大小（为了方便表述，下文设 1 个 TCP 报文段 = *MSS*）。

$$cwnd = cwnd + MSS$$

也就是说，发送方在收到 1 个 ACK 之后，就可以发送 2 个 TCP 报文段了。只要数据发送量没有达到接收方告知的窗口大小 *rwnd*，就一直增加传输量。以从发送 TCP 报文段开始到收到对应 ACK 的这 1 个 *RTT* 的时间为单位来看，*cwnd* 的值是呈指数级增大的。慢启动算法可以控制通信初期的数据流量，减少拥塞的发生概率。

慢启动时通信流程、滑动窗口和拥塞窗口大小 *cwnd* 的变化情况如图 3.9 所示（为了简化，这里将 *MSS* 设为 1）。

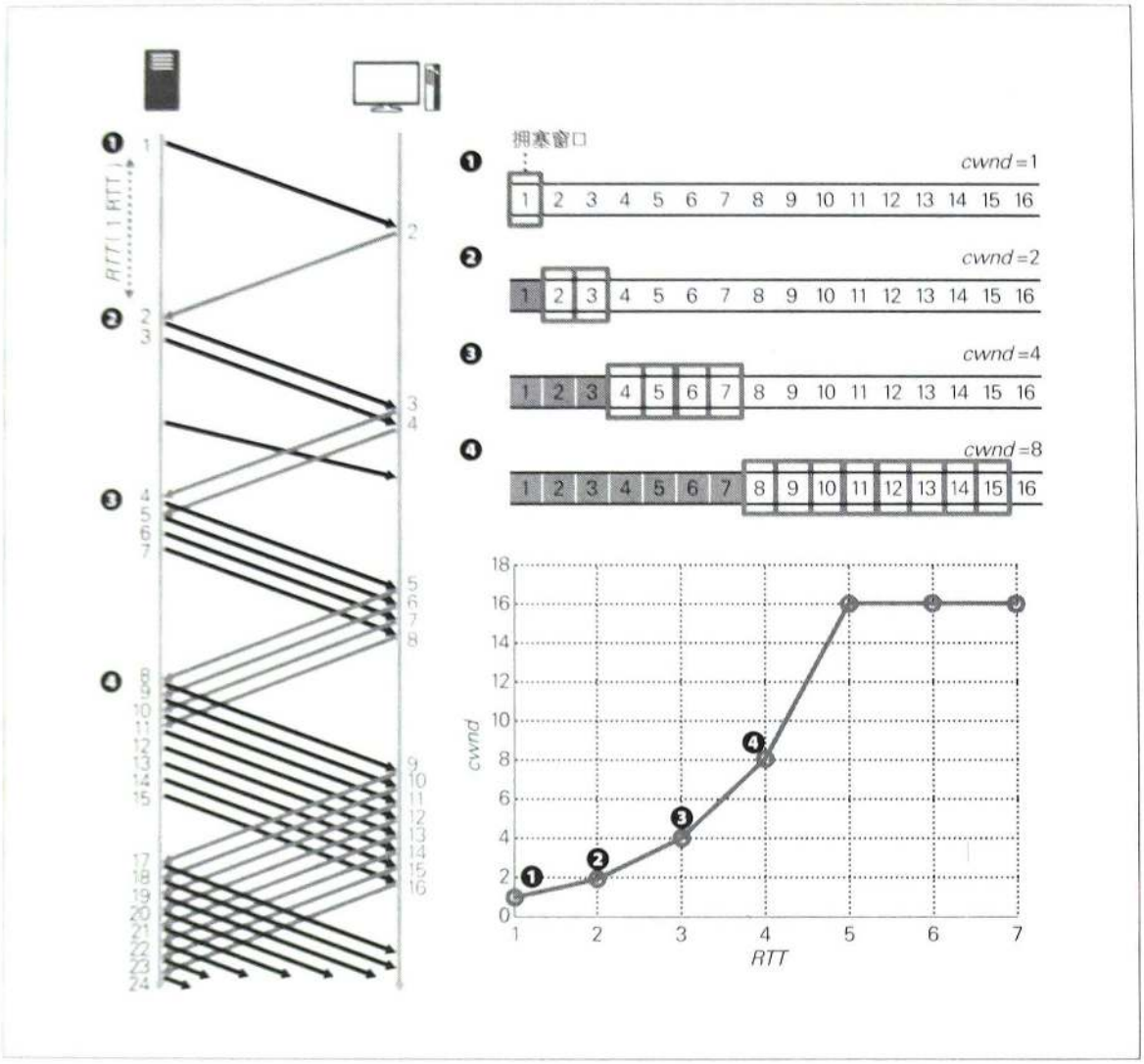

图3.9 慢启动

发送方在*cwnd*为1的情况下开始发送数据（❶），当收到ACK后，*cwnd*增大到2（❷）、4（❸）、8（❹）。请注意，如3.1节所述，ACK的序号代表接收方尚未收到的数据的序列号。发送方每次收到ACK之后窗口都会往右滑动，与此同时发送未发送的数据。当增大到16之后，*cwnd*的值便不再增大，而是保持不变。之后，即使收到ACK，*cwnd*也不会再增大，发送方则会发送与收到的ACK数量相同的新TCP报文段。

拥塞避免

在慢启动的过程中*cwnd*呈指数级增大，因此数据传输量会随着时间