一点，主机 B 必须跟踪几个与连接有关的变量。

主机 A 轮流跟踪两个变量，LastByteSent 和 LastByteAcked，这两个变量的意义很明显。注意到这两个变量之间的差 LastByteSent - LastByteAcked，就是主机 A 发送到连接中但未被确认的数据量。通过将未确认的数据量控制在值 rwnd 以内，就可以保证主机 A 不会使主机 B 的接收缓存溢出。因此，主机 A 在该连接的整个生命周期须保证：

$$\text{LastByteSent} - \text{LastByteAcked} \leqslant \text{rwnd}$$

对于这个方案还存在一个小小的技术问题。为了理解这一点，假设主机 B 的接收缓存已经存满，使得 rwnd = 0。在将 rwnd = 0 通告给主机 A 之后，还要假设主机 B 没有任何数据要发给主机 A。此时，考虑会发生什么情况。因为主机 B 上的应用进程将缓存清空，TCP 并不向主机 A 发送带有 rwnd 新值的新报文段；事实上，TCP 仅当在它有数据或有确认要发时才会发送报文段给主机 A。这样，主机 A 不可能知道主机 B 的接收缓存已经有新的空间了，即主机 A 被阻塞而不能再发送数据！为了解决这个问题，TCP 规范中要求：当主机 B 的接收窗口为 0 时，主机 A 继续发送只有一个字节数据的报文段。这些报文段将会被接收方确认。最终缓存将开始清空，并且确认报文里将包含一个非 0 的 rwnd 值。

位于 http://www.awl.com/kurose-ross 的在线站点为本书提供了一个交互式 Java 小程序，用以说明 TCP 接收窗口的运行情况。

描述了 TCP 的流量控制服务以后，我们在此要简要地提一下 UDP 并不提供流量控制。为了理解这个问题，考虑一下从主机 A 上的一个进程向主机 B 上的一个进程发送一系列 UDP 报文段的情形。对于一个典型的 UDP 实现，UDP 将会把这些报文段添加到相应套接字（进程的门户）“前面”的一个有限大小的缓存中。进程每次从缓存中读取一个完整的报文段。如果进程从缓存中读取报文段的速度不够快，那么缓存将会溢出，并且将丢失报文段。

3.5.6 TCP 连接管理

在本小节中，我们更为仔细地观察如何建立和拆除一条 TCP 连接。尽管这个主题并不特别令人兴奋，但是它很重要，因为 TCP 连接的建立会显著地增加人们感受到的时延（如在 Web 上冲浪时）。此外，许多常见的网络攻击（包括极为流行的 SYN 洪泛攻击）利用了 TCP 连接管理中的弱点。现在我们观察一下一条 TCP 连接是如何建立的。假设运行在一台主机（客户）上的一个进程想与另一台主机（服务器）上的一个进程建立一条连接。客户应用进程首先通知客户 TCP，它想建立一个与服务器上某个进程之间的连接。客户中的 TCP 会用以下方式与服务器中的 TCP 建立一条 TCP 连接：

- 第一步：客户端的 TCP 首先向服务器端的 TCP 发送一个特殊的 TCP 报文段。该报文段中不包含应用层数据。但是在报文段的首部（参见图 3-29）中的一个标志位（即 SYN 比特）被置为 1。因此，这个特殊报文段被称为 SYN 报文段。另外，客户会随机地选择一个初始序号（client_isn），并将此编号放置于该起始的 TCP SYN 报文段的序号字段中。该报文段会被封装在一个 IP 数据报中，并发送给服务器。为了避免某些安全性攻击，在适当地随机化选择 client_isn 方面有着不少有趣的研究［CERT 2001-09］。