看到的是数据报，而不是连接。

TCP 连接提供的是**全双工服务**（full-duplex service）：如果一台主机上的进程 A 与另一台主机上的进程 B 存在一条 TCP 连接，那么应用层数据就可在从进程 B 流向进程 A 的同时，也从进程 A 流向进程 B。TCP 连接也总是**点对点**（point-to-point）的，即在单个发送方与单个接收方之间的连接。所谓“多播”（参见 4.7 节），即在一次发送操作中，从一个发送方将数据传送给多个接收方，对 TCP 来说这是不可能的。对于 TCP 而言，两台主机是一对，而 3 台主机则太多！

我们现在来看看 TCP 连接是怎样建立的。假设运行在某台主机上的一个进程想与另一台主机上的一个进程建立一条连接。前面讲过，发起连接的这个进程被称为*客户进程*，而另一个进程被称为*服务器进程*。该客户应用进程首先要通知客户运输层，它想与服务器上的一个进程建立一条连接。2.7.2 节讲过，一个 Python 客户程序通过发出下面的命令来实现此目的。

```
clientSocket.connect((serverName,serverPort))
```

其中 serverName 是服务器的名字，serverPort 标识了服务器上的进程。客户上的 TCP 便开始与服务器上的 TCP 建立一条 TCP 连接。我们将在本节后面更为详细地讨论连接建立的过程。现在知道下列事实就可以了：客户首先发送一个特殊的 TCP 报文段，服务器用另一个特殊的 TCP 报文段来响应，最后，客户再用第三个特殊报文段作为响应。前两个报文段不承载“有效载荷”，也就是不包含应用层数据；而第三个报文段可以承载有效载荷。由于在这两台主机之间发送了 3 个报文段，所以这种连接建立过程常被称为**三次握手**（three-way handshake）。

一旦建立起一条 TCP 连接，两个应用进程之间就可以相互发送数据了。我们考虑一下从客户进程向服务器进程发送数据的情况。如 2.7 节中所述，客户进程通过套接字（该进程之门）传递数据流。数据一旦通过该门，它就由客户中运行的 TCP 控制了。如图 3-28 所示，TCP 将这些数据引导到该连接的**发送缓存**（send buffer）里，发送缓存是在三次握手初期设置的缓存之一。接下来 TCP 就会不时从发送缓存里取出一块数据。有趣的是，在 TCP 规范［RFC 793］中却没提及 TCP 应何时实际发送缓存里的数据，只是描述为“TCP 应该在它方便的时候以报文段的形式发送数据”。TCP 可从缓存中取出并放入报文段中的数据数量受限于**最大报文段长度**（Maximum Segment Size，MSS）。MSS 通常根据最初确定的由本地发送主机发送的最大链路层帧长度（即所谓的**最大传输单元**（Maximum Transmission Unit，MTU））来设置。设置该 MSS 要保证一个 TCP 报文段（当封装在一个 IP 数据报中）加上 TCP/IP 首部长度（通常 40 字节）将适合单个链路层帧。以太网和 PPP 链路层协议都具有 1500 字节的 MTU，因此 MSS 的典型值为 1460 字节。已经提出了多种发现路径 MTU 的方法，并基于路径 MTU 值设置 MSS（路径 MTU 是指能在从源到目的地的所有链路上发送的最大链路层帧［RFC 1191］）。注意到 MSS 是指在报文段里应用层数据的最大长度，而不是指包括 TCP 首部的 TCP 报文段的最大长度。（该术语很容易混淆，但是我们不得不采用它，因为它已经根深蒂固了。）

TCP 为每块客户数据配上一个 TCP 首部，从而形成多个 **TCP 报文段**（TCP segment）。这些报文段被下传给网络层，网络层将其分别封装在网络层 IP 数据报中。然后这些 IP 数据报被发送到网络中。当 TCP 在另一端接收到一个报文段后，该报文段的数据就被放入该 TCP 连接的接收缓存中，如图 3-28 中所示。应用程序从此缓存中读取数据流。TCP 连接的