第2章 传输层：TCP、UDP和SCTP

2.1 概述

本章提供本书示例所用TCP/IP协议的概貌。我们的目的是从网络编程角度提供足够的细节以理解如何使用这些协议，同时提供有关这些协议的实际设计、实现及历史的具体描述的参考点。

本章的焦点是传输层，包括TCP、UDP和SCTP（Stream Control Transmission Protocol，流控制传输协议）。绝大多数客户/服务器网络应用使用TCP或UDP。SCTP是一个较新的协议，最初设计用于跨因特网传输电话信令。这些传输协议都转而使用网络层协议IP：或是IPv4，或是IPv6。尽管可以绕过传输层直接使用IPv4或IPv6，但这种技术（往往称为*原始套接字*）却极少使用。因此，我们把IPv4和IPv6以及ICMPv4和ICMPv6的详细描述安排在附录A中。

UDP是一个简单的、不可靠的数据报协议，而TCP是一个复杂、可靠的字节流协议。SCTP与TCP类似之处在于它也是一个可靠的传输协议，但它还提供消息边界、传输级别多宿（multihoming）支持以及将头端阻塞（head-of-line blocking）减少到最小的一种方法。我们必须了解由这些传输层协议提供给应用进程的服务，这样才能弄清这些协议处理什么，应用进程中又需要处理什么。

TCP的某些特性一旦理解，就很容易编写健壮的客户和服务器程序，也很容易使用诸如`netstat`等普遍可用的工具来调试客户和服务器程序。本章将阐述以下相关主题：TCP的三路握手、TCP的连接终止序列和TCP的TIME_WAIT状态，SCTP的四路握手和SCTP的连接终止，加上由套接字层提供的TCP、UDP和SCTP缓冲机制，等等。

2.2 总图

虽然协议族被称为“TCP/IP”，但除了TCP和IP这两个主要协议外，还有许多其他成员。图2-1展示了这些协议的概况。

图2-1中同时展示了IPv4和IPv6。从右向左查看该图，最右边的5个网络应用在使用IPv6；我们将在第3章中随`sockaddr_in6`结构讲解`AF_INET6`常值。随后的6个网络应用使用IPv4。

最左边名为`tcpdump`的网络应用或者使用BSD分组过滤器（BSD packet filter，BPF），或者使用数据链路提供者接口（datalink provider interface，DLPI）直接与数据链路进行通信。处于其右边所有9个应用下面的虚线标记为API，它通常是套接字或XTI。访问BPF或DLPI的接口不使用套接字或XTI。

这种情况存在一个例外：Linux使用一种称为`SOCK_PACKET`的特殊套接字类型提供对于数据链路的访问。我们将在第28章中详细讲述这个例外。

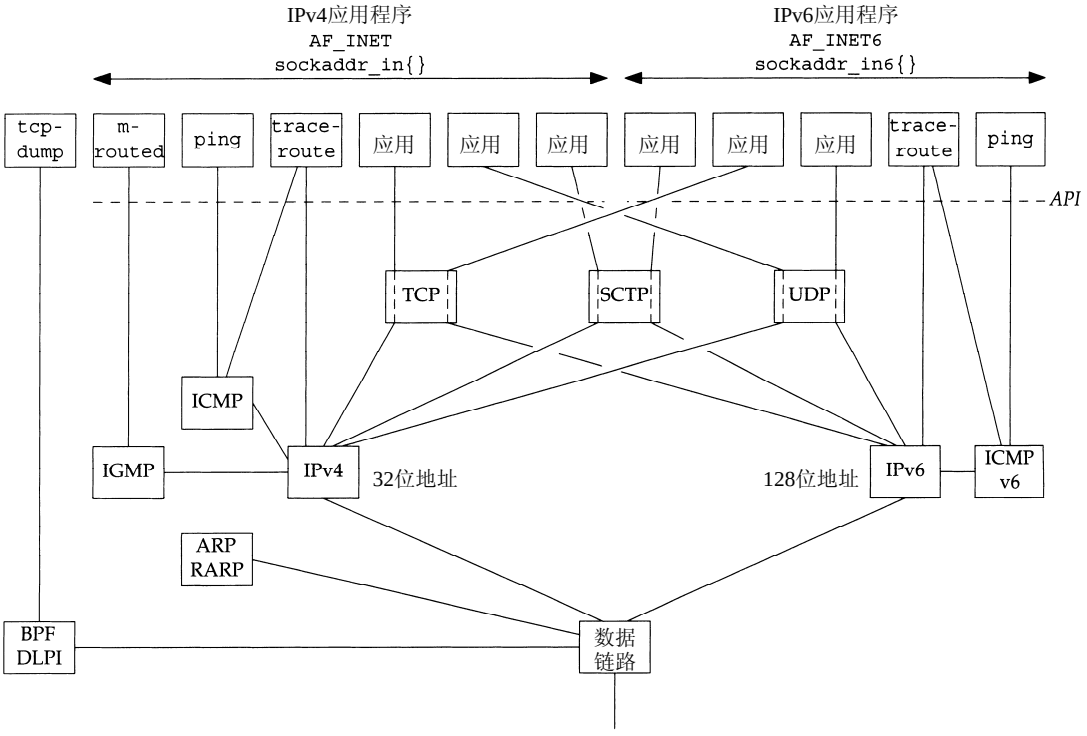

图2-1　TCP/IP协议概况

图2-1中还标明traceroute程序使用两种套接字：IP套接字用于访问IP，ICMP套接字用于访问ICMP。在第28章中，我们将开发ping和traceroute这两个应用的IPv4和IPv6版本。

下面我们讲解一下图2-1中的每一个协议框。

IPv4　网际协议版本4（Internet Protocol version 4）。IPv4（通常称之为IP）自20世纪80年代早期以来一直是网际协议族的主力协议。它使用32位地址（见A.4节）。IPv4给TCP、UDP、SCTP、ICMP和IGMP提供分组递送服务。

IPv6　网际协议版本6（Internet Protocol version 6）。IPv6是在20世纪90年代中期作为IPv4的一个替代品设计的。其主要变化是使用128位更大地址（见A.5节）以应对20世纪90年代因特网的爆发性增长。IPv6给TCP、UDP、SCTP和ICMPv6提供分组递送服务。

当无需区别IPv4和IPv6时，我们经常把“IP”一词作为形容词使用，如IP层、IP地址等。

TCP　传输控制协议（Transmission Control Protocol）。TCP是一个面向连接的协议，为用户进程提供可靠的全双工字节流。TCP套接字是一种流套接字（stream socket）。TCP关心确认、超时和重传之类的细节。大多数因特网应用程序使用TCP。注意，TCP既可以使用IPv4，也可以使用IPv6。

UDP　用户数据报协议（User Datagram Protocol）。UDP是一个无连接协议。UDP套接字是一种数据报套接字（datagram socket）。UDP数据报不能保证最终到达它们的目的地。与TCP一样，UDP既可以使用IPv4，也可以使用IPv6。

SCTP　流控制传输协议（Stream Control Transmission Protocol）。SCTP是一个提供可靠

全双工关联的面向连接的协议，我们使用“关联”一词来指称SCTP中的连接，因为SCTP是多宿的，从而每个关联的两端均涉及一组IP地址和一个端口号。SCTP提供消息服务，也就是维护来自应用层的记录边界。与TCP和UDP一样，SCTP既可以使用IPv4，也可以使用IPv6，而且能够在同一个关联中同时使用它们。

ICMP　网际控制消息协议（Internet Control Message Protocol）。ICMP处理在路由器和主机之间流通的错误和控制消息。这些消息通常由TCP/IP网络支持软件本身（而不是用户进程）产生和处理，不过图中展示的`ping`和`traceroute`程序同样使用ICMP。有时我们称这个协议为ICMPv4，以便与ICMPv6相区别。

IGMP　网际组管理协议（Internet Group Management Protocol）。IGMP用于多播（见第21章），它在IPv4中是可选的。

32 ~ 33

ARP　地址解析协议（Address Resolution Protocol）。ARP把一个IPv4地址映射成一个硬件地址（如以太网地址）。ARP通常用于诸如以太网、令牌环网和FDDI等广播网络，在点到点网络上并不需要。

RARP　反向地址解析协议（Reverse Address Resolution Protocol）。RARP把一个硬件地址映射成一个IPv4地址。它有时用于无盘节点的引导。

ICMPv6　网际控制消息协议版本6（Internet Control Message Protocol version 6）。ICMPv6综合了ICMPv4、IGMP和ARP的功能。

BPF　BSD分组过滤器（BSD packet filter）。该接口提供对于数据链路层的访问能力，通常可以在源自Berkeley的内核中找到。

DLPI　数据链路提供者接口（datalink provider interface）。该接口也提供对于数据链路层的访问能力，通常随SVR4内核提供。

所有网际协议由一个或多个称为请求评注（Request for Comments，RFC）的文档定义，这些RFC就是它们的正式规范。习题2.1的答案说明如何获得这些RFC。

我们使用术语“IPv4/IPv6主机”或“双栈主机”表示同时支持IPv4和IPv6的主机。

TCP/IP协议的其他细节参见TCPv1。TCP/IP在4.4BSD上的实现参见TCPv2。

2.3 用户数据报协议（UDP）

UDP是一个简单的传输层协议，在RFC 768［Postel 1980］中有详细说明。应用进程往一个UDP套接字写入一个消息，该消息随后被封装（encapsulating）到一个UDP数据报，该UDP数据报进而又被封装到一个IP数据报，然后发送到目的地。UDP不保证UDP数据报会到达其最终目的地，不保证各个数据报的先后顺序跨网络后保持不变，也不保证每个数据报只到达一次。

我们使用UDP进行网络编程所遇到的问题是它缺乏可靠性。如果一个数据报到达了其最终目的地，但是校验和检测发现有错误，或者该数据报在网络传输途中被丢弃了，它就无法被投递给UDP套接字，也不会被源端自动重传。如果想要确保一个数据报到达其目的地，可以往应用程序中添置一大堆的特性：来自对端的确认、本端的超时与重传等。

每个UDP数据报都有一个长度。如果一个数据报正确地到达其目的地，那么该数据报的长度将随数据一道传递给接收端应用进程。我们已经提到过TCP是一个字节流（byte-stream）协议，没有任何记录边界（见1.2节），这一点不同于UDP。

我们也说UDP提供无连接的（connectionless）服务，因为UDP客户与服务器之间不必存在任何长期的关系。举例来说，一个UDP客户可以创建一个套接字并发送一个数据报给一个给定的服务器，然后立即用同一个套接字发送另一个数据报给另一个服务器。同样地，一个UDP服
34 务器可以用同一个UDP套接字从若干个不同的客户接收数据报，每个客户一个数据报。

2.4 传输控制协议（TCP）

由TCP向应用进程提供的服务不同于由UDP提供的服务。TCP在RFC 793［Poste1 1981c］中有详细说明，然后由RFC 1323［Jacobson, Braden, and Borman 1992］、RFC 2581［Allman, Paxson, and Stevens 1999］、RFC 2988［Paxson and Allman 2000］和RFC 3390［Allman, Floyd, and Partridge 2002］加以更新。首先，TCP提供客户与服务器之间的连接（connection）。TCP客户先与某个给定服务器建立一个连接，再跨该连接与那个服务器交换数据，然后终止这个连接。

其次，TCP还提供了可靠性（reliability）。当TCP向另一端发送数据时，它要求对端返回一个确认。如果没有收到确认，TCP就自动重传数据并等待更长时间。在数次重传失败后，TCP才放弃，如此在尝试发送数据上所花的总时间一般为4～10分钟（依赖于具体实现）。

> 注意，TCP并不保证数据一定会被对方端点接收，因为这是不可能做到的。如果有可能，TCP就把数据递送到对方端点，否则就（通过放弃重传并中断连接这一手段）通知用户。这么说来，TCP也不能被描述成是100%可靠的协议，它提供的是数据的可靠递送或故障的可靠通知。

TCP含有用于动态估算客户和服务器之间的往返时间（round-trip time，RTT）的算法，以便它知道等待一个确认需要多少时间。举例来说，RTT在一个局域网上大约是几毫秒，跨越一个广域网则可能是数秒钟。另外，因为RTT受网络流通各种变化因素影响，TCP还持续估算一个给定连接的RTT。

TCP通过给其中每个字节关联一个序列号对所发送的数据进行排序（sequencing）。举例来说，假设一个应用写2048字节到一个TCP套接字，导致TCP发送2个分节：第一个分节所含数据的序列号为1～1024，第二个分节所含数据的序列号为1025～2048。（分节是TCP传递给IP的数据单元。）如果这些分节非顺序到达，接收端TCP将先根据它们的序列号重新排序，再把结果数据传递给接收应用。如果接收端TCP接收到来自对端的重复数据（譬如说对端认为一个分节已丢失并因此重传，而这个分节并没有真正丢失，只是网络通信过于拥挤），它可以（根据序列号）判定数据是重复的，从而丢弃重复数据。

> UDP不提供可靠性。UDP本身不提供确认、序列号、RTT估算、超时和重传等机制。如果一个UDP数据报在网络中被复制，两份副本就可能都递送到接收端的主机。同样地，如果一个UDP客户发送两个数据报到同一个目的地，它们可能被网络重新排序，颠倒顺序后到达目的地。UDP应用必须处理所有这些情况，在22.5节中我们将展示如何处理。

再次，TCP提供流量控制（flow control）。TCP总是告知对端在任何时刻它一次能够从对端接收多少字节的数据，这称为通告窗口（advertised window）。在任何时刻，该窗口指出接收缓
35 冲区中当前可用的空间量，从而确保发送端发送的数据不会使接收缓冲区溢出。该窗口时刻动态变化：当接收到来自发送端的数据时，窗口大小就减小，但是当接收端应用从缓冲区中读取数据时，窗口大小就增大。通告窗口大小减小到0是有可能的：当TCP对应某个套接字的接收缓冲区已满，导致它必须等待应用从该缓冲区读取数据时，方能从对端再接收数据。

UDP不提供流量控制。如我们将在8.13节所示，让较快的UDP发送端以一个UDP接收端难以跟上的速率发送数据报是非常容易的。

最后，TCP连接是全双工的（full-duplex）。这意味着在一个给定的连接上应用可以在任何时刻在进出两个方向上既发送数据又接收数据。因此，TCP必须为每个数据流方向跟踪诸如序列号和通告窗口大小等状态信息。建立一个全双工连接后，需要的话可以把它转换成一个单工连接（见6.6节）。

UDP可以是全双工的。

2.5 流控制传输协议（SCTP）

SCTP提供的服务与UDP和TCP提供的类似。SCTP在RFC 2960［Stewart et al. 2000］中详细说明，并由RFC 3309［Stone, Stewart, and Otis 2002］加以更新。RFC 3286［Ong and Yoakum 2002］给出了SCTP的简要介绍。SCTP在客户和服务器之间提供关联（association），并像TCP那样给应用提供可靠性、排序、流量控制以及全双工的数据传送。SCTP中使用“关联”一词取代“连接”是为了避免这样的内涵：一个连接只涉及两个IP地址之间的通信。一个关联指代两个系统之间的一次通信，它可能因为SCTP支持多宿而涉及不止两个地址。

与TCP不同的是，SCTP是面向消息的（message-oriented）。它提供各个记录的按序递送服务。与UDP一样，由发送端写入的每条记录的长度随数据一道传递给接收端应用。

SCTP能够在所连接的端点之间提供多个流，每个流各自可靠地按序递送消息。一个流上某个消息的丢失不会阻塞同一关联其他流上消息的投递。这种做法与TCP正好相反，就TCP而言，在单一字节流中任何位置的字节丢失都将阻塞该连接上其后所有数据的递送，直到该丢失被修复为止。

SCTP还提供多宿特性，使得单个SCTP端点能够支持多个IP地址。该特性可以增强应对网络故障的健壮性。一个端点可能有多个冗余的网络连接，每个网络又可能有各自接入因特网基础设施的连接。当该端点与另一个端点建立一个关联后，如果它的某个网络或某个跨越因特网的通路发生故障，SCTP就可以通过切换到使用已与该关联相关的另一个地址来规避所发生的故障。

类似的健壮性在路由协议的辅助下也可以从TCP中获得。举例来说，由iBGP实现的同一域内的BGP连接往往把赋予路由器内某个虚拟接口的多个地址用作TCP连接的端点。该域的路由协议确保两个路由器之间只要存在一条路由，该路由就会被用上，从而保证这两个路由器之间的BGP连接可用；要是使用属于某个物理接口的地址来建立BGP连接，该物理接口又变得不工作了，这一点就不可能做到。SCTP的多宿特性允许主机（而不仅仅是路由器）也多宿，而且允许多宿跨越不同的服务供应商发生，这些基于路由的TCP多宿方法都无法做到。

36

2.6 TCP 连接的建立和终止

为帮助大家理解`connect`、`accept`和`close`这3个函数并使用`netstat`程序调试TCP应用，我们必须了解TCP连接如何建立和终止，并掌握TCP的状态转换图。

2.6.1 三路握手

建立一个TCP连接时会发生下述情形。

(1) 服务器必须准备好接受外来的连接。这通常通过调用socket、bind和listen这3个函数来完成，我们称之为被动打开（passive open）。

(2) 客户通过调用connect发起主动打开（active open）。这导致客户TCP发送一个SYN（同步）分节，它告诉服务器客户将在（待建立的）连接中发送的数据的初始序列号。通常SYN分节不携带数据，其所在IP数据报只含有一个IP首部、一个TCP首部及可能有的TCP选项（我们稍后讲解）。

(3) 服务器必须确认（ACK）客户的SYN，同时自己也得发送一个SYN分节，它含有服务器将在同一连接中发送的数据的初始序列号。服务器在单个分节中发送SYN和对客户SYN的ACK（确认）。

(4) 客户必须确认服务器的SYN。

这种交换至少需要3个分组，因此称之为TCP的三路握手（three-way handshake）。图2-2展示了所交换的3个分节。

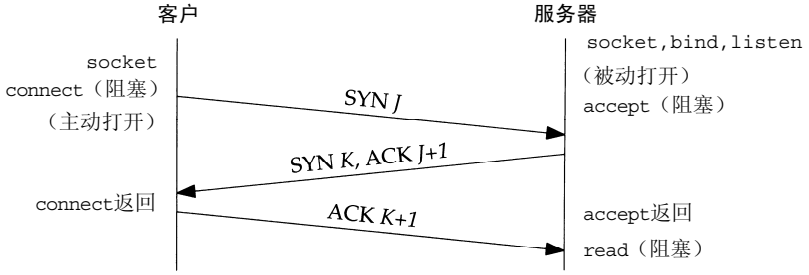

图2-2　TCP的三路握手

37

图2-2给出的客户的初始序列号为J，服务器的初始序列号为K。ACK中的确认号是发送这个ACK的一端所期待的下一个序列号。因为SYN占据一个字节的序列号空间，所以每一个SYN的ACK中的确认号就是该SYN的初始序列号加1。类似地，每一个FIN（表示结束）的ACK中的确认号为该FIN的序列号加1。

> 建立TCP连接就好比一个电话系统[Nemeth 1997]。socket函数等同于有电话可用。bind函数是在告诉别人你的电话号码，这样他们可以呼叫你。listen函数是打开电话振铃，这样当有一个外来呼叫到达时，你就可以听到。connect函数要求我们知道对方的电话号码并拨打它。accept函数发生在被呼叫的人应答电话之时。由accept返回客户的标识（即客户的IP地址和端口号）类似于让电话机的呼叫者ID功能部件显示呼叫者的电话号码。然而两者的不同之处在于accept只在连接建立之后返回客户的标识，而呼叫者ID功能部件却在我们选择应答或不应答电话之前显示呼叫者的电话号码。如果使用域名系统DNS（见第11章），它就提供了一种类似于电话簿的服务。getaddrinfo类似于在电话簿中查找某个人的电话号码，getnameinfo则类似于有一本按照电话号码而不是按照用户名排序的电话簿。

2.6.2　TCP选项

每一个SYN可以含有多个TCP选项。下面是常用的TCP选项。

- MSS选项。发送SYN的TCP一端使用本选项通告对端它的最大分节大小（maximum segment size）即MSS，也就是它在本连接的每个TCP分节中愿意接受的最大数据量。发送端TCP使用接收端的MSS值作为所发送分节的最大大小。我们将在7.9节看到如何使用

TCP_MAXSEG套接字选项提取和设置这个TCP选项。

- 窗口规模选项。TCP连接任何一端能够通告对端的最大窗口大小是65535，因为在TCP首部中相应的字段占16位。然而当今因特网上业已普及的高速网络连接（45 Mbit/s或更快，如RFC 1323［Jacobson, Braden, and Borman 1992］所述）或长延迟路径（卫星链路）要求有更大的窗口以获得尽可能大的吞吐量。这个新选项指定TCP首部中的通告窗口必须扩大（即左移）的位数（0～14），因此所提供的最大窗口接近1 GB（65535×2^{14}）。在一个TCP连接上使用窗口规模的前提是它的两个端系统必须都支持这个选项。我们将在7.5节看到如何使用SO_RCVBUF套接字选项影响这个TCP选项。

  为提供与不支持这个选项的较早实现间的互操作性，需应用如下规则。TCP可以作为主动打开的部分内容随它的SYN发送该选项，但是只在对端也随它的SYN发送该选项的前提下，它才能扩大自己窗口的规模。类似地，服务器的TCP只有接收到随客户的SYN到达的该选项时，才能发送该选项。本逻辑假定实现忽略它们不理解的选项，如此忽略是必需的要求，也已普遍满足，但无法保证所有实现都满足此要求。 38

- 时间戳选项。这个选项对于高速网络连接是必要的，它可以防止由失而复现的分组[①]可能造成的数据损坏。它是一个较新的选项，也以类似于窗口规模选项的方式协商处理。作为网络编程人员，我们无需考虑这个选项。

TCP的大多数实现都支持这些常用选项。后两个选项有时称为“RFC 1323选项”，因为它们是在RFC 1323［Jacobson, Braden, and Borman 1992］中说明的。既然高带宽或长延迟的网络被称为“长胖管道”（long fat pipe），这两个选项也称为“长胖管道选项”。TCPv1的第24章对这些选项有详细的叙述。

2.6.3 TCP连接终止

TCP建立一个连接需3个分节，终止一个连接则需4个分节。

(1) 某个应用进程首先调用close，我们称该端执行*主动关闭*（active close）。该端的TCP于是发送一个FIN分节，表示数据发送完毕。

(2) 接收到这个FIN的对端执行*被动关闭*（passive close）。这个FIN由TCP确认。它的接收也作为一个文件结束符（end-of-file）传递给接收端应用进程（放在已排队等候该应用进程接收的任何其他数据之后），因为FIN的接收意味着接收端应用进程在相应连接上再无额外数据可接收。

(3) 一段时间后，接收到这个文件结束符的应用进程将调用close关闭它的套接字。这导致它的TCP也发送一个FIN。

(4) 接收这个最终FIN的原发送端TCP（即执行主动关闭的那一端）确认这个FIN。

既然每个方向都需要一个FIN和一个ACK，因此通常需要4个分节。我们使用限定词“通常”是因为：某些情形下步骤1的FIN随数据一起发送；另外，步骤2和步骤3发送的分节都出自执行被动关闭那一端，有可能被合并成一个分节。图2-3展示了这些分组。

① “失而复现的分组”这个译法出自第2版，这一版中改为“陈旧的、延迟的或重复的分节”，却没能准确表达Stevens先生的原意。失而复现的分组并不是超时重传的分组，而是由暂时的路由原因造成的迷途的分组。当路由稳定后，它们又会正常到达目的地，其前提是它们在此前尚未被路由器丢弃。高速网络中32位的序列号短时间内就可能循环一轮重新使用，若不用时间戳选项，失而复现的分组所承载的分节可能与再次使用相同序列号的真正分节发生混淆。——译者注

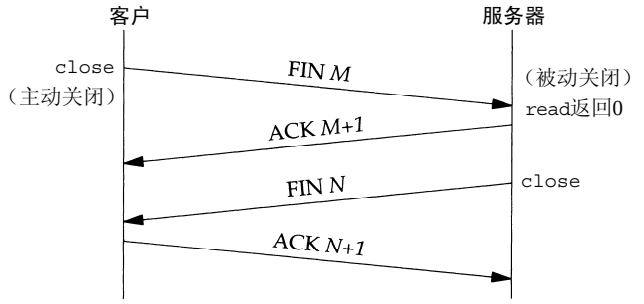

图2-3　TCP连接关闭时的分组交换

类似SYN，一个FIN也占据1个字节的序列号空间。因此，每个FIN的ACK确认号就是这个FIN的序列号加1。

在步骤2与步骤3之间，从执行被动关闭一端到执行主动关闭一端流动数据是可能的。这称为半关闭（half-close），我们将在6.6节随shutdown函数再详细介绍。

当套接字被关闭时，其所在端TCP各自发送了一个FIN。我们在图中指出，这是由应用进程调用close而发生的，不过需认识到，当一个Unix进程无论自愿地（调用exit或从main函数返回）还是非自愿地（收到一个终止本进程的信号）终止时，所有打开的描述符都被关闭，这也导致仍然打开的任何TCP连接上也发出一个FIN。

图2-3展示了客户执行主动关闭的情形，不过我们指出，无论是客户还是服务器，任何一端都可以执行主动关闭。通常情况是客户执行主动关闭，但是某些协议（譬如值得注意的HTTP/1.0）却由服务器执行主动关闭。

2.6.4　TCP 状态转换图

TCP涉及连接建立和连接终止的操作可以用状态转换图（state transition diagram）来说明，如图2-4所示。

TCP为一个连接定义了11种状态，并且TCP规则规定如何基于当前状态及在该状态下所接收的分节从一个状态转换到另一个状态。举例来说，当某个应用进程在CLOSED状态下执行主动打开时，TCP将发送一个SYN，且新的状态是SYN_SENT。如果这个TCP接着接收到一个带ACK的SYN，它将发送一个ACK，且新的状态是ESTABLISHED。这个最终状态是绝大多数数据传送发生的状态。

自ESTABLISHED状态引出的两个箭头处理连接的终止。如果某个应用进程在接收到一个FIN之前调用close（主动关闭），那就转换到FIN_WAIT_1状态。但如果某个应用进程在ESTABLISHED状态期间接收到一个FIN（被动关闭），那就转换到CLOSE_WAIT状态。

我们用粗实线表示通常的客户状态转换，用粗虚线表示通常的服务器状态转换。图中还注明存在两个我们未曾讨论的转换：一个为同时打开（simultaneous open），发生在两端几乎同时发送SYN并且这两个SYN在网络中交错的情形下，另一个为同时关闭（simultaneous close），发生在两端几乎同时发送FIN的情形下。TCPv1的第18章中有这两种情况的例子和讨论，它们是可能发生的，不过非常罕见。

展示状态转换图的原因之一是给出11种TCP状态的名称。这些状态可使用netstat显示，它是一个在调试客户/服务器应用时很有用的工具。我们将在第5章中使用netstat去监视状态的变化。

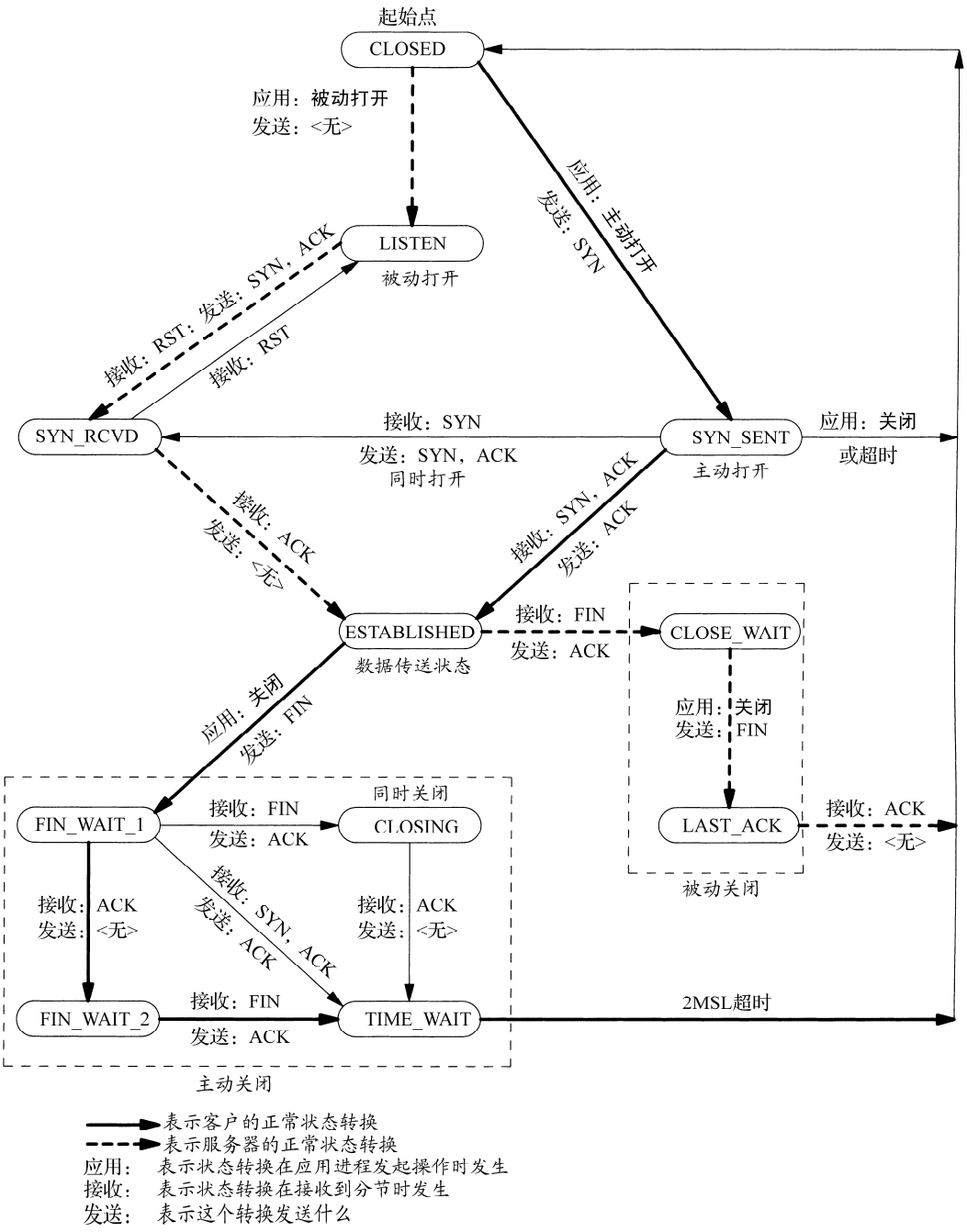

图2-4 TCP状态转换图

41

2.6.5 观察分组

图2-5展示一个完整的TCP连接所发生的实际分组交换情况，包括连接建立、数据传送和连接终止3个阶段。图中还展示了每个端点所历经的TCP状态。

本例中的客户通告一个值为536的MSS（表明该客户只实现了最小重组缓冲区大小），服务器通告一个值为1460的MSS（以太网上IPv4的典型值）。不同方向上MSS值不相同不成问题（见习题2.5）。

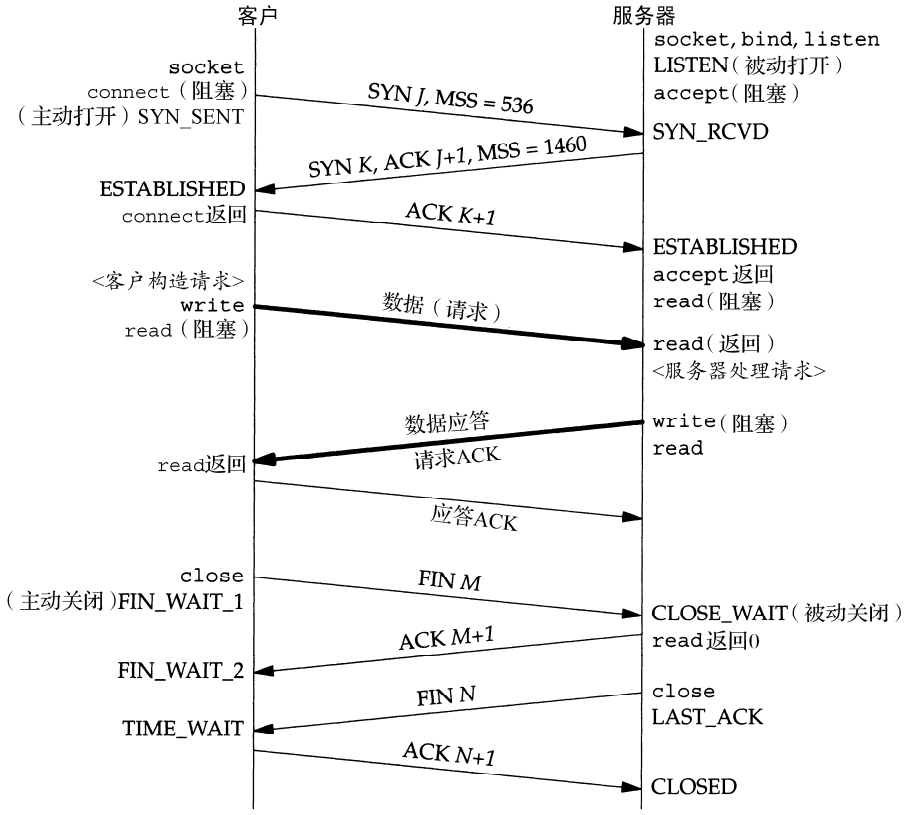

图2-5　TCP连接的分组交换

一旦建立一个连接，客户就构造一个请求并发送给服务器。这里我们假设该请求适合于单个TCP分节（即请求大小小于服务器通告的值为1460字节的MSS）。服务器处理该请求并发送一个应答，我们假设该应答也适合于单个分节（本例即小于536字节）。图中使用粗箭头表示这两个数据分节。注意，服务器对客户请求的确认是伴随其应答发送的。这种做法称为捎带（piggybacking），它通常在服务器处理请求并产生应答的时间少于200 ms时发生。如果服务器耗用更长时间，譬如说1 s，那么我们将看到先是确认后是应答。（TCP数据流机理在TCPv1的第19章和第20章中详细叙述。）

42

图中随后展示的是终止连接的4个分节。注意，执行主动关闭的那一端（本例子中为客户）进入我们将在下一节中讨论的TIME_WAIT状态。

图2-5中值得注意的是，如果该连接的整个目的仅仅是发送一个单分节的请求和接收一个单分节的应答，那么使用TCP有8个分节的开销。如果改用UDP，那么只需交换两个分组：一个承载请求，一个承载应答。然而从TCP切换到UDP将丧失TCP提供给应用进程的全部可靠性，迫使可靠服务的一大堆细节从传输层（TCP）转移到UDP应用进程。TCP提供的另一个重要特性即拥塞控制也必须由UDP应用进程来处理。尽管如此，我们仍然需要知道许多网络应用是使用UDP构建的，因为它们需要交换的数据量较少，而UDP避免了TCP连接建立和终止所需的开销。

2.7 TIME_WAIT状态

毫无疑问，TCP中有关网络编程最不容易理解的是它的TIME_WAIT状态。在图2-4中我们看到执行主动关闭的那端经历了这个状态。该端点停留在这个状态的持续时间是最长分节生命期（maximum segment lifetime，MSL）的两倍，有时候称之为2MSL。

任何TCP实现都必须为MSL选择一个值。RFC 1122［Braden 1989］的建议值是2分钟，不过源自Berkeley的实现传统上改用30秒这个值。这意味着TIME_WAIT状态的持续时间在1分钟到4分钟之间。MSL是任何IP数据报能够在因特网中存活的最长时间。我们知道这个时间是有限的，因为每个数据报含有一个称为跳限（hop limit）的8位字段（见图A-1中IPv4的TTL字段和图A-2中IPv6的跳限字段），它的最大值为255。尽管这是一个跳数限制而不是真正的时间限制，我们仍然假设：具有最大跳限（255）的分组在网络中存在的时间不可能超过MSL秒。

分组在网络中“迷途”通常是路由异常的结果。某个路由器崩溃或某两个路由器之间的某个链路断开时，路由协议需花数秒钟到数分钟的时间才能稳定并找出另一条通路。在这段时间内有可能发生路由循环（路由器A把分组发送给路由器B，而B再把它们发送回A），我们关心的分组可能就此陷入这样的循环。假设迷途的分组是一个TCP分节，在它迷途期间，发送端TCP超时并重传该分组，而重传的分组却通过某条候选路径到达最终目的地。然而不久后（自迷途的分组开始其旅程起最多MSL秒以内）路由循环修复，早先迷失在这个循环中的分组最终也被送到目的地。这个原来的分组称为*迷途的重复分组*（lost duplicate）或*漫游的重复分组*（wandering duplicate）。TCP必须正确处理这些重复的分组。

43

TIME_WAIT状态有两个存在的理由：

(1) 可靠地实现TCP全双工连接的终止；

(2) 允许老的重复分节在网络中消逝。

第一个理由可以通过查看图2-5并假设最终的ACK丢失了来解释。服务器将重新发送它的最终那个FIN，因此客户必须维护状态信息，以允许它重新发送最终那个ACK。要是客户不维护状态信息，它将响应以一个RST（另外一种类型的TCP分节），该分节将被服务器解释成一个错误。如果TCP打算执行所有必要的工作以彻底终止某个连接上两个方向的数据流（即全双工关闭），那么它必须正确处理连接终止序列4个分节中任何一个分节丢失的情况。本例子也说明了为什么执行主动关闭的那一端是处于TIME_WAIT状态的那一端：因为可能不得不重传最终那个ACK的就是那一端。

为理解存在TIME_WAIT状态的第二个理由，我们假设在12.106.32.254的1500端口和206.168.112.219的21端口之间有一个TCP连接。我们关闭这个连接，过一段时间后在相同的IP地址和端口之间建立另一个连接。后一个连接称为前一个连接的*化身*（incarnation），因为它们的IP地址和端口号都相同。TCP必须防止来自某个连接的老的重复分组在该连接已终止后再现，从而被误解成属于同一连接的某个新的化身。为做到这一点，TCP将不给处于TIME_WAIT状态的连接发起新的化身。既然TIME_WAIT状态的持续时间是MSL的2倍，这就足以让某个方向上的分组最多存活MSL秒即被丢弃，另一个方向上的应答最多存活MSL秒也被丢弃。通过实施这个规则，我们就能保证每成功建立一个TCP连接时，来自该连接先前化身的老的重复分组都已在网络中消逝了。

这个规则存在一个例外：如果到达的SYN的序列号大于前一化身的结束序列号，源自Berkeley的实现将给当前处于TIME_WAIT状态的连接启动新的化身。TCPv2第958～959页对

这种情况有详细的叙述。它要求服务器执行主动关闭，因为接收下一个SYN的那一端必须处于TIME_WAIT状态。rsh命令具备这种能力。RFC 1185 [Jacobson, Braden, and Zhang 1990] 讲述了有关这种情形的一些陷阱。

2.8 SCTP 关联的建立和终止

与TCP一样，SCTP也是面向连接的，因而也有关联的建立与终止的握手过程。不过SCTP
44 的握手过程不同于TCP，我们在此加以说明。

2.8.1 四路握手

建立一个SCTP关联的时候会发生下述情形（类似于TCP）。

(1) 服务器必须准备好接受外来的关联。这通常通过调用socket、bind和listen这3个函数来完成，称为被动打开。

(2) 客户通过调用connect或者发送一个隐式打开该关联的消息进行主动打开。这使得客户SCTP发送一个INIT消息（初始化），该消息告诉服务器客户的IP地址清单、初始序列号、用于标识本关联中所有分组的起始标记、客户请求的外出流的数目以及客户能够支持的外来流的数目。

(3) 服务器以一个INIT ACK消息确认客户的INIT消息，其中含有服务器的IP地址清单、初始序列号、起始标记、服务器请求的外出流的数目、服务器能够支持的外来流的数目以及一个状态cookie。状态cookie包含服务器用于确信本关联有效所需的所有状态，它是数字化签名过的，以确保其有效性。

(4) 客户以一个COOKIE ECHO消息回射服务器的状态cookie。除COOKIE ECHO外，该消息可能在同一个分组中还捆绑了用户数据。

(5) 服务器以一个COOKIE ACK消息确认客户回射的cookie是正确的，本关联于是建立。该消息也可能在同一个分组中还捆绑了用户数据。

以上交换过程至少需要4个分组，因此称之为SCTP的四路握手（four-way handshake）。图2-6展示了这4个分节。

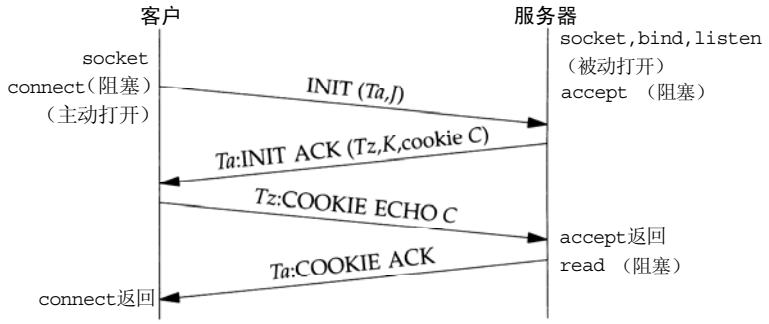

图2-6 SCTP的四路握手

45 SCTP的四路握手在很多方面类似于TCP的三路握手，差别主要在于作为SCTP整体一部分的cookie的生成。INIT（随其众多参数一道）承载一个验证标记Ta和一个初始序列号J。在关联的有效期内，验证标记Ta必须在对端发送的每个分组中出现。初始序列号J用作承载用户数据的

DATA块的起始序列号。对端也在INIT ACK中承载一个验证标记Tz，在关联的有效期内，验证标记Tz也必须在其发送的每个分组中出现。除了验证标记Tz和初始序列号K外，INIT的接收端还在作为响应的INIT ACK中提供一个cookie C。该cookie包含设置本SCTP关联所需的所有状态，这样服务器的SCTP栈就不必保存所关联客户的有关信息。SCTP关联设置的细节参见［Stewart and Xie 2001］的第4章。

四路握手过程结束时，两端各自选择一个主目的地址（primary destination address）。当不存在网络故障时，主目的地址将用作数据要发送到的默认目的地。

在SCTP中使用四路握手是为了避免一种将在4.5节讨论的拒绝服务攻击。

> SCTP使用cookie的四路握手定形了一种防护这种攻击的方法。TCP的许多实现也使用类似的方法。两者的主要差别在于，TCP中cookie状态必须编码到只有32位长的初始序列号中。SCTP为此提供了一个任意长度的字段，并且要求实施基于加密的安全性以防护攻击。

2.8.2 关联终止

SCTP不像TCP那样允许“半关闭”的关联。当一端关闭某个关联时，另一端必须停止发送新的数据。关联关闭请求的接收端发送完已经排队的数据（如果有的话）后，完成关联的关闭。图2-7展示了这一交换过程。

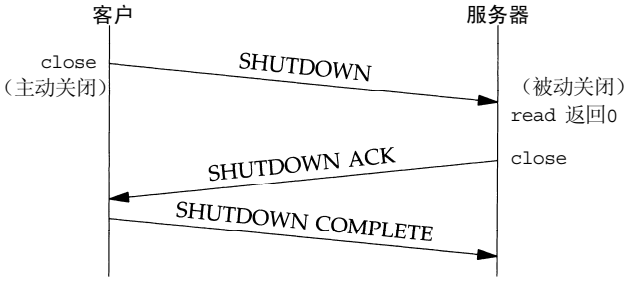

图2-7 SCTP关联关闭时的分组交换

SCTP没有类似于TCP的TIME_WAIT状态，因为SCTP使用了验证标记。所有后续块都在捆绑它们的SCTP分组的公共首部标记了初始的INIT块和INIT ACK块中作为起始标记交换的验证标记；由来自旧连接的块通过所在SCTP分组的公共首部间接携带的验证标记对于新连接来说是不正确的。因此，SCTP通过放置验证标记值就避免了TCP在TIME_WAIT状态保持整个连接的做法。

2.8.3 SCTP状态转换图

SCTP涉及关联建立和关联终止的操作可以用状态转换图（state transition diagram）来说明，如图2-8所示。

与图2-4一样，本状态机中从一个状态到另一个状态的转换由SCTP规则基于当前状态及在该状态下所接收的块规定。举例来说，当某个应用进程在CLOSED状态下执行主动打开时，SCTP将发送一个INIT，且新的状态是COOKIE-WAIT。如果这个SCTP接着接收到一个INIT ACK，它将发送一个COOKIE ECHO，且新的状态是COOKIE-ECHOED。如果该SCTP随后接收到一个COOKIE ACK，它将转换成ESTABLISHED状态。这个最终状态是绝大多数数据传送发生点的状态，尽管DATA块也可以由COOKIE ECHO块或COOKIE ACK块所在消息捆绑捎带。

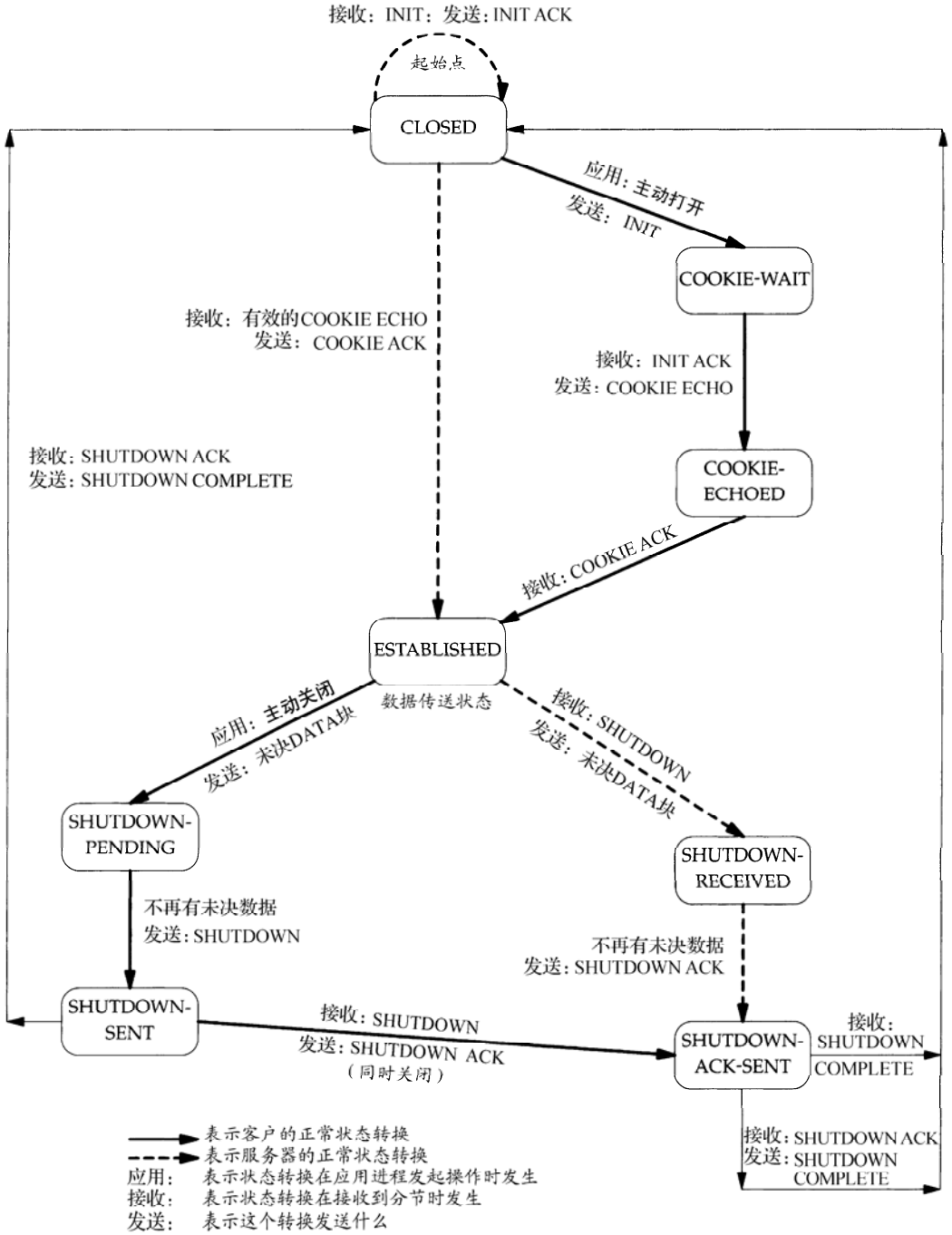

图2-8　SCTP状态转换图

从ESTABLISHED状态引出的两个箭头处理关联的终止。如果某个应用进程在接收到一个SHUTDOWN之前调用`close`（主动关闭），那就转换到SHUTDOWN-PENDING状态。否则，如果某个应用进程在ESTABLISHED状态期间接收到一个SHUTDOWN（被动关闭），那就转换到SHUTDOWN-RECEIVED状态。

47
~
48

2.8.4 观察分组

图2-9展示一个作为样例的SCTP关联所发生的实际分组交换情况，包括关联建立、数据传送和关联终止3个阶段。图中还展示了每个端点所历经的SCTP状态。

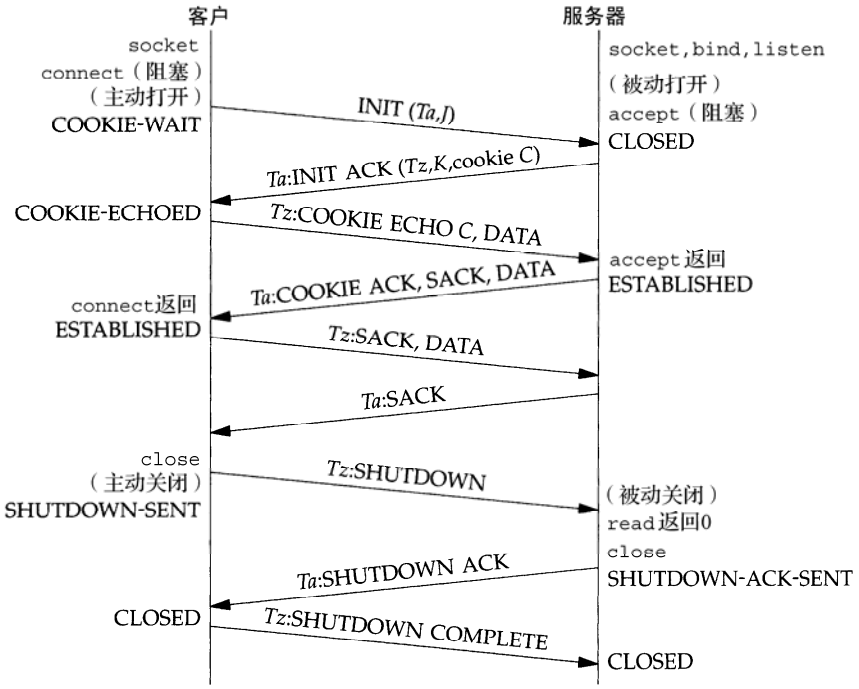

图2-9 SCTP关联中的分组交换

本例中，客户在COOKIE ECHO块所在分组中捎带了它的第一个DATA块，服务器则在作为应答的COOKIE ACK块所在分组中捎带了数据。一般而言，当网络应用采用一到多接口式样时（我们将在9.2节中讨论一到一和一到多这两种接口式样），COOKIE ECHO通常捎带一个或多个DATA块。

SCTP分组中信息的单位称为块（chunk）。块是自描述的，包含一个块类型、若干个块标记和一个块长度。这样做方便了多个块的绑缚，只要把它们简单地组合到一个SCTP外出消息中（[Stewart and Xie 2001]的第5章给出了块捆绑和常规数据传输过程的细节）。

49

2.8.5 SCTP 选项

SCTP使用参数和块方便增设可选特性。新的特性通过添加这两个条目之一加以定义，并允许通常的SCTP处理规则汇报未知的参数和未知的块。参数类型字段和块类型字段的高两位指明SCTP接收端该如何处置未知的参数或未知的块（[Stewart and Xie 2001]的3.1节给出了更多的细节）。

当前如下两个对SCTP的扩展正在开发中。

(1) 动态地址扩展，允许协作的SCTP端点从已有的某个关联中动态增删IP地址。

(2) 不完全可靠性扩展，允许协作的SCTP端点在应用进程的指导下限制数据的重传。当一个消息变得过于陈旧而无须发送时（按照应用进程的指导），该消息将被跳过而不再发送到对端。

这意味着不是所有数据都确保到达关联的另一端。

2.9 端口号

任何时候，多个进程可能同时使用TCP、UDP和SCTP这3种传输层协议中的任何一种。这3种协议都使用16位整数的*端口号*（port number）来区分这些进程。

当一个客户想要跟一个服务器联系时，它必须标识想要与之通信的这个服务器。TCP、UDP和SCTP定义了一组*众所周知的端口*（well-known port），用于标识众所周知的服务。举例来说，支持FTP的任何TCP/IP实现都把21这个众所周知的端口分配给FTP服务器。分配给*简化文件传送协议*（Trivial File Trqnsfer Protocol，TFTP）的是UDP端口号69。

另一方面，客户通常使用短期存活的*临时端口*（ephemeral port）。这些端口号通常由传输层协议自动赋予客户。客户通常不关心其临时端口的具体值，而只需确信该端口在所在主机中是唯一的就行。传输协议的代码确保这种唯一性。

IANA（the Internet Assigned Numbers Authority，因特网已分配数值权威机构）维护着一个端口号分配状况的清单。该清单一度作为RFC多次发布；RFC 1700［Reynolds and Postel 1994］是这个系列的最后一个。RFC 3232［Reynolds 2002］给出了替代RFC 1700的在线数据库的位置：http://www.iana.org/。端口号被划分成以下3段。

(1) *众所周知的端口为*0～1023。这些端口由IANA分配和控制。可能的话，相同端口号就分配给TCP、UDP和SCTP的同一给定服务。例如，不论TCP还是UDP端口号80都被赋予Web服务器，尽管它目前的所有实现都单纯使用TCP。

50

> *端口号*80分配时SCTP尚不存在。新的端口分配将针对这3种协议执行，RFC 2960则声明所有现有的TCP端口号对于使用SCTP的同一服务同样有效。

(2) *已登记的端口*（registered port）为1024～49151。这些端口不受IANA控制，不过由IANA登记并提供它们的使用情况清单，以方便整个群体。可能的话，相同端口号也分配给TCP和UDP的同一给定服务。例如，6000～6063分配给这两种协议的X Window服务器，尽管它的所有实现当前单纯使用TCP。49151这个上限的引入是为了给临时端口留出范围，而RFC 1700［Reynolds and Postel 1994］所列的上限为65535。

(3) 49152～65535是*动态的*（dynamic）或*私用的*（private）端口。IANA不管这些端口。它们就是我们所称的*临时端口*。（49152这个魔数是65536的四分之三。）

图2-10展示了端口号的划分情况和常见的分配情况。

我们要注意图2-10中以下几点。

- Unix系统有*保留端口*（reserved port）的概念，指的是小于1024的任何端口。这些端口只能赋予特权用户进程的套接字。所有IANA众所周知的端口都是保留端口，分配使用这些端口的服务器（例如FTP服务器）必须以超级用户特权启动。
- 由于历史原因，源自Berkeley的实现（从4.3BSD开始）曾在1024～5000范围内分配临时端口。这在20世纪80年代初期是可行的，但是如今很容易就找到一个在任何给定时间内同时支持多于3977个连接的主机。于是许多较新的系统从另外的范围分配临时端口以提供更多的临时端口，它们或者使用由IANA定义的临时端口范围，或者使用一个更大的其他范围（如图2-10所示的Solaris）。

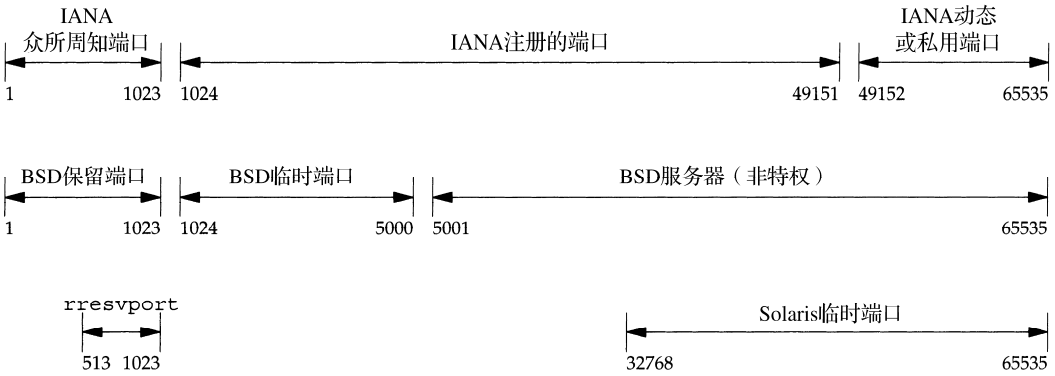

图2-10 端口号的分配

由于这个原因，许多较早的系统实现的临时端口范围的上限为5 000。5 000这个上限后来发现是一个排版错误［Borman 1997a］，本应该是50 000。

51

- 有少数客户（而不是服务器）需要一个保留端口用于客户/服务器的认证：`rlogin`和`rsh`客户就是常见的例子。这些客户调用库函数`rresvport`创建一个TCP套接字，并赋予它一个在513～1023范围内未使用的端口。该函数通常先尝试绑定端口1023，若失败则尝试1022，依次类推，直到在端口513上亦或成功，亦或失败。

注意：BSD的保留端口和`rresvport`函数都跟IANA众所周知端口的后半部分重叠。这是因为IANA众所周知端口早先的上限为255。1992年的RFC 1340（早先的一个“Assigned Numbers”RFC）开始在256～1023之间分配众所周知的端口。1990年的RFC 1060（更早先的一个“Assigned Numbers”RFC）称256～1023之间的端口为Unix标准服务（Unix Standard Services）。20世纪80年代有不少源自Berkeley的服务器在512以后挑选它们的众所周知的端口（留下256～511这个空档）。`rresvport`函数选择从1023开始往下寻找，直至513。

套接字对

一个TCP连接的套接字对（socket pair）是一个定义该连接的两个端点的四元组：本地IP地址、本地TCP端口号、外地IP地址、外地TCP端口号。套接字对唯一标识一个网络上的每个TCP连接。就SCTP而言，一个关联由一组本地IP地址、一个本地端口、一组外地IP地址、一个外地端口标识。在两个端点均非多宿这一最简单的情形下，SCTP与TCP所用的四元组套接字对一致。然而在某个关联的任何一个端点为多宿的情形下，同一个关联可能需要多个四元组标识（这些四元组的IP地址各不相同，但端口号是一样的）。

标识每个端点的两个值（IP地址和端口号）通常称为一个套接字。

我们可以把套接字对的概念扩展到UDP，即使UDP是无连接的。当讲解套接字函数（`bind`、`connect`、`getpeername`等）时，我们将指明它们在指定套接字对中的哪些值。举例来说，`bind`函数要求应用程序给TCP、UDP或SCTP套接字指定本地IP地址和本地端口号。

2.10 TCP端口号与并发服务器

并发服务器中主服务器循环通过派生一个子进程来处理每个新的连接。如果一个子进程继

续使用服务器众所周知的端口来服务一个长时间的请求，那将发生什么？让我们来看一个典型的序列。首先，在主机freebsd上启动服务器，该主机是多宿的，其IP地址为12.106.32.254和192.168.42.1。服务器在它的众所周知的端口（本例为21）上执行被动打开，从而开始等待客户
52 的请求，如图2-11所示。

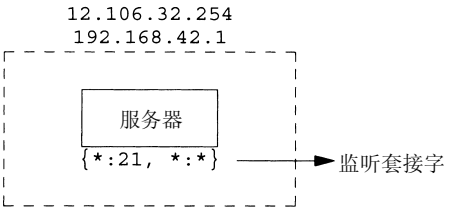

图2-11 TCP服务器在端口21上执行被动打开

我们使用记号{*:21, *:*}指出服务器的套接字对。服务器在任意本地接口（第一个星号）的端口21上等待连接请求。外地IP地址和外地端口都没有指定，我们用“*.*”来表示。我们称它为监听套接字（listening socket）。

> 我们用分号来分割IP地址和端口号，因为这是HTTP的用法，其他地方也常见。netstat程序使用点号来分割IP地址和端口号，不过如此表示有时候会让人混淆，因为点号既用于域名（如freebsd.unpbook.com.21），也用于IPv4的点分十进制数记法（如12.106.32.254.21）。

这里指定本地IP地址的星号称为通配（wildcard）符。如果运行服务器的主机是多宿的（如本例），服务器可以指定它只接受到达某个特定本地接口的外来连接。这里要么选一个接口要么选任意接口。服务器不能指定一个包含多个地址的清单。通配的本地地址表示“任意”这个选择。在图1-9中，通配地址通过在调用bind之前把套接字地址结构中的IP地址字段设置成INADDR_ANY来指定。

稍后在IP地址为206.168.112.219的主机上启动第一个客户，它对服务器的IP地址之一12.106.32.254执行主动打开。我们假设本例中客户主机的TCP为此选择的临时端口为1500，如图2-12所示。图中在该客户的下方标出了它的套接字对。

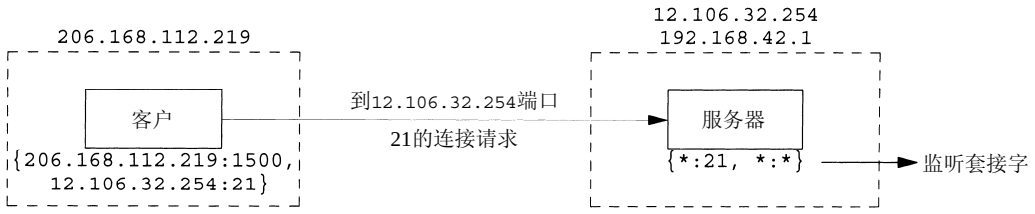

图2-12 客户对服务器的连接请求

当服务器接收并接受这个客户的连接时，它fork一个自身的副本，让子进程来处理该客户
53 的请求，如图2-13所示。（我们将在4.7节中讲解fork函数。）

至此，我们必须在服务器主机上区分监听套接字和已连接套接字（connected socket）。注意已连接套接字使用与监听套接字相同的本地端口（21）。还要注意在多宿服务器主机上，连接一旦建立，已连接套接字的本地地址（12.106.32.254）随即填入。

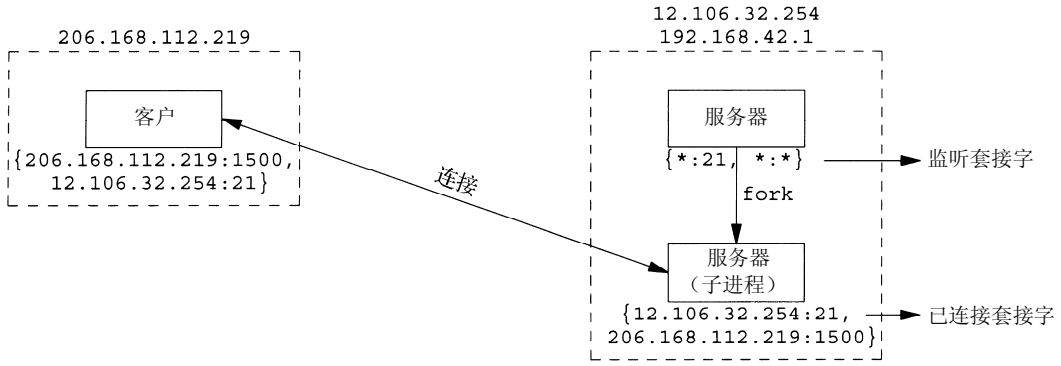

图2-13 并发服务器让子进程处理客户

下一步我们假设在客户主机上另有一个客户请求连接到同一个服务器。客户主机的TCP为这个新客户的套接字分配一个未使用的临时端口，譬如说1501，如图2-14所示。服务器上这两个连接是有区别的：第一个连接的套接字对和第二个连接的套接字对不一样，因为客户的TCP给第二个连接选择了一个未使用的端口（1501）。

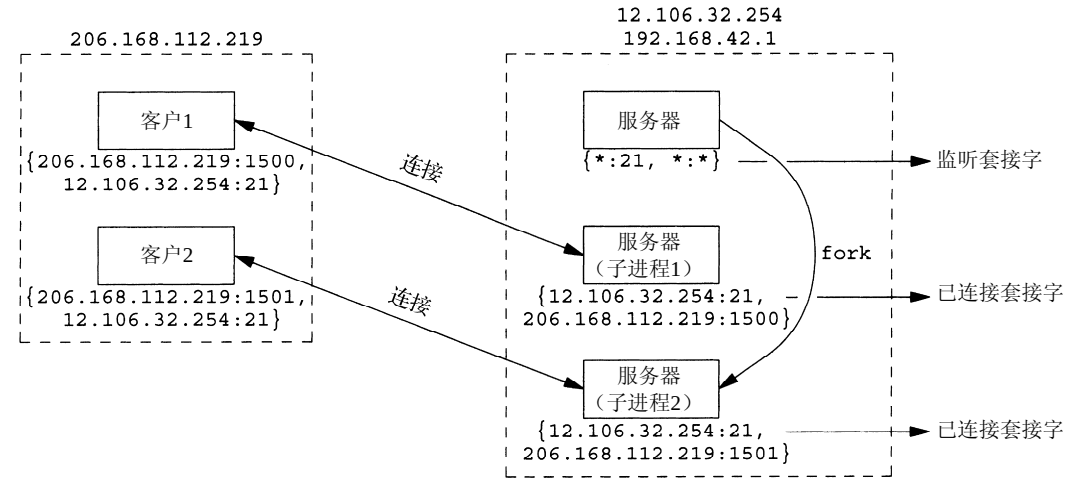

图2-14 第二个客户与同一个服务器的连接

通过本例应注意，TCP无法仅仅通过查看目的端口号来分离外来的分节到不同的端点。它必须查看套接字对的所有4个元素才能确定由哪个端点接收某个到达的分节。图2-14中对于同一个本地端口（21）存在3个套接字。如果一个分节来自206.168.112.219端口1500，目的地为12.106.32.254端口21，它就被递送给第一个子进程。如果一个分节来自206.168.112.219端口1501，目的地为12.106.32.254端口21，它就被递送给第二个子进程。所有目的端口为21的其他TCP分节都被递送给拥有监听套接字的最初那个服务器（父进程）。

2.11 缓冲区大小及限制

下面我们将介绍一些影响IP数据报大小的限制。我们首先介绍这些限制，然后就它们如何

影响应用进程能够传送的数据进行综合分析。

- IPv4数据报的最大大小是65 535字节，包括IPv4首部。这是因为如图A-1所示其总长度字段占据16位。
- IPv6数据报的最大大小是65 575字节，包括40字节的IPv6首部。这是因为如图A-2所示其净荷长度字段占据16位。注意，IPv6的净荷长度字段不包括IPv6首部，而IPv4的总长度字段包括IPv4首部。

  IPv6有一个特大净荷（jumbo payload）选项，它把净荷长度字段扩展到32位，不过这个选项需要MTU（maximum transmission unit，最大传输单元）超过65 535的数据链路提供支持。（这是为主机到主机的内部连接而设计的，譬如HIPPI，它们通常没有内在的MTU。）
- 许多网络有一个可由硬件规定的MTU。举例来说，以太网的MTU是1500字节。另有一些链路（例如使用PPP协议的点到点链路）其MTU可以人为配置。较老的SLIP链路通常使用1006字节或296字节的MTU。

  IPv4要求的最小链路MTU是68字节。这允许最大的IPv4首部（包括20字节的固定长度部分和最多40字节的选项部分）拼接最小的片段（IPv4首部中片段偏移字段以8个字节为单位）。IPv6要求的最小链路MTU为1280字节。IPv6可以运行在MTU小于此最小值的链路上，不过需要特定于链路的分片和重组功能，以使得这些链路看起来具有至少为1280字节的MTU（RFC 2460［Deering and Hinden 1998］）。

54 ~ 55

- 在两个主机之间的路径中最小的MTU称为路径MTU（path MTU）。1500字节的以太网MTU是当今常见的路径MTU。两个主机之间相反的两个方向上路径MTU可以不一致，因为在因特网中路由选择往往是不对称的［Paxson 1196］，也就是说从A到B的路径与从B到A的路径可以不相同。
- 当一个IP数据报将从某个接口送出时，如果它的大小超过相应链路的MTU，IPv4和IPv6都将执行分片（fragmentation）。这些片段在到达最终目的地之前通常不会被重组（reassembling）。IPv4主机对其产生的数据报执行分片，IPv4路由器则对其转发的数据报执行分片。然而IPv6只有主机对其产生的数据报执行分片，IPv6路由器不对其转发的数据报执行分片。

  我们必须小心这些术语的使用。一个标记为IPv6路由器的设备可能执行分片，不过只是对于那些由它产生的数据报，而绝不是对于那些由它转发的数据报。当该设备产生IPv6数据报时，它实际上作为主机运作。举例来说，大多数路由器支持Telnet协议，管理员就用它来配置路由器。由路由器的Telnet服务器产生的IP数据报是由路由器产生的，而不是由路由器转发的。

  你可能注意到，IPv4首部（图A-1）有用于处理IPv4分片的字段，IPv6首部（图A-2）却没有类似的字段。既然分片是例外情况而不是通常情况，IPv6于是引入一个可选首部以提供分片信息。

  某些通常用作路由器的防火墙可能会重组分片了的分组，以便查看整个IP数据报的内容。这样做使得不必在防火墙上引入额外的复杂性就能够防止某些攻击。它还要求防火墙设备是进出网络的唯一路径上的设备，从而减少了冗余的机会。

- IPv4首部（图A-1）的“不分片（don’t fragment）”位（即DF位）若被设置，那么不管是发送这些数据报的主机还是转发它们的路由器，都不允许对它们分片。当路由器接收到一个超过其外出链路MTU大小且设置了DF位的IPv4数据报时，它将产生一个ICMPv4“destination unreachable, fragmentation needed but DF bit set”（目的地不可达，需分片但DF位已设置）出错消息（图A-15）。

既然IPv6路由器不执行分片，每个IPv6数据报于是隐含一个DF位。当IPv6路由器接收到一个超过其外出链路MTU大小的IPv6数据报时，它将产生一个ICMPv6“packet too big”（分组太大）出错消息（图A-16）。

IPv4的DF位和IPv6的隐含DF位可用于路径MTU发现（IPv4的情形见RFC 1191［Mogul and Deering 1990］，IPv6的情形见RFC 1981［McCann, Deering, and Mogul 1996］）。举例来说，如果基于IPv4的TCP使用该技术，那么它将在所发送的所有数据报中设置DF位。如果某个中间路由器返回一个ICMP“destination unreachable, fragmentation needed but DF bit set”错误，TCP就减小每个数据报的数据量并重传。路径MTU发现对于IPv4是可选的，然而IPv6的所有实现要么必须支持它，要么必须总是使用最小的MTU发送IPv6数据报。 56

> 路径MTU发现在如今的因特网上是有问题的，许多防火墙丢弃所有ICMP消息，包括用于路径MTU发现的上述消息。这意味着TCP永远得不到要求它降低所发送数据量的信号。编写本书时，IETF已经开始尝试定义不依赖于ICMP出错消息的另一种路径MTU发现方法。

- IPv4和IPv6都定义了最小重组缓冲区大小（minimum reassembly buffer size），它是IPv4或IPv6的任何实现都必须保证支持的最小数据报大小。其值对于IPv4为576字节，对于IPv6为1500字节。例如，就IPv4而言，我们不能判定某个给定目的地能否接受577字节的数据报。为此有许多使用UDP的IPv4网络应用（如DNS、RIP、TFTP、BOOTP、SNMP）避免产生大于这个大小的数据报。
- TCP有一个MSS（maximum segment size，最大分节大小），用于向对端TCP通告对端在每个分节中能发送的最大TCP数据量。在图2-5中我们看到过SYN分节上的MSS选项。MSS的目的是告诉对端其重组缓冲区大小的实际值，从而试图避免分片。MSS经常设置成MTU减去IP和TCP首部的固定长度。在以太网中使用IPv4的MSS值为1460，使用IPv6的MSS值为1440（两者的TCP首部都是20个字节，但IPv4首部是20字节，IPv6首部却是40字节）。在TCP的MSS选项中，MSS值是一个16位的字段，限定其最大值为65 535。这对于IPv4是适合的，因为IPv4数据报中的最大TCP数据量为65 495（65 535减去IPv4首部的20字节和TCP首部的20字节）。然而对于具有特大净荷选项的IPv6，却需要使用另外一种技巧（RFC 2675［Borman, Deering, and Hinden 1999］）。首先，没有特大净荷选项的IPv6数据报中的最大TCP数据量为65 515（65 535减去TCP首部的20字节）。65 535这个MSS值于是被视为表示“无限”的一个特殊值。该值只在用到特大净荷选项时才使用，不过这种情况却要求实际的MTU超过65 535。其次，如果TCP使用特大净荷选项，并且接收到的对端通告的MSS为65 535，那么它所发送数据报的大小限制就是接口MTU。如果这个值太大（也就是说所在路径中某个链路的MTU比较小），那么路径MTU发现功能将确定这个较小值。
- SCTP基于到对端所有地址发现的最小路径MTU保持一个分片点。这个最小MTU大小用于把较大的用户消息分割成较小的能够以单个IP数据报发送的若干片段。SCTP_MAXSEG套接字选项可以影响该值，使得用户能够请求一个更小的分片点。 57

2.11.1 TCP 输出

图2-15展示了某个应用进程写数据到一个TCP套接字中时发生的步骤。

每一个TCP套接字有一个发送缓冲区，我们可以使用SO_SNDBUF套接字选项来更改该缓冲区的大小（见7.5节）。当某个应用进程调用write时，内核从该应用进程的缓冲区中复制所有数

据到所写套接字的发送缓冲区。如果该套接字的发送缓冲区容不下该应用进程的所有数据（或是应用进程的缓冲区大于套接字的发送缓冲区，或是套接字的发送缓冲区中已有其他数据），该应用进程将被投入睡眠。这里假设该套接字是阻塞的，它是通常的默认设置。（我们将在第16章中阐述非阻塞的套接字。）内核将不从write系统调用返回，直到应用进程缓冲区中的所有数据都复制到套接字发送缓冲区。因此，从写一个TCP套接字的write调用成功返回仅仅表示我们可以重新使用原来的应用进程缓冲区，并不表明对端的TCP或应用进程已接收到数据。（我们将在7.5节随SO_LINGER套接字选项详细讨论这一点。）

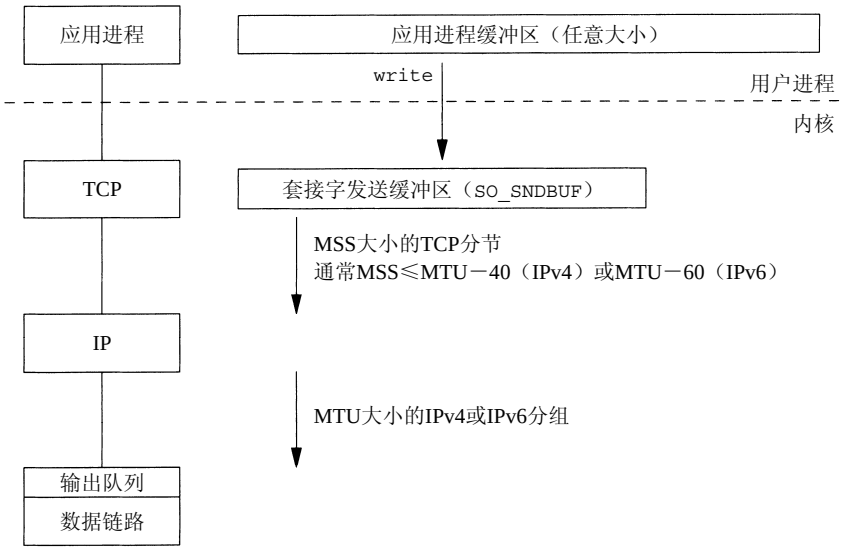

图2-15　应用进程写TCP套接字时涉及的步骤和缓冲区

这一端的TCP提取套接字发送缓冲区中的数据并把它发送给对端TCP，其过程基于TCP数据传送的所有规则（TCPv1的第19章和第20章）。对端TCP必须确认收到的数据，伴随来自对端的ACK的不断到达，本端TCP至此才能从套接字发送缓冲区中丢弃已确认的数据。TCP必须为已发送的数据保留一个副本，直到它被对端确认为止。

本端TCP以MSS大小的或更小的块把数据传递给IP，同时给每个数据块安上一个TCP首部以构成TCP分节，其中MSS或是由对端通告的值，或是536（若对端未发送一个MSS选项）。（536是IPv4最小重组缓冲区字节数576减去IPv4首部字节数20和TCP首部字节数20的结果。）IP给每个TCP分节安上一个IP首部以构成IP数据报，并按照其目的IP地址查找路由表项以确定外出接口，然后把数据报传递给相应的数据链路。IP可能在把数据报传递给数据链路之前将其分片，不过我们已经谈到MSS选项的目的之一就是试图避免分片，较新的实现还使用了路径MTU发现功能。每个数据链路都有一个输出队列，如果该队列已满，那么新到的分组将被丢弃，并沿协议栈向上返回一个错误：从数据链路到IP，再从IP到TCP。TCP将注意到这个错误，并在以后某个时刻重传相应的分节。应用进程并不知道这种暂时的情况。

58

2.11.2　UDP 输出

图2-16展示了某个应用进程写数据到一个UDP套接字中时发生的步骤。

这一次我们以虚线框展示套接字发送缓冲区，因为它实际上并不存在。任何UDP套接字都有发送缓冲区大小（我们可以使用SO_SNDBUF套接字选项更改它，见7.5节），不过它仅仅是可

写到该套接字的UDP数据报的大小上限。如果一个应用进程写一个大于套接字发送缓冲区大小的数据报，内核将返回该进程一个EMSGSIZE错误。既然UDP是不可靠的，它不必保存应用进程数据的一个副本，因此无需一个真正的发送缓冲区。（应用进程的数据在沿协议栈向下传递时，通常被复制到某种格式的一个内核缓冲区中，然而当该数据被发送之后，这个副本就被数据链路层丢弃了。）

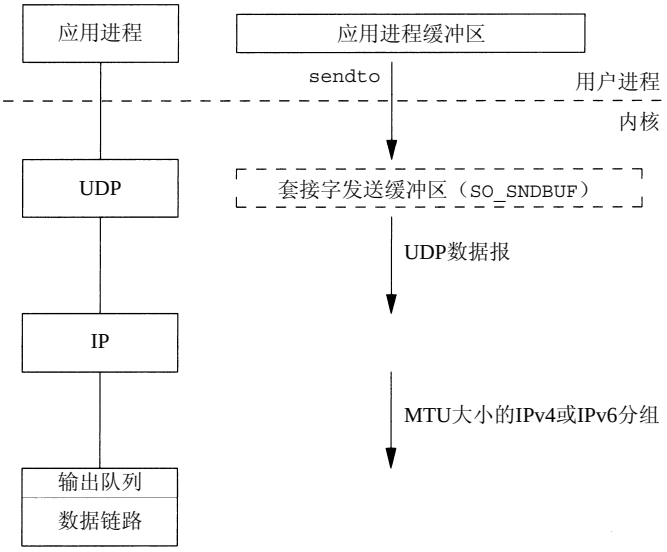

图2-16 应用进程写UDP套接字时涉及的步骤与缓冲区

这一端的UDP简单地给来自用户的数据报安上它的8字节的首部以构成UDP数据报，然后传递给IP。IPv4或IPv6给UDP数据报安上相应的IP首部以构成IP数据报，执行路由操作确定外出接口，然后或者直接把数据报加入数据链路层输出队列（如果适合于MTU），或者分片后再把每个片段加入数据链路层的输出队列。如果某个UDP应用进程发送大数据报（譬如说2000字节的数据报），那么它们相比TCP应用数据更有可能被分片，因为TCP会把应用数据划分成MSS大小的块，而UDP却没有对等的手段。

59

从写一个UDP套接字的write调用成功返回表示所写的数据报或其所有片段已被加入数据链路层的输出队列。如果该队列没有足够的空间存放该数据报或它的某个片段，内核通常会返回一个ENOBUFS错误给它的应用进程。

> 不幸的是，有些UDP的实现不返回这种错误，这样甚至数据报未经发送就被丢弃的情况应用进程也不知道。

2.11.3 SCTP输出

图2-17展示了某个应用进程写数据到一个SCTP套接字中时发生的步骤。

既然SCTP是与TCP类似的可靠协议，它的套接字也有一个发送缓冲区，而且跟TCP一样，我们可以用SO_SNDBUF套接字选项来更改这个缓冲区的大小（见7.5节）。当一个应用进程调用write时，内核从该应用进程的缓冲区中复制所有数据到所写套接字的发送缓冲区。如果该套接字的发送缓冲区容不下该应用进程的所有数据（或是应用进程的缓冲区大于套接字的发送缓

冲区，或是套接字的发送缓冲区中已有其他数据），应用进程将被投入睡眠。这里假设该套接字是阻塞的，它是通常的默认设置。（我们将在第16章中阐述非阻塞的套接字。）内核将不从write系统调用返回，直到应用进程缓冲区中的所有数据都复制到套接字发送缓冲区。因此，从写一个SCTP套接字的write调用成功返回仅仅表示我们可以重新使用原来的应用进程缓冲区，并不表明对端的SCTP或应用进程已接收到数据。

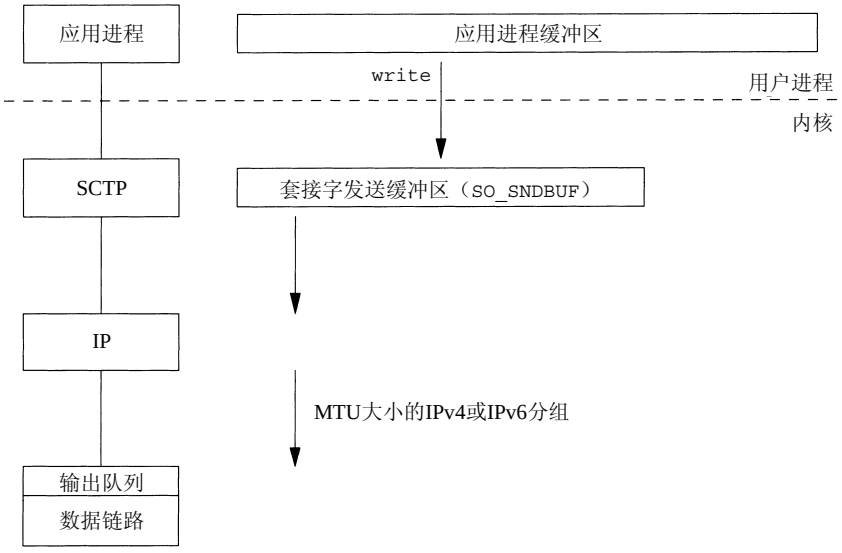

图2-17　应用进程写SCTP套接字时涉及的步骤和缓冲区

60

这一端的SCTP提取套接字发送缓冲区的数据并把它发送给对端SCTP，其过程基于SCTP数据传送的所有规则（数据传送的细节见［Stewart and Xie 2001］的第5章）。本端SCTP必须等待SACK，在累积确认点超过已发送的数据后，才可以从套接字缓冲区中删除该数据。

2.12　标准因特网服务

图2-18列出了TCP/IP多数实现都提供的若干标准服务。注意，表中所有服务同时使用TCP和UDP提供，并且这两个协议所用端口号也相同。

这些服务通常由Unix主机的inetd守护进程提供（见13.5节）。它们还提供使用标准的Telnet客户程序就能完成的简易测试机制。举例来说，下面就是时间获取和回射这两个标准服务器的测试过程：

```
aix % telnet freebsd daytime
Trying 12.106.32.254...                    Telnet客户输出
Connected to freebsd.unpbook.com.          Telnet客户输出
Escape character is '^]'.                  Telnet客户输出
Mon Jul 28 11:56:22 2003                   daytime服务器输出
Connection closed by foreign host.         Telnet客户输出（服务器关闭连接）

aix % telnet freebsd echo
Trying 12.106.32.254...                    Telnet客户输出
Connected to freebsd.unpbook.com.          Telnet客户输出
```

```
Escape character is '^]'.          Telnet客户输出
hello,world                        我们键入这行
hello,world                        它由服务器回射回来
^]                                 键入Ctrl+]以与Telnet客户交谈
telnet> quit                       告诉客户我们已测试完毕
Connection closed.                 这次客户自己关闭连接
```

名　字	TCP端口	UDP端口	RFC	说　明
echo（回射）	7	7	862	服务器返回客户发送的数据
discard（丢弃）	9	9	863	服务器废弃客户发送的数据
daytime（时间获取）	13	13	867	服务器返回直观可读的日期和时间
chargen（字符生成）	19	19	864	TCP服务器发送连续的字符流，直到客户终止连接。UDP服务器则每当客户发送一个数据报就返送一个包含随机数量（0~512）字符的数据报
time（流逝时间获取）	37	37	868	服务器返回一个32位二进制数值表示的时间。这个数值表示从1900年1月1日子时（UTC时间）以来所流逝的秒数

图2-18　大多数实现提供的标准TCP/IP服务[①]

在这两个例子中，我们键入主机名和服务名（daytime和echo）。这些服务名由/etc/services文件映射到图2-18所示的端口号，详见11.5节。 61

注意，当我们连接到daytime服务器时，服务器执行主动关闭，然而当连接到echo服务器时，客户执行主动关闭。回顾图2-4，我们知道执行主动关闭的那一端就是历经TIME_WAIT状态的那一端。

为了应付针对它们的拒绝服务攻击和其他资源使用攻击，在如今的系统中，这些简单的服务通常被禁用。

2.13　常见因特网应用的协议使用

图2-19总结了各种常见的因特网应用对协议的使用情况。

前两个因特网应用ping和traceroute是使用ICMP协议实现的网络诊断应用。traceroute自行构造UDP分组来发送并读取所引发的ICMP应答。

紧接着是3个流行的路由协议，它们展示了路由协议使用的各种传输协议。OSPF通过原始套接字直接使用IP，RIP使用UDP，BGP使用TCP。

接下来5个是基于UDP的网络应用，然后是7个TCP网络应用和4个同时使用UDP和TCP的网络应用，最后5个是IP电话网络应用，它们或者独自使用SCTP，或者选用UDP、TCP或SCTP。

① 本图同时给出了这些标准因特网服务的英文名称和中文名称，其中英文名称是正式名称（/etc/services文件使用这些名称）。之所以这么区分是因为本书围绕其中两种服务（回射和时间获取）的实现展开，为区分本书中的实现与各个Unix系统的内部实现，我们用中文名称称呼前者，用英文名称称呼后者（原书也对两者做了类似区分）。另外内部实现的服务总是使用标准端口号，本书实现的服务则可根据情况选择。因此当使用英文名称服务名时，必定与其标准端口号对应。——译者注

因特网应用	IP	ICMP	UDP	TCP	SCTP
ping		•			
traceroute		•	•		
OSPF（路由协议）	•				
RIP（路由协议）			•		
BGP（路由协议）				•	
BOOTP（引导协议）			•		
DHCP（引导协议）			•		
NTP（时间协议）			•		
TFTP（低级FTP）			•		
SNMP（网络管理）			•		
SMTP（电子邮件）				•	
Telnet（远程登录）				•	
SSH（安全的远程登录）				•	
FTP（文件传送）				•	
HTTP（Web）				•	
NNTP（网络新闻）				•	
LPR（远程打印）				•	
DNS（域名系统）			•	•	
NFS（网络文件系统）			•	•	
Sun RPC（远程过程调用）			•	•	
DCE RPC（远程过程调用）			•	•	
IUA（IP之上的ISDN）					•
M2UA/M3UA（SS7电话信令）					•
H.248（媒体网关控制）			•	•	•
H.323（IP电话）			•	•	•
SIP（IP电话）			•	•	•

62

图2-19　各种常见因特网应用的协议使用情况

2.14　小结

UDP是一个简单、不可靠、无连接的协议，而TCP是一个复杂、可靠、面向连接的协议。SCTP组合了这两个协议的一些特性，并提供了TCP所不具备的额外特性。尽管绝大多数因特网应用（Web、Telnet、FTP和电子邮件）使用TCP，但这3个协议对传输层都是必要的。在22.4节中我们将阐述选用UDP替代TCP的理由。在23.12节中我们将阐述选用SCTP替代TCP的理由。

TCP使用三路握手建立连接，使用四分组交换序列终止连接。当一个TCP连接被建立时，它从CLOSED状态转换到ESTABLISHED状态；当该连接被终止时，它又回到CLOSED状态。一个TCP连接可处于11种状态之一，其状态转换图给出了从一种状态转换到另一种状态的规则。理解状态转换图是使用`netstat`命令诊断网络问题的基础，也是理解当某个应用进程调用诸如`connect`、`accept`和`close`等函数时所发生过程的关键。

TCP的TIME_WAIT状态一直是一个造成网络编程人员混淆的来源。存在这一状态是为了实现TCP的全双工连接终止（即处理最终那个ACK丢失的情形），并允许老的重复分节从网络

SCTP使用四路握手建立关联；使用三分组交换序列终止关联。当一个SCTP关联被建立时，它从CLOSED状态转换到ESTABLISHED状态；当该关联被终止时，它又回到CLOSED状态。一个SCTP关联可处于8种状态之一，其状态转换图给出从一种状态转换到另一种状态的规则。SCTP不像TCP那样需要TIME_WAIT状态，因为它使用了验证标记。

习题

2.1 我们已经提到IPv4（IP版本4）和IPv6（版本6）。IP版本5情况如何，IP版本0、1、2和3又是什么？（提示：查IANA的“Internet Protocol”注册处。要是你无法访问IANA所在网址http://www.iana.org，那就查看附录中的解答吧。）

2.2 你从哪里可以找到有关IP版本5的信息？

2.3 在讲解图2-15时我们说过，如果没收到来自对端的MSS选项，本端TCP就采用536这个MSS值。为什么使用这个值？

2.4 给在第1章中讲解的时间获取客户/服务器应用画出类似于图2-5的分组交换过程，假设服务器在单个TCP分节中返回26个字节的完整数据。

2.5 在一个以太网上的主机和一个令牌环网上的主机之间建立一个连接，其中以太网上主机的TCP通告的MSS为1460，令牌环网上主机的TCP通告的MSS为4096。两个主机都没有实现路径MTU发现功能。观察分组，我们在两个相反方向上都找不到大于1460字节的数据，为什么？ 63

2.6 在讲解图2-19时我们说过OSPF直接使用IP。承载OSPF数据报的IPv4首部（见图A-1）的协议字段是什么值？

2.7 在讨论SCTP输出时我们说过，SCTP发送端必须等待累积确认点超过已发送的数据，才可以从套接字缓冲区中释放该数据。假设某个选择性确认（SACK）表明累积确认点之后的数据也得到了确认，这样的数据为什么却不能被释放呢？ 64