第5章 网 络 层

网络层关注的是如何将源端数据包一路送到目标方。为了到达目标方，可能要求沿途经过许多跳(hop)中间路由器。这种功能显然与数据链路层的功能不同，数据链路层的目标没那么“宏伟”，只是将帧从(虚拟的)“线路”一端传送到另一端。因此，网络层是处理端到端数据传输的最底层。

为了实现这个目标，网络层必须知道网络拓扑结构(即所有路由器和链路的集合)，并从中计算出适当的路径，即使是大型的网络。同时，网络层还必须仔细选择路由器，避免某些通信线路和路由器负载过重，而其他路由器和线路空闲。最后，当源和目标方位于不同的、独立运营的网络(有时称为自治系统)中时，新的挑战又会出现，比如跨越多个网络协调流量流和管理网络的使用情况。这些问题通常都需要由网络层解决；网络运营商通常用手工方式处理这些问题。传统上，网络运营商不得不采用手工方式在底层重新配置网络层。然而，最近，随着软件定义网络和可编程硬件的出现，使得有可能通过高层的软件程序配置网络层，甚至可以整体上重新定义网络层的功能。在本章中，将介绍并演示所有这些问题，尤其是 Internet 和它的网络层协议(Internet Protocol，IP)的问题。

5.1 网络层的设计问题

本节将简要描述网络层设计人员必须尽力解决的一些问题，其中包括向传输层提供的服务以及网络的内部设计。

5.1.1 存储-转发数据包交换

在开始介绍网络层细节之前，有必要再次说明网络层协议运行的环境。从图 5-1 可以看到这样的环境。网络中最主要的组件是 ISP 的设备(路由器、交换机以及通过传输线路连接的中间盒)和客户的设备，在图 5-1 中，ISP 的设备位于阴影椭圆内，而客户设备位于椭圆之外。主机 H1 直接连接到 ISP 的路由器 A，这或许是一台家用计算机，通过 DSL 调制解调器接入。而 H2 位于一个 LAN 中，这可能是一个办公室以太网，其中有一台路由器 F，客户拥有并运行该路由器。路由器 F 通过一条租用线路连接到 ISP 的设备上。可以看到，F

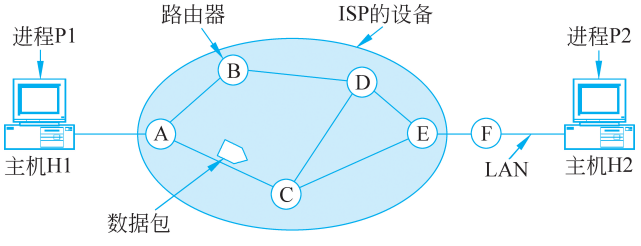


图 5-1 网络层协议的环境

位于椭圆的外面，因为它不属于 ISP。然而，为了本章的目的，还是把客户侧的路由器作为 ISP 网络的一部分来考虑，因为它们运行的算法与 ISP 路由器上运行的算法相同（这里的主要关注点是算法）。

这种网络配置的使用方式如下所述。如果一台主机要发送一个数据包，它就将数据包传输给最近的路由器，该路由器可能在它自己的 LAN 中，也可能在一条通向 ISP 的点到点链路上。该数据包首先被存储在路由器上，直到它完全到达，并且该链路完成了对它的校验和的验证处理。然后它被沿着路径转发到下一台路由器，直至到达目标主机，在目标主机上完成数据递交。这种机制就是存储-转发数据包交换，在前面几章已经作了介绍。

5.1.2　提供给传输层的服务

网络层通过网络层/传输层接口向传输层提供服务。设计中，一个重要的问题是明确网络层向传输层提供什么类型的服务。这些服务需要谨慎地进行设计，在实现下面这些目标：

（1）这些服务应该独立于路由器技术。

（2）应该向传输层屏蔽路由器的数量、类型和拓扑结构。

（3）可供传输层使用的网络地址应该使用统一的编址方案，甚至可以跨越 LAN 和 WAN。

给定这些目标后，网络层设计者有很大的自由度来制定这些提供给传输层的服务的详细规范。这种自由度通常演变为两个阵营之间的激烈争论。最终的讨论集中在网络层应该提供面向连接的服务还是提供无连接的服务这一点上。

一个阵营（以 Internet 社团为代表）认为，路由器的任务仅仅是传送数据包，不用再做别的事情。按照这种观点（基于 40 年来从一个实际计算机网络获得的经验），不管如何设计网络，从本质上讲，网络总是不可靠的。因此，主机应该接受这样的事实，自己完成错误控制（即错误检测和纠正）和流量控制。

这种观点导致了这样的结论：网络服务应该是无连接的，只需要原语 SEND PACKET 和 RECEIVE PACKET 以及少量其他的原语就够了。特别是数据包的排序和流量控制不应该在这里完成，因为主机将会完成这些工作，做两遍同样的工作通常不会带来更多好处。这个推理就是**端到端的观点**（end-to-end argument）的例子，这种设计原则对 Internet 的形成有着很大的影响（Saltzer 等，1984）。而且，每个数据包必须携带完整的目标地址，因为每个数据包的运送独立于它前面的那些数据包（如果此前有数据包）。

另一个阵营（以电话公司为代表）认为，网络应该提供可靠的、面向连接的服务。他们声称，具有 100 多年成功经验的全球电话系统就是一个极好的指导。按照这一观点，服务质量是最主要的因素，并且，如果在网络中没有连接，要实现服务质量非常困难，特别对于诸如语音和视频这样的实时流量。

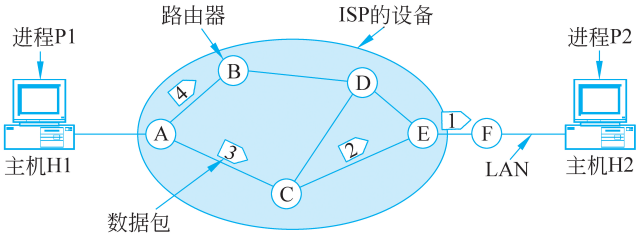
即使几十年以后，这场争论仍然十分活跃。早期被广泛使用的数据网络都是面向连接的，比如 20 世纪 70 年代的 X.25 和 20 世纪 80 年代代替它的**帧中继**（Frame Relay）。然而，自从有了 ARPANET 和早期 Internet，无连接网络层得到了突飞猛进的普及。现在 IP 俨然是一个无时不在的成功象征。虽然在 20 世纪 80 年代它受到一种称为 ATM 的面向连接技术的威胁，该技术就是为了推翻 IP 而开发的，但是结果刚好相反，现在 ATM 终于找到了自己的用武之地，而 IP 正在接管电话网络。然而，在幕后，Internet 正朝着面向连接的特性进

化,因为服务质量变得越来越重要了。两个面向连接技术的例子是多协议标签交换(MultiProtocol Label Switching,MPLS)和VLAN,本章将描述MPLS,有关VLAN在第4章中已经介绍过了。目前这两种技术都已经被广泛使用。

5.1.3 无连接服务的实现

在考察了网络层向它的用户提供的两类服务以后,现在来看网络层内部是如何工作的。根据网络层提供的服务类型,可能存在两种不同的组织方式。如果网络层提供的是无连接的服务,那么,所有的数据包都被独立地输入网络,并且每个数据包独立路由,不需要提前建立任何设置。在这样的环境中,数据包通常称为**数据报**(datagram),它类似于电报(telegram),这种网络称为**数据报网络**(datagram network)。如果网络层提供了面向连接的服务,那么,在发送任何数据包以后,必须首先建立一条从源路由器到目标路由器之间全程的路径。这个连接称为**虚电路**(Virtual Circuit,**VC**),类似于(老式)电话系统中建立的物理电路,这种网络称为**虚电路网络**(virtual-circuit network)。在本节,将讨论数据报网络;在下一节,将讨论虚电路网络。

现在来看数据报网络是如何工作的。假设图5-2中的进程P1有一个很长的消息要发送给进程P2。它将消息转交给传输层,并指示传输层将消息递交给主机H2上的进程P2。传输层代码运行在H1上,通常在操作系统内部。它在消息的前面加上一个传输头,然后将结果交给网络层,这里的网络层可能只是操作系统内部的另一个过程。



A的路由表(初始化)

A	—
B	B
C	C
D	B
E	C
F	C

A的路由表(稍后)

A	—
B	B
C	C
D	B
E	B
F	B

C的路由表

A	A
B	A
C	—
D	E
E	E
F	E

E的路由表

A	C
B	D
C	C
D	D
E	—
F	F

图5-2 数据报网络内部的路由

假设在这个例子中消息的长度是最大数据包长度的4倍,所以,网络层必须将消息拆分成4个数据包:1、2、3和4,然后用某一种点到点协议(比如PPP)将这些数据包依次发送给路由器A。到这里,ISP将消息的传输任务接管过来。每一台路由器都有一个内部的路由表,它指明了针对每一个可能的目标地址应该将数据包送到哪里去。每个表项是一对数据:目标地址和通往该目标地址使用的输出线路。只有直接连接的线路才可以作为输出线路。比如,在图5-2中,A只有两条输出线路——分别通往B和C,所以,每一个进来的数据包必须被转发给这两台路由器之一,即使它的最终目标地址是其他某一台路由器。A的初始路

由表在图 5-2 中标示为“初始化”。

在路由器 A 上，数据包 1、2 和 3 到达进来的链路并且经过校验和验证以后被路由器暂时保存起来。然后，根据 A 的路由表，每个数据包被放在一个新帧中，并转发到通往 C 的输出链路上。然后，数据包 1 被转发给 E，进一步又被转发给 F。当它到达 F 时，它被封装在一个帧内，通过 LAN 发送给 H2。数据包 2 和 3 遵循同样的路径。

然而，数据包 4 的情形有所不同。当它到达 A 时，尽管它的目标地址也是 F，但它被发送给路由器 B。出于某种原因，A 决定采用不同于前 3 个数据包的路径来发送数据包 4。或许它了解到在 ACE 路径上发生了流量拥塞，因而更新了路由表，这个路由表在图 5-2 中标示为“稍后”。管理这些路由表并做出路由决策的算法称为**路由算法**(routing algorithm)。路由算法是本章的一个主要话题。正如后面将会看到的那样，有几种不同类型的路由算法。

IP 是整个 Internet 的基础，它是无连接网络服务的经典案例。每个数据包携带一个目标 IP 地址，路由器使用该地址单独转发每一个数据包。IPv4 数据包的地址是 32 位的，IPv6 数据包的地址是 128 位的。在本章后面部分将详细描述 IP 和这两个版本。

5.1.4　面向连接服务的实现

对于面向连接服务，需要一个虚电路网络。现在讨论它是如何工作的。虚电路背后的思想是避免为每个要发送的数据包选择一条新路径(像图 5-2 所示的那样)。相反，当建立一个连接时，从源主机到目标主机之间的一条路径就被当作这个连接设置的一部分确定下来，并且保存在这些路由器的路由表中。所有在这个连接上通过的流量都使用这条路径，这与电话系统的工作方式完全一致。当连接被释放时，虚电路也随之终止。采用面向连接服务时，每个数据包携带一个标识符，指明了它属于哪一个虚电路。

作为一个例子，考虑图 5-3 的情形。在这里，主机 H1 已经建立了一个与主机 H2 之间的连接，其标识符为 1。该连接被记录在每个路由表中的第一项。A 的路由表的第一行说明：如果一个标示了连接 1 的数据包来自 H1，那么它将被发送到路由器 C，并且被赋予连接标识符 1。类似地，C 的第一项将该数据包路由到 E，也被赋予连接标识符 1。

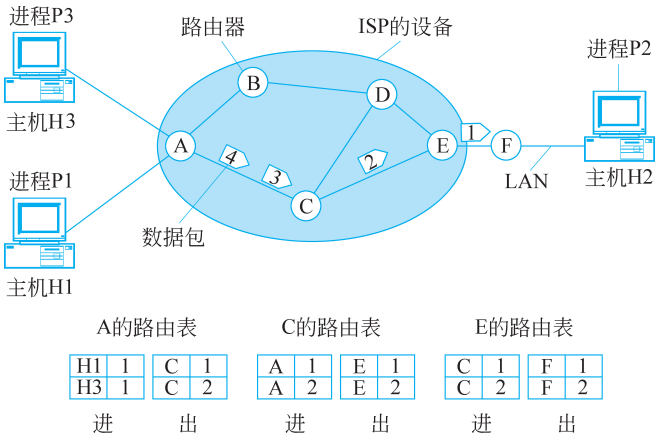


图 5-3　虚电路网络的路由

现在考虑 H3 也希望与 H2 建立连接时情形会怎么样。H3 选择连接 1(因为是它发起

连接,而且也是它唯一的连接),并且告诉网络要建立虚电路。这导致路由表创建第二行。请注意,这里出现了一个冲突:尽管A很容易区分出标识为连接1的数据包是来自H1还是来自H3,但C无法区分它们。基于这个原因,A给第二个连接的输出流量分配一个不同的连接标识符。这种避免冲突的做法正是路由器需要具备在输出数据包中替换连接标识符的能力的原因。

面向连接服务的一个例子是MPLS。它被用在Internet的ISP网络中,IP数据包被包装在一个有20位连接标识符(称为标签)的MPLS头中。MPLS往往对客户是隐藏的,ISP用它为超大流量建立长期的连接。但是,当服务质量变得很重要而且还需要其他ISP流量管理任务时,MPLS更适于提供相应的帮助。在本章后面将更多地介绍MPLS。

5.1.5 数据报网络与虚电路网络的比较

数据报网络和虚电路网络都有各自的支持者和反对者。本节从多个角度对两方的论点做个总结。图5-4列出了主要的论点,尽管总是有可能找到其中每一项的反例。

问　题	数据报网络	虚电路网络
电路建立	不需要	需要
寻址	每个数据包包含全部的源和目标地址	每个数据包包含简短的虚电路号
状态信息	路由器不保留连接的状态信息	针对每个连接,每条虚电路都需要路由器保存其状态
路由	每个数据包被单独路由	建立虚电路时选择路由,所有数据包都遵循该路由
路由器失效的影响	没有影响,除了在路由器崩溃期间丢失的数据包以外	穿过故障路由器的所有虚电路都将中断
服务质量	困难	容易,如果在预先建立每条虚电路时有足够的资源可分配
拥塞控制	困难	容易,如果在预先建立每条虚电路时有足够的资源可分配

图5-4　数据报网络和虚电路网络的比较

在网络内部,数据报网络和虚电路网络之间存在着几方面的权衡。一个权衡在于建立时间和地址解析时间。使用虚电路需要一个建立阶段,这个阶段既花费时间也消耗资源。然而,一旦付出了这个代价,则在虚电路网络中处理一个数据包就非常容易:路由器只要使用虚电路号作为索引,在路由表中找到该数据包的去向即可。在数据报网络中,不需要建立虚电路,但路由器需要执行一个更为复杂的查找过程,以便找到目标表项。

与此相关的一个问题是,数据报网络所用的目标地址比虚电路网络所用的虚电路号长得多,因为数据报网络的目标地址具备全局意义。如果数据包大多比较短,那么,在每个数据包中都包含完整的目标地址可能意味着有显著的额外开销,因而造成带宽浪费。

另一个问题是路由器内存中的路由表空间大小。数据报网络需要为每一个可能的目标地址都建立一个表项,而虚电路网络只需要为每一个虚电路提供一个表项即可。然而,这种优势多少有点迷惑性,因为建立连接时涉及的数据包也需要被路由,并且它们也使用目标地

址，如同数据报网络的做法一样。

从保证服务质量以及避免网络拥塞的角度，虚电路网络有一些优势，因为在虚电路网络建立连接时，可以提前预留资源（比如缓冲区、带宽和 CPU 周期）。一旦数据包开始到来，需要的带宽和路由器容量都已经准备就绪。而对于数据报网络，避免拥塞更为困难。

对于事务处理系统（比如商场购物时通过电话验证信用卡），用于建立和清除虚电路需要的开销有可能削弱虚电路的优势。如果预期大多数流量都是这种类型，那么在网络内部使用虚电路就意义不大。而在公司的两个办公楼之间长期运行诸如 VPN 流量的情况下，永久性的虚电路或许更加有用（手工建立虚电路，并且持续使用几个月或者几年）。

虚电路网络也存在脆弱性问题。如果一台路由器崩溃并且丢失了它的内存信息，那么，即使它在 1s 以后又恢复了，所有经过它的虚电路都将不得不中断。相反，如果数据报网络中的一台路由器宕机，那么，只有那些当时尚留在路由器队列中的数据包会受到影响（也可能不会受到影响，因为发送方可能很快重传这些数据包）。一条通信线路的失败对于使用了该线路的虚电路来说是致命的；但如果使用数据报，则这种失败很容易得到弥补。数据报网络还允许路由器平衡整个网络的流量，因为一个长序列的数据包传输路径可以部分被改变。

5.2　单个网络中的路由算法

网络层的主要功能是将数据包从源主机路由到目标主机。在本节中，讨论网络层如何在单个管理域或自治系统内实现这一功能。在大多数网络中，数据包需要经过多跳才能完成这一旅程。唯一一个值得指出的例外是广播网络，但即使在广播网络中，如果源主机和目标主机不在同一个网段中，路由仍然是一个问题。路由算法以及这些算法所用到的数据结构是网络层设计的主要内容。

路由算法（routing algorithm）是网络层软件的一部分，它负责确定一个进来的数据包应该被传输到哪一条输出线路上。如果使用数据报网络，那么必须针对每一个到达的数据包重新进行路由决策，因为自上一次选择了路径以后，最佳路径可能已经发生了改变。如果使用虚电路网络，那么只有当建立一条新的虚电路时才需要做路由决策。此后，数据包只需要沿着已经建立的路径进行传递即可。后一种情形有时候也称为**会话路由**（session routing），因为一条路径在整个会话过程中（比如 VPN 上的一个终端登录会话）保持有效。

有的时候对路由和转发这两个功能进行区分是非常有用的，路由是对使用哪些路径做出决策，而转发则是当一个数据包到达时决定应该采取什么动作。可以把路由器想象成内部有两个进程：一个进程在每个数据包到达的时候对它进行处理，在路由表中查找此数据包应该使用哪条输出线路，这个进程就是**转发**（forwarding）；另一个进程负责填充和更新路由表。这正是路由算法发挥作用的地方。

无论是针对每个数据包独立选择路径，还是仅在建立新连接时选择路径，路由算法都必须满足特定的要求：正确性、简单性、健壮性、稳定性、公平性和效率。

正确性和简单性无须多加解释，但对健壮性的要求则没有那么显而易见了。一旦一个重要网络投入运行，它有可能需要连续运行数年而不允许有系统范围的失败。在此期间，将会出现各种各样的硬件和软件故障。主机、路由器和线路将会不停地失败，拓扑结构也将多次发生变化。路由算法应该能够处理拓扑结构和流量方面的各种变化，而且不能要求所有

主机上的所有任务都停止工作。可以想象,每当有某台路由器崩溃时,如果都需要网络重新启动,那么后果会有多严重。

稳定性对于路由算法来说也是一个重要目标。有一些路由算法无论运行多长时间都不会收敛到一个固定的路径集合。一个稳定的算法应该能达到平衡,并且保持这一平衡状态。它应该迅速收敛,因为在路由算法达到平衡之前,通信可能会被中断。

公平性和效率看起来显然是理所应当的——肯定不会有人反对这两种特性,但是,事实证明它们往往是两个相互矛盾的目标。举一个简单的例子说明这种冲突,如图5-5所示。假设在A和A′之间、B和B′之间以及C和C′之间有足够的流量使得水平的链路达到饱和。为了使总的流量达到最大,X和X′之间的流量应该完全被切断。不幸的是,X和X′可能看不到这一点。很显然,在全局效率和单独连接的公平性之间需要有某种折中。

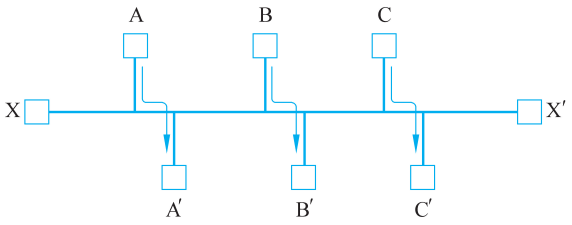


图5-5 公平性和效率发生冲突的网络

在找到公平性和效率之间的权衡办法之前,必须确定要优化什么性能指标。使数据包的平均延迟达到最小是有效地通过网络发送流量的一个显而易见的目标,但是使网络的总吞吐量最大化也是一个不错的目标。而且,这两个目标也是相互冲突的,因为运行任何一个接近容量的排队系统都意味着排队延迟很长。作为一种折中,许多网络尝试最小化一个数据包必须经过的距离,或者简单地减少一个数据包必须经历的跳数。无论哪一种选择,都有利于降低延迟,同时减少每个数据包消耗的带宽数量,这往往也有助于提高整个网络的吞吐量。

路由算法可以分成两大类:非自适应算法和自适应算法。**非自适应算法**(nonadaptive algorithm)不会根据对当前拓扑结构和流量的任何测量或评估来调整它们的路由决策。从I到J(对所有的I和J之间的数据包)使用的路径选择是预先在离线情况下计算好并在网络启动时被下载到路由器中的。这个过程有时候也称为**静态路由**(static routing)。因为它无法响应故障,所以静态路由对于路由选择已经很确定的场合非常有用。比如,在图5-3中,路由器F应该把指向网络的数据包都发送给路由器E,而不管最终目的地在哪里。

与此相反,**自适应算法**(adaptive algorithm)则会改变它们的路由决策以反映拓扑结构的变化,有时还会反映流量的变化。这些**动态路由**(dynamic ronting)算法在多个方面有所不同:获取信息的来源不同(比如,来自本地、相邻路由器或者所有路由器)、改变路径的时间不同(比如,每当拓扑发生变化时就改变路径,或者每隔Δt秒随负载改变路径)以及用于路由优化的度量不同(比如,距离、跳数或者估计的传输时间)。

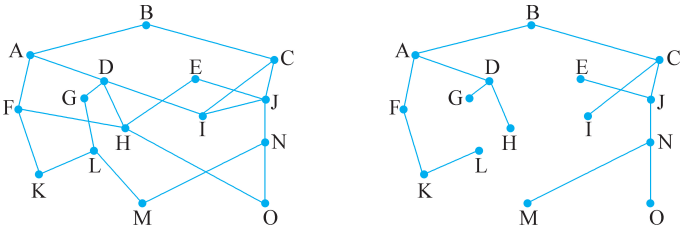
在下面几节中,将讨论不同的路由算法。这些算法除了从源发送数据包到接收方以外,还涵盖递交模型。有时候路由的目标是把数据包发送到一组目标地址中的全部、多个或者

一个地址。这里描述的所有路由算法都基于拓扑结构进行路由决策，而把基于流量进行路由决策的可能性放在 5.3 节中介绍。

5.2.1　优化原则

在讨论具体的算法以前，先不考虑网络拓扑结构或流量情况，而是给出最优路径的一般性论述，这可能会有助于对路由算法的理解。这个论述称为**最优化原则**(optimality principle)(Bellman，1957)。其表述如下：如果路由器 J 在从路由器 I 到路由器 K 的最优路径上，那么从 J 到 K 的最优路径也必定落在从 I 到 K 的最优路径上。为了看清这一点，将从 I 到 J 的路径部分记作 r_1，余下的路径部分记作 r_2。如果从 J 到 K 还存在一条路径比 r_2 更好，那么，它可以与 r_1 级联，从而可以改善从 I 到 K 的路径，这与 $r_1 r_2$ 是最优路径的声明矛盾。

作为最优化原则的一个直接结果，可以看到，在图 5-6(a)所示的网络中，从所有的源到一个指定目标的最优路径的集合构成了一棵以目标节点为根的树。这样的树称为**汇集树**(sink tree)，如图 5-6(b)所示。在图 5-6 中，距离度量是跳数。所有路由算法的目标是为所有路由器发现和使用这些汇集树。



(a) 一个网络　　(b) 以路由器B为根的汇集树

图 5-6　一个网络及其汇集树

请注意，汇集树不一定是唯一的，有可能存在具有相同路径长度的其他汇集树。如果允许选择所有可能的路径，则这棵树就变成了更一般的结构，称为 DAG(Directed Acyclic Graph，**有向无环图**)。DAG 没有环。后面将把汇集树用作两种情况下的便利表述。这两种情况都依赖于这些路径不会互相干扰的技术假设，所以，一条路径上的流量拥堵不会造成另一条路径的变动。

由于汇集树确实是一棵树，它不包含任何环，所以每个数据包将在有限的跳数内完成传递。在实践中，情形并非如此简单。在运行过程中，链路和路由器可能会出现故障，然后又恢复运行，所以，不同的路由器对当前拓扑结构的了解可能有所不同。而且，必须做出以下决定：每台路由器是独立地获取用于计算汇集树的信息还是通过其他的方法收集这些信息。稍后将具体讨论这个问题。不管怎么样，最优化原则和汇集树为测量其他路由算法提供了一个基准。

5.2.2　最短路径算法

本节从一个简单技术开始介绍路由算法：给定一个完整的网络图，用这一技术计算最优路径。这些路径正是分布式路由算法要发现的路径，尽管并非全部路由器都知道网络的

所有细节。

基本想法是：构造一个代表网络的图，图中的每个节点代表一台路由器，每条边代表一条通信线路或者链路。为了在一对给定的路由器之间选择一条路径，路由算法只需要在图中找出它们之间的最短路径。

对**最短路径**(shortest path)概念有必要做一些解释。一种测量路径长度的方法是计算跳数。若采用这种度量方法，则图5-7中ABC和ABE两条路径是等长的。另一种度量方法是计算以千米为单位的物理距离，在这种情况下，ABC明显比ABE长很多(假定该图是按照比例画的)。

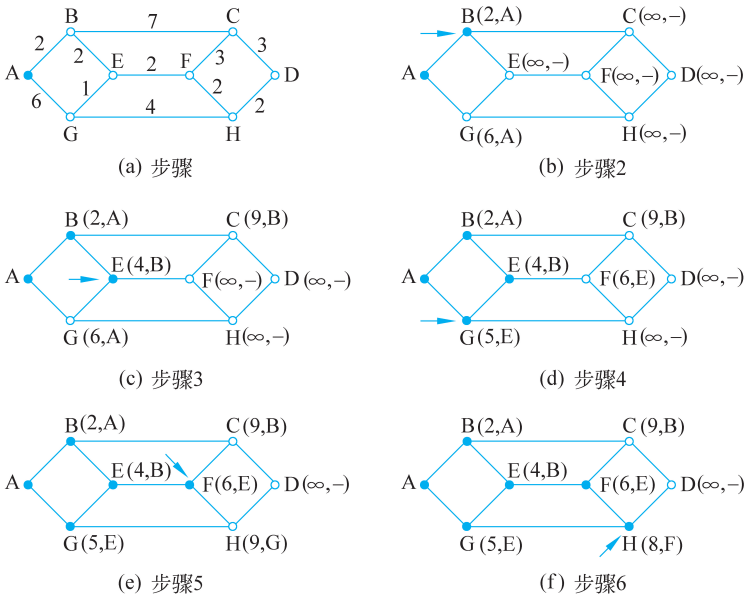


图5-7 计算从A到D的最短路径的前6个步骤

然而，除跳数和物理距离以外，还有许多其他度量方法也是可能的。比如，图中每条边用一个标准测试包的平均延迟来标记，这种延迟是每小时的测量结果。采用这种标记方法，最短路径就是最快路径，而不是边数最少或者距离最短的路径。

一般情况下，边上面的标记可以是距离、带宽、平均流量、通信成本、测量的延迟以及其他因素的一个函数，称为权重函数。通过改变权重函数，该算法就可以根据任何一种标准或者多种标准的组合来计算最短路径。

计算一个图的两个节点之间最短路径的算法有很多。本节采用的算法要归功于Dijkstra的研究(1959)。利用这个算法能找出网络中从一个源到全部目标节点之间的最短路径。每一个节点都标出了(在图5-7中节点后面的括号内)从源节点沿着已知的最佳路径到达本节点的距离。距离必须是非负的，如果基于带宽和延迟这样的实际测量值，那么距离本来就不可能为负。初始时，所有的路径都不确定，因此所有节点都被标记为无限远。随着算法的不断进行，陆续有一些路径被找到，于是节点的标记可能发生变化，以便反映出更好的路径。每个标记可能是暂时性的，也可能是永久性的。初始时，所有的标记都是暂时性的。当发现一个标记代表了从源到指定节点的最短可能路径时，该标记就变成永久性的，以

后不再改变。

为了说明标记算法的工作过程，请看图 5-7(a)中的加权无向图，这里的权值代表了一种度量，比如距离。现在要找到从 A 到 D 的最短路径。开始时，将节点 A 标记为永久性的，在图中用一个实心圆表示。然后依次检查每一个与 A(工作节点)相邻的节点(探测节点)，并且用它们与 A 之间的距离重新进行标记。每当一个节点被重新标记时，也要对探测节点进行标记，这样后面可以重构出最终路径。如果该网络具有多条从 A 到 D 的最短路径，并且希望找到所有这些路径，那么就需要记住所有以相同距离到达一个节点的探测节点。

在检查了每一个与 A 相邻的节点以后，再检查整个图中所有具有暂时性标记的节点，并且使得其中具有最小标记的节点成为永久性的，如图 5-7(b)的节点 B。这个节点就变成新的工作节点。

接下来从 B 开始，检查所有与 B 相邻的节点。对于每一个考虑的相邻节点，如果节点 B 上的标记加上从 B 到该节点的距离小于该节点上的标记，则说明获得了一条更短的路径，所以该节点需要重新标记。

在检查了所有与工作节点相邻的节点并且(如果可能)修改了暂时性标记以后，需要对整个图进行搜索，找到具有最小标记值的暂时性节点。这个节点将变成永久性的，并且成为下一轮的工作节点。图 5-7 显示了该算法的前 6 个步骤。

为了看清楚该算法的工作原理，请看图 5-7(c)。此时，刚刚把 E 变成永久性节点。假设存在一条比 ABE 还要短的路径，比如 AXYZE(对某个 X 和 Y)。有两种可能性：节点 Z 已经是永久性的，或者它还没有变成永久性节点。如果 Z 是永久性的，则 E 已经被探测过了(当 Z 变成永久性节点之后的下一轮探测)，所以路径 AXYZE 不会超出搜索范围，因而它不可能是一条最短路径。

现在考虑 Z 仍然是暂时性标记的情形。如果节点 Z 上的标记大于或者等于 E 上的标记，则 AXYZE 不可能是一条比 ABE 更短的路径；如果 Z 上的标记小于 E 上的标记，则应该首先是 Z 而不是 E 变成永久性节点，允许从 Z 探测 E。

图 5-8 给出了这个算法的 C 语言版本。全局变量 n 和 dist 描述了图，它们在 shortest_path()被调用以前被初始化。这一程序与前面描述的算法之间的唯一区别是，在图 5-8 所示的程序中，从目标节点 t 开始计算最短路径，而不是从源节点 s 开始。

由于在一个无向图中从 t 到 s 的最短路径与从 s 到 t 的最短路径是相同的，所以，无论从哪一端开始计算都并没有关系。之所以选择后向搜索的原因是：每个节点都被标记了它的前驱节点而不是它的后继节点。当最终的路径被复制到输出变量 path 中的时候，该路径因此被逆转。两次逆转的效果刚好相互抵消，于是结果按正确的顺序产生。

5.2.3　泛洪算法

在实现一个路由算法时，每台路由器必须根据本地知识而不是网络的全貌做决策。一个简单的本地技术是**泛洪**(flooding)，这种技术将每一个进来的数据包发送到除了该数据包到达的那条线路以外的每一条输出线路上。

很显然，泛洪算法会产生大量的重复数据包，事实上，除非采取某些措施抑制这一过程，否则将会产生无限多的数据包。其中一种措施是在每个数据包的头包含一个跳计数器，每经过一跳，跳计数器减 1，当计数器为 0 时就丢弃该数据包。理想情况下，跳计数器应该被