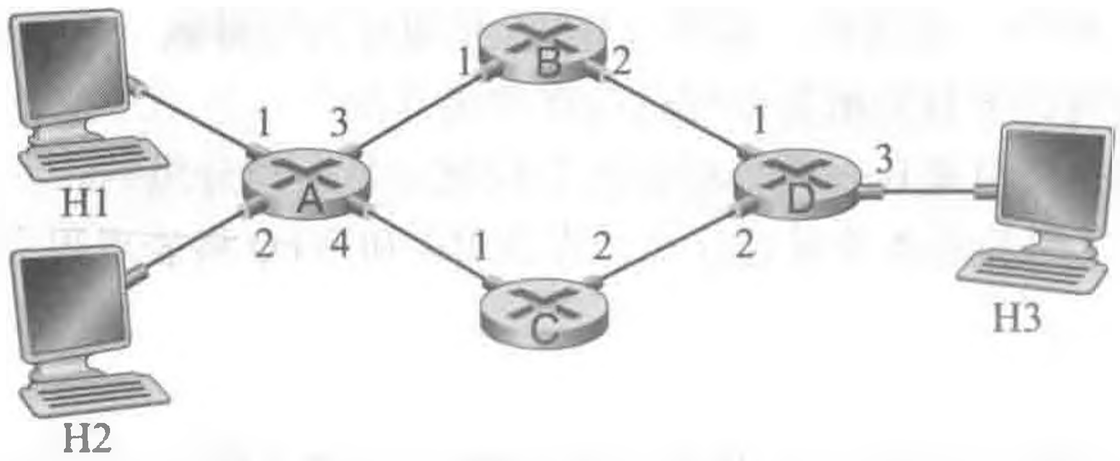


P2. 假设两个分组在完全相同的时刻到达一台路由器的两个不同输入端口。同时假设在该路由器中没有其他分组。

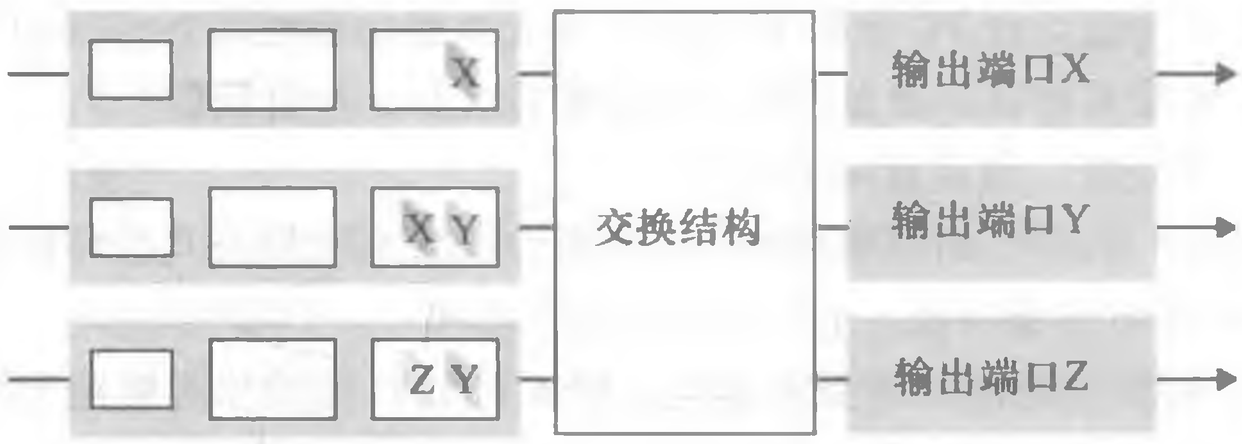
a. 假设这两个分组朝着两个不同的输出端口转发。当交换结构使用一条共享总线时，这两个分组可能在相同时刻通过该交换结构转发吗？

b. 假设这两个分组朝着两个不同的输出端口转发。当交换结构使用经内存交换时，这两个分组可能在相同时刻通过该交换结构转发吗？

c. 假设这两个分组朝着相同的输出端口转发。当交换结构使用纵横式时，这两个分组可能在相同时刻通过该交换结构转发吗？

P3. 在 4.2 节中，我们注意到如果交换结构速率是输入线路速率的 n 倍，其最大的排队时延为 $(n-1)D$。假设所有分组有相同长度，在相同时刻 n 个分组到达 n 个输出端口，同时所有 n 个分组要转发到不同的输出端口。对于内存、总线和纵横式交换结构，一个分组的最大时延是多少？

P4. 考虑下列交换机。假设所有数据报具有相同长度，交换机以一种分时隙、同步的方式运行，在一个时隙中一个数据报能够从某输入端口传送到某输出端口。其交换结构是纵横式的，因此在一个时隙中至多一个数据报能够传送到一个给定输出端口，但在一个时隙中不同的输出端口能够接收到来自不同输入端口的数据报，从输入端口到它们的输出端口传送所示的分组，所需的时隙数量最小是多少？此时假定使用你所需要的任何输入排队调度方法（即此时没有 HOL 阻塞）。假定采用你能够设计的最差情况下的调度方案，且非空输入队列不会空闲，所需的时隙数量最大是多少？



P5. 考虑使用 32 比特主机地址的某数据报网络。假定一台路由器具有 4 条链路，编号为 0～3，分组能被转发到如下的各链路接口：

目的地址范围	链路接口
11100000 00000000 00000000 00000000	
到	0
11100000 00111111 11111111 11111111	
11100000 01000000 00000000 00000000	
到	1
11100000 01000000 11111111 11111111	
11100000 01000001 00000000 00000000	
到	2
11100001 01111111 11111111 11111111	
其他	3

a. 提供一个具有 5 个表项的转发表，使用最长前缀匹配，转发分组到正确的链路接口。

b. 描述你的转发表是如何为具有下列目的地址的数据报决定适当的链路接口的。

11001000 10010001 01010001 01010101

11100001 01000000 11000011 00111100

11100001 10000000 00010001 01110111

P6. 考虑使用 8 比特主机地址的某数据报网络。假定一台路由器使用最长前缀匹配并具有下列转发表：

前缀匹配	接口
00	0
010	1
011	2
10	2
11	3

对这 4 个接口中的每个，给出相应的目的主机地址的范围和在该范围中的地址数量。

P7. 考虑使用 8 比特主机地址的数据报网络。假定一台路由器使用最长前缀匹配并具有下列转发表：

前缀匹配	接口
1	0
10	1
111	2
其他	3

对这 4 个接口中的每个，给出相应的目的主机地址的范围和在该范围中的地址数量。

P8. 考虑互联 3 个子网（子网 1、子网 2 和子网 3）的一台路由器。假定这 3 个子网的所有接口要求具有前缀 223.1.17/24。还假定子网 1 要求支持多达 60 个接口，子网 2 要求支持多达 90 个接口，子网 3 要求支持多达 12 个接口。提供 3 个满足这些限制的网络地址（形式为 *a.b.c.d/x*）。

P9. 在 4.2.2 节中给出了一个转发表（使用最长前缀匹配）的例子。使用 *a.b.c.d/x* 记法代替二进制字符串记法，重写该转发表。

P10. 在习题 P5 中要求你给出转发表（使用最长前缀匹配）。使用 *a.b.c.d/x* 记法代替二进制字符串记法，重写该转发表。

P11. 考虑一个具有前缀 128.119.40.128/26 的子网。给出能被分配给该网络的一个 IP 地址（形式为 xxx.xxx.xxx.xxx）的例子。假定一个 ISP 拥有形式为 128.119.40.64/26 的地址块。假定它要从该地址块生成 4 个子网，每块具有相同数量的 IP 地址。这 4 个子网（形式为 *a.b.c.d/x*）的前缀是什么？

P12. 考虑图 4-20 中显示的拓扑。（在 12:00 以顺时针开始）标记具有主机的 3 个子网为网络 A、B 和 C，标记没有主机的子网为网络 D、E 和 F。

a. 为这 6 个子网分配网络地址，要满足下列限制：所有地址必须从 214.97.254/23 起分配；子网 A 应当具有足够地址以支持 250 个接口；子网 B 应当具有足够地址以支持 120 个接口；子网 C 应当具有足够地址以支持 120 个接口。当然，子网 D、E 和 F 应当支持两个接口。对于每个子网，分配采用的形式是 *a.b.c.d/x* 或 *a.b.c.d/x ~ e.f.g.h/y*。

b. 使用你对（a）部分的答案，为这 3 台路由器提供转发表（使用最长前缀匹配）。

P13. 使用美国因特网编码注册机构（http://www.arin.net/whois）的 whois 服务来确定三所大学所用的 IP 地址块。whois 服务能被用于确定某个特定的 IP 地址的确定地理位置吗？使用 www.maxmind.com 来确定位于这三所大学的 Web 服务器的位置。

P14. 考虑向具有 700 字节 MTU 的一条链路发送一个 2400 字节的数据报。假定初始数据报标有标识号 422。将会生成多少个分片？在生成相关分片的数据报中各个字段的值是多少？

P15. 假定在源主机 A 和目的主机 B 之间的数据报被限制为 1500 字节（包括首部）。假设 IP 首部为 20 字节，要发送一个 5MB 的 MP3 文件需要多少个数据报？解释你的答案是如何计算的。

P16. 考虑在图 4-25 中建立的网络。假定 ISP 现在为路由器分配地址 24.34.112.235，家庭网络的网络地址是 192.168.1/24。

a. 在家庭网络中为所有接口分配地址。

b. 假定每台主机具有两个进行中的 TCP 连接，所有都是针对主机 128.119.40.86 的 80 端口的。在 NAT 转换表中提供 6 个对应表项。

P17. 假设你有兴趣检测 NAT 后面的主机数量。你观察到在每个 IP 分组上 IP 层顺序地标出一个标识号。由一台主机生成的第一个 IP 分组的标识号是一个随机数，后继 IP 分组的标识号是顺序分配的。假设由 NAT 后面主机产生的所有 IP 分组都发往外部。

a. 基于这个观察，假定你能够俘获由 NAT 向外部发送的所有分组，你能概要给出一种简单的技术来检测 NAT 后面不同主机的数量吗？评估你的答案。

b. 如果标识号不是顺序分配而是随机分配的，这种技术还能正常工作吗？评估你的答案。

P18. 在这个习题中，我们将探讨 NAT 对 P2P 应用程序的影响。假定用户名为 Arnold 的对等方通过查询发现，用户名为 Bernard 的对等方有一个要下载的文件。同时假定 Bernard 和 Arnold 都位于 NAT 后面。尝试设计一种技术，使得 Arnold 与 Bernard 创建一条 TCP 连接，而不对 NAT 做应用特定的配置。如果难以设计这样的技术，试讨论其原因。

P19. 考虑显示在图 4-30 中的 SDN OpenFlow 网络。假定对于到达 s2 的数据报的期望转发行为如下：

- 来自主机 h5 或 h6 并且发往主机 h1 或 h2 的任何数据报应当通过输出端口 2 转发到输入端口 1。
- 来自主机 h1 或 h2 并且发往主机 h5 或 h6 的任何数据报应当通过输出端口 1 转发到输入端口 2。
- 任何在端口 1 或 2 到达并且发往主机 h3 或 h4 的数据报应当传递到特定的主机。
- 主机 h3 和 h4 应当能够向彼此发送数据报。

详述实现这种转发行为的 s2 中的流表项。

P20. 再次考虑显示在图 4-30 中的 SDN OpenFlow 网络。假定在 s2 对于来自主机 h3 或 h4 的数据报的期望转发行为如下：

- 任何来自主机 h3 并且发往主机 h1、h2、h5 或 h6 的数据报应当在网络中以顺时针方向转发。
- 任何来自主机 h4 并且发往主机 h1、h2、h5 或 h6 的数据报应当在网络中以逆时针方向转发。

详述实现这种转发行为的 s2 中的流表项。

P21. 再次考虑上面 P19 的场景。给出分组交换机 s1 和 s3 的流表项，使得具有 h3 或 h4 源地址的任何到达数据报被路由到在 IP 数据报的目的地址字段中定义的目的主机。（提示：你的转发表规则应当包括如下情况，即到达的数据报被发往直接连接的主机，或应当转发到相邻路由器以便传递到最终主机。）

P22. 再次考虑显示在图 4-30 中的 SDN OpenFlow 网络。假定我们希望交换机 s2 的功能像防火墙一样。在 s2 中定义实现下列防火墙行为的流表，以传递目的地为 h3 和 h4 的数据报（对下列四种防火墙行为，每种定义一张不同的流表）。不需要在 s2 中定义将流量转发到其他路由器的转发行为。

- 仅有从主机 h1 和 h6 到达的流量应当传递到主机 h3 或 h4（即从主机 h2 和 h5 到达的流量被阻塞）。
- 仅有 TCP 流量被允许传递给主机 h3 或 h4（即 UDP 流量被阻塞）。
- 仅有发往 h3 的流量被传递（即所有到 h4 的流量被阻塞）。
- 仅有来自 h1 并且发往 h3 的 UDP 流量被传递。所有其他流量被阻塞。

Wireshark 实验

在与本书配套的 Web 站点 www.pearsonhighered.com/cs-resources 上，你将找到一个 Wireshark 实验作业，该作业考察了 IP 协议的运行，特别是 IP 数据报的格式。

人物专访

Vinton G. Cerf

Vinton G. Cerf 是 Internet Evangelist for Google 公司的副总裁兼主席。他在 MCI 公司服务了 16 年，担任过各种职位，最后以技术战略部资深副总裁的身份结束了他在那里的任期。他作为 TCP/IP 协议和因特网体系结构的共同设计者而广为人知。1976 年到 1982 年在美国国防部高级研究计划署（DARPA）任职期间，他在领导因特网以及与因特网相关的数据分组和安全技术的研发方面发挥了重要作用。他于 2005 年获得了美国总统自由奖章，于 1977 年获得了美国国家技术奖章。他在斯坦福大学获得数学学士学位，在加利福尼亚大学洛杉矶分校（UCLA）获得了计算机科学的硕士和博士学位。

- 是什么使您专注于网络技术的呢?

20 世纪 60 年代末，我在 UCLA 一直做程序员的工作。我的工作得到了美国国防部高级研究计划署（那时叫 ARPA，现在叫 DARPA）的支持。我那时在刚创建不久的 ARPAnet 的网络测量中心，为 Leonard Kleinrock 教授的实验室工作。ARPAnet 的第一个节点于 1969 年 9 月 1 日安装在 UCLA。我负责为计算机编程，以获取有关 ARPAnet 的性能信息，并报告这些信息以便与数学模型作比较，预测网络性能。

我和其他几名研究生负责研制所谓的 ARPAnet 主机级协议，该协议的过程和格式将使得网络中许多不同类型的计算机相互交互。这是我进入分布式计算与通信新世界中的一次迷人的探索。

- 当您第一次设计该协议时，您曾想象过 IP 会像今天这样变得无所不在吗?

当我和 Bob Kahn 于 1973 年最初从事该项工作时，我想我们的注意力大多集中在这样一个重要的问题上：假定我们不能实际改变这些网络本身，那么怎样才能让异构的分组网络彼此互操作呢？我们希望能找到一种方法可以使任意多的分组交换网以透明的方式进行互联，以便主机彼此之间不做任何转换就能进行端到端通信。我认为我们那时已经知道了我们正在处理强大的和可扩充的技术，但还没清楚地想过有数亿台计算机都连入因特网时的世界会是什么样。

- 您现在能预见网络与因特网的未来吗？您认为在它们的发展中存在的最大挑战或障碍是什么?

我相信因特网本身以及一般的网络都将要继续扩大。已有令人信服的证据表明，在因特网上将有数十亿个因特网使能设备，包括移动电话、冰箱、个人数字助理、家用服务器、电视等家用电器，以及大批通常的便携机、服务器等。重大挑战包括支持移动性、电池寿命、网络接入链路的容量、以不受限的方式扩展网络光学核心的能力。设计因特网的星际扩展是我在喷气推进实验室深入研究的一项计划。我们需要从 IPv4（32 比特地址）过渡到 IPv6（128 比特）。要做的事情实在是太多了!

- 是谁激发了您的职业灵感?

我的同事 Bob Kahn、我的论文导师 Gerald Estrin、我最好的朋友 Steve Crocker（我们在高中就认识了，1960 年是他带我进入了计算机学科之门!），以及数千名今天仍在继续推动因特网发展的工程师。

- 您对进入网络/因特网领域的学生有什么忠告吗?

要跳出现有系统的限制来思考问题，想一想什么是可行的；随后再做艰苦工作以谋划如何从事物的当前状态到达所想的状态。要敢于想象：我和喷气推进实验室的 6 个同事一直在从事陆地因特网的星际扩展设计。这也许要花几十年才能实现，任务会一个接着一个地出现，可以用这句话来总结："一个人总是要不断地超越自我，否则还有什么乐趣可言?"

第5章

Computer Networking: A Top-Down Approach, Seventh Edition

网络层：控制平面

在本章中，我们将通过包含网络层的**控制平面**组件来完成我们的网络层之旅。控制平面作为一种网络范围的逻辑，不仅控制沿着从源主机到目的主机的端到端路径间的路由器如何转发数据报，而且控制网络层组件和服务如何配置和管理。在5.2节中，我们将包含传统的计算图中最低开销路径的路由选择算法。这些算法是两个广为部署的因特网路由选择协议OSPF和BGP的基础，我们将分别在5.3节和5.4节中涉及。如我们将看到的那样，OSPF是一种运行在单一ISP的网络中的路由选择算法。BGP是一种在因特网中用于互联所有网络的路由选择算法，因此常被称为因特网的“黏合剂”。传统上，控制平面功能与数据平面的转发功能在一起实现，在路由器中作为统一的整体。如我们在第4章所学习的那样，软件定义网络（SDN）在数据平面和控制平面之间做了明确分割，在一台分离的“控制器”服务中实现了控制平面功能，该控制器服务与它所控制的路由器的转发组件完全分开并远离。我们将在5.5节中讨论SDN控制器。

在5.6节和5.7节中，我们将涉及管理IP网络的某些具体细节：ICMP（互联网控制报文协议）和SNMP（简单网络管理协议）。

5.1 概述

我们通过回顾图4-2和图4-3，迅速建立起学习网络控制平面的环境。在这里，我们看到了转发表（在基于目的地转发的场景中）和流表（在通用转发的场景中）是链接网络层的数据平面和控制平面的首要元素。我们知道这些表定义了一台路由器的本地数据平面转发行为。我们看到在通用转发的场景下，所采取的动作（4.4.2节）不仅包括转发一个分组到达路由器的每个输出端口，而且能够丢弃一个分组、复制一个分组和/或重写第2、3或4层分组首部字段。

在本章中，我们将学习这些转发表和流表是如何计算、维护和安装的。在4.1节的网络层概述中，我们已经学习了完成这些工作有两种可能的方法。

- *每路由器控制*。图5-1显示了在每台路由器中运行一种路由选择算法的情况，每台路由器中都包含转发和路由选择功能。每台路由器有一个路由选择组件，用于与其他路由器中的路由选择组件通信，以计算其转发表的值。这种每路由器控制的方法在因特网中已经使用了几十年。将在5.3节和5.4节中学习的OSPF和BGP协议都是基于这种每路由器的方法进行控制的。
- *逻辑集中式控制*。图5-2显示了逻辑集中式控制器计算并分发转发表以供每台路由器使用的情况。如我们在4.4节中所见，通用的“匹配加动作”抽象允许执行传统的IP转发以及其他功能（负载共享、防火墙功能和NAT）的丰富集合，而这些功能先前是在单独的中间盒中实现的。

该控制器经一种定义良好的协议与每台路由器中的一个控制代理（CA）进行交互，以配置和管理该路由器的转发表。CA一般具有最少的功能，其任务是与控制器通信并且

按控制器命令行事。与图 5-1 中的路由选择算法不同，这些 CA 既不能直接相互交互，也不能主动参与计算转发表。这是每路由器控制和逻辑集中式控制之间的关键差异。

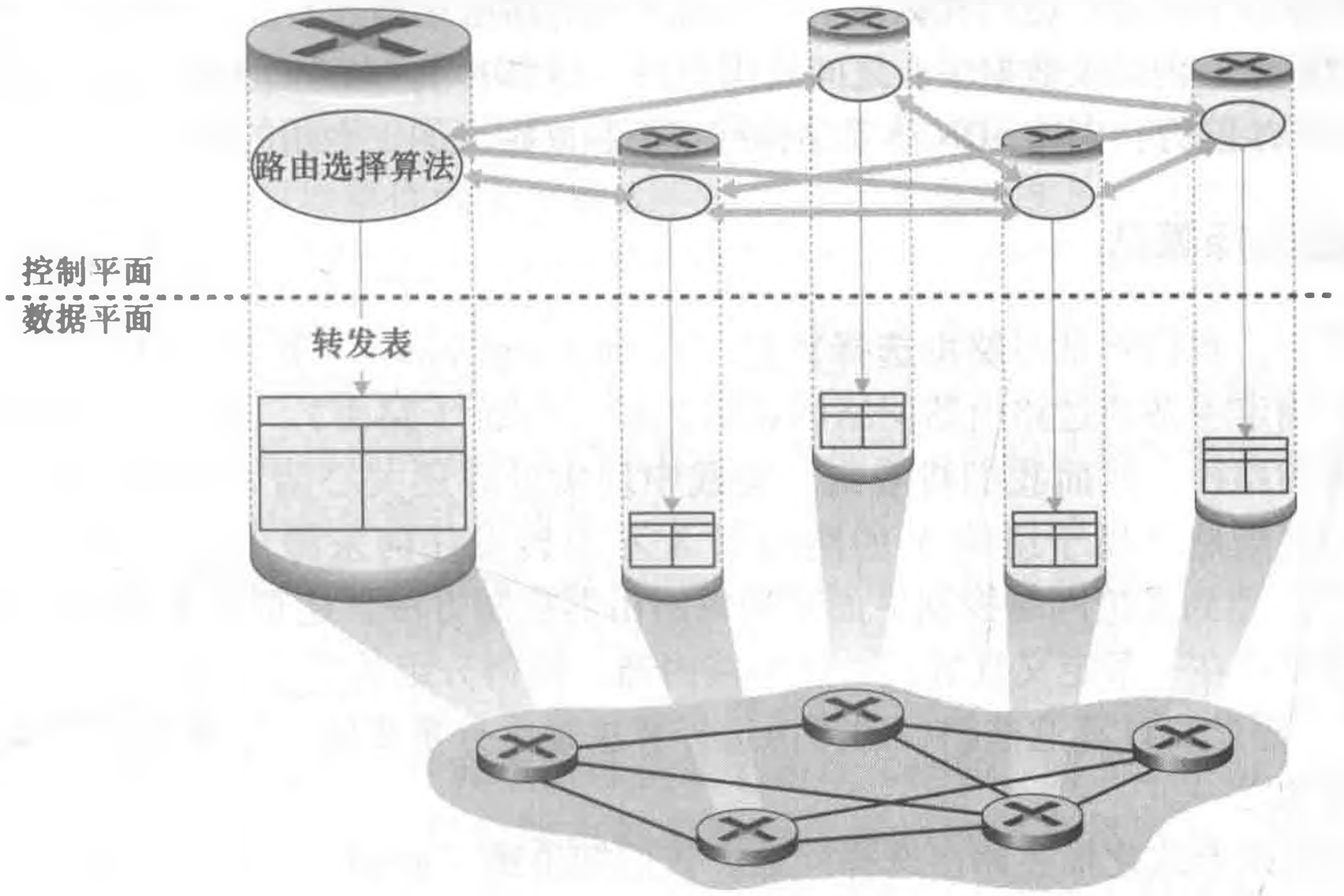


图 5-1　每路由器控制：在控制平面中各个路由选择算法组件相互作用

图 5-2　逻辑集中式控制：一个完全分开的（通常是远地的）控制器与本地控制代理交互

“逻辑集中式”控制［Levin 2012］意味着就像路由选择控制服务位于单一的集中服务点那样获取它们，即使该服务出于容错和性能扩展性的原因，很可能经由多个服务器实现。正如我们将在 5.5 节中所见，SDN 采用了逻辑集中式控制器的概念，而这种方法在生产部署中

得到了越来越多的应用。谷歌在它的内部 B4 全球广域网中使用了 SDN 控制路由器，该广域网互联了它的数据中心［Jain 2013］。来自微软研究院的 SWAN［Hong 2013］，使用了一个逻辑集中式控制器来管理广域网和数据中心网络之间的路由选择和转发。中国电信和中国联通在它们的数据中心内以及数据中心之间使用 SDN［Li 2015］。AT&T 已经通告［AT&T 2013］："支持许多 SDN 能力，支持 SDN 体系结构框架下独立定义的、专用的机制。"

5.2　路由选择算法

在本节中，我们将学习**路由选择算法**（routing algorithm），其目的是从发送方到接收方的过程中确定一条通过路由器网络的好的路径（等价于路由）。通常，一条好路径指具有最低开销的路径。然而我们将看到，实践中现实世界还关心诸如策略之类的问题（例如，有一个规则是"属于组织 Y 的路由器 X 不应转发任何来源于组织 Z 所属网络的分组"）。我们注意到无论网络控制平面采用每路由器控制方法，还是采用逻辑集中式控制方法，必定总是存在一条定义良好的一连串路由器，使得分组从发送主机到接收主机跨越网络"旅行"。因此，计算这些路径的路由选择算法是十分重要的，是最重要的 10 个十分重要的网络概念之一。

可以用图来形式化描述路由选择问题。我们知道**图**（graph）$G=(N, E)$ 是一个 N 个节点和 E 条边的集合，其中每条边是取自 N 的一对节点。在网络层路由选择的环境中，图中的节点表示路由器，这是做出分组转发决定的点；连接这些节点的边表示这些路由器之间的物理链路。这样一个计算机网络的图抽象显示在图 5-3 中。若要查看某些表示实际网络的图，参见［Dodge 2016；Cheswick 2000］；对于基于不同的图模型建模因特网的好坏的讨论，参见［Zegura 1997；Faloutsos 1999；Li 2004］。

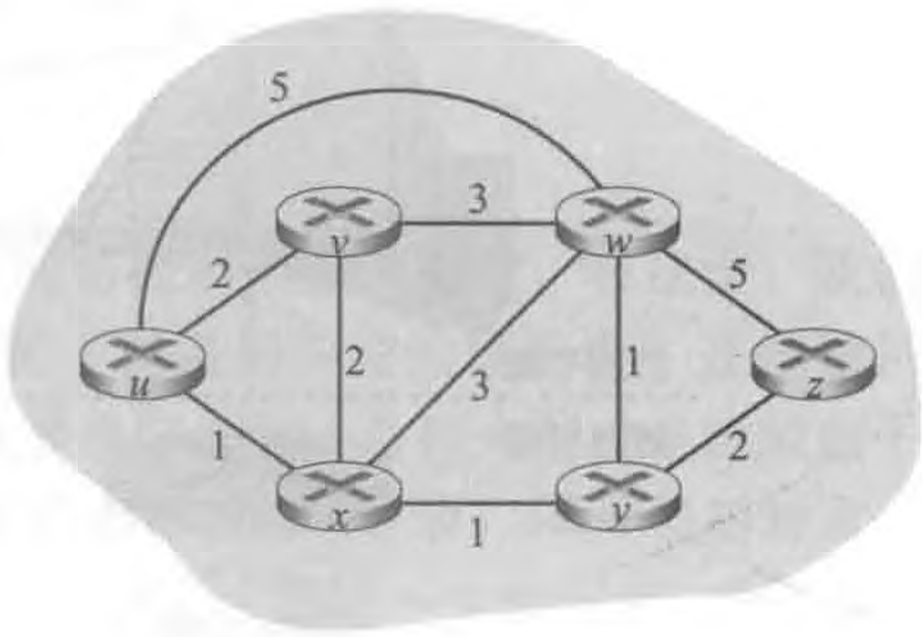


图 5-3　一个计算机网络的抽象图模型

如图 5-3 所示，一条边还有一个值表示它的开销。通常，一条边的开销可反映出对应链路的物理长度（例如一条越洋链路的开销可能比一条短途陆地链路的开销高），它的链路速度，或与该链路相关的金钱上的开销。为了我们的目的，我们只将这些链路开销看成是给定的，而不必操心这些值是如何确定的。对于 E 中的任一条边 (x, y)，我们用 $c(x, y)$ 表示节点 x 和 y 间边的开销。如果节点对 (x, y) 不属于 E，则置 $c(x, y)=\infty$。此外，我们在这里考虑的都是无向图（即图的边没有方向），因此边 (x, y) 与边 (y, x) 是相同的并且 $c(x, y)=c(y, x)$。然而，我们将学习的算法能够很容易地扩展到在每个方向有不同开销的有向链路场合。同时，如果 (x, y) 属于 E，节点 y 也被称为节点 x 的**邻居**（neighbor）。

在图抽象中为各条边指派了开销后，路由选择算法的天然目标是找出从源到目的地间的最低开销路径。为了使问题更为精确，回想在图 $G=(N, E)$ 中的一条**路径**（path）是一个节点序列 $(x_1, x_2, \cdots, x_p)$，这样每一个对 (x_1, x_2)，(x_2, x_3)，$\cdots$，(x_{p-1}, x_p) 是 E 中的边。路径 $(x_1, x_2, \cdots, x_p)$ 的开销只是沿着路径所有边的开销的总和，即 $c(x_1, x_2)+c(x_2, x_3)+\cdots+c(x_{p-1}, x_p)$。给定任何两个节点 x 和 y，通常在这两个节点之间有许多条路径，每条路径都有一个开销。这些路径中的一条或多条是**最低开销路径**

(least-cost path)。因此最低开销路径问题是清楚的：找出源和目的地之间具有最低开销的一条路。例如，在图 5-3 中，源节点 u 和目的节点 w 之间的最低开销路径是（u，x，y，w），具有的路径开销是 3。注意到若在图中的所有边具有相同的开销，则最低开销路径也就是**最短路径**（shortest path），即在源和目的地之间的具有最少链路数量的路径。

作为一个简单练习，试找出图 5-3 中从节点 u 到节点 z 的最低开销路径，并要反映出你是如何算出该路径的。如果你像大多数人一样，通过考察图 5-3，跟踪几条从 u 到 z 的路由，你就能找出从 u 到 z 的路径，然后以某种方式来确信你所选择的路径就是所有可能的路径中具有最低开销的路径。（你考察过 u 到 z 之间的所有 17 条可能的路径吗？很可能没有！）这种计算就是一种集中式路由选择算法的例子，即路由选择算法在一个位置（你的大脑中）运行，该位置具有网络的完整信息。一般而言，路由选择算法的一种分类方式是根据该算法是集中式还是分散式来划分。

- **集中式路由选择算法**（centralized routing algorithm）用完整的、全局性的网络知识计算出从源到目的地之间的最低开销路径。也就是说，该算法以所有节点之间的连通性及所有链路的开销为输入。这就要求该算法在真正开始计算以前，要以某种方式获得这些信息。计算本身可在某个场点（例如，图 5-2 中所示的逻辑集中式控制器）进行，或在每台路由器的路由选择组件中重复进行（例如在图 5-1 中）。然而，这里的主要区别在于，集中式算法具有关于连通性和链路开销方面的完整信息。具有全局状态信息的算法常被称作**链路状态**（Link State，LS）**算法**，因为该算法必须知道网络中每条链路的开销。我们将在 5.2.1 节中学习 LS 算法。
- 在**分散式路由选择算法**（decentralized routing algorithm）中，路由器以迭代、分布式的方式计算出最低开销路径。没有节点拥有关于所有网络链路开销的完整信息。相反，每个节点仅有与其直接相连链路的开销知识即可开始工作。然后，通过迭代计算过程以及与相邻节点的信息交换，一个节点逐渐计算出到达某目的节点或一组目的节点的最低开销路径。我们将在后面的 5.2.2 节学习一个称为**距离向量**（Distance-Vector，DV）**算法**的分散式路由选择算法。之所以叫作 DV 算法，是因为每个节点维护到网络中所有其他节点的开销（距离）估计的向量。这种分散式算法，通过相邻路由器之间的交互式报文交换，也许更为天然地适合那些路由器直接交互的控制平面，就像在图 5-1 中那样。

路由选择算法的第二种广义分类方式是根据算法是静态的还是动态的进行分类。在**静态路由选择算法**（static routing algorithm）中，路由随时间的变化非常缓慢，通常是人工进行调整（如人为手工编辑一条链路开销）。**动态路由选择算法**（dynamic routing algorithm）随着网络流量负载或拓扑发生变化而改变路由选择路径。一个动态算法可周期性地运行或直接响应拓扑或链路开销的变化而运行。虽然动态算法易于对网络的变化做出反应，但也更容易受诸如路由选择循环、路由振荡之类问题的影响。

路由选择算法的第三种分类方式是根据它是负载敏感的还是负载迟钝的进行划分。在**负载敏感算法**（load-sensitive algorithm）中，链路开销会动态地变化以反映出底层链路的当前拥塞水平。如果当前拥塞的一条链路与高开销相联系，则路由选择算法趋向于绕开该拥塞链路来选择路由。而早期的 ARPAnet 路由选择算法就是负载敏感的［McQuillan 1980］，所以遇到了许多难题［Huitema 1998］。当今的因特网路由选择算法（如 RIP、OSPF和 BGP）都是**负载迟钝的**（load-insensitive），因为某条链路的开销不明确地反映其当前（或最近）的拥塞水平。

5.2.1　链路状态路由选择算法

前面讲过，在链路状态算法中，网络拓扑和所有的链路开销都是已知的，也就是说可用作 LS 算法的输入。实践中这是通过让每个节点向网络中所有其他节点广播链路状态分组来完成的，其中每个链路状态分组包含它所连接的链路的标识和开销。在实践中（例如使用因特网的 OSPF 路由选择协议，讨论见 5.3 节），这经常由**链路状态广播**（link state broadcast）算法［Perlman 1999］来完成。节点广播的结果是所有节点都具有该网络的统一、完整的视图。于是每个节点都能够像其他节点一样，运行 LS 算法并计算出相同的最低开销路径集合。

我们下面给出的链路状态路由选择算法叫作 Dijkstra 算法，该算法以其发明者命名。一个密切相关的算法是 Prim 算法，有关图算法的一般性讨论参见［Cormen 2001］。Dijkstra 算法计算从某节点（源节点，我们称之为 u）到网络中所有其他节点的最低开销路径。Dijkstra 算法是迭代算法，其性质是经算法的第 k 次迭代后，可知道到 k 个目的节点的最低开销路径，在到所有目的节点的最低开销路径之中，这 k 条路径具有 k 个最低开销。我们定义下列记号。

- $D(v)$：到算法的本次迭代，从源节点到目的节点 v 的最低开销路径的开销。
- $p(v)$：从源到 v 沿着当前最低开销路径的前一节点（v 的邻居）。
- N'：节点子集；如果从源到 v 的最低开销路径已确知，v 在 N' 中。

该集中式路由选择算法由一个初始化步骤和其后的循环组成。循环执行的次数与网络中节点个数相同。一旦终止，该算法就计算出了从源节点 u 到网络中每个其他节点的最短路径。

源节点 u 的链路状态（LS）算法

```
1  Initialization:
2   N' = {u}
3   for all nodes v
4     if v is a neighbor of u
5       then D(v) = c(u,v)
6     else D(v) = ∞
7
8  Loop
9   find w not in N' such that D(w) is a minimum
10  add w to N'
11  update D(v) for each neighbor v of w and not in N':
12        D(v) = min(D(v), D(w)+ c(w,v) )
13   /* new cost to v is either old cost to v or known
14    least path cost to w plus cost from w to v */
15 until N'= N
```

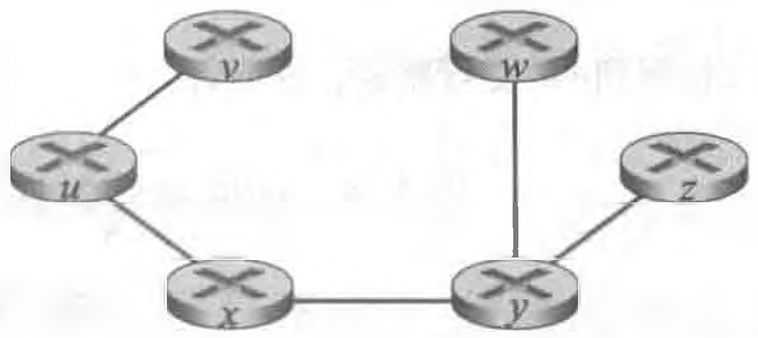
举一个例子，考虑图 5-3 中的网络，计算从 u 到所有可能目的地的最低开销路径。该算法的计算过程以表格方式总结于表 5-1 中，表中的每一行给出了迭代结束时该算法的变量的值。我们详细地考虑前几个步骤。

表 5-1　在图 5-3 中的网络上运行的链路状态算法

步骤	N'	$D(v)$，$p(v)$	$D(w)$，$p(w)$	$D(x)$，$p(x)$	$D(y)$，$p(y)$	$D(z)$，$p(z)$
0	u	2，u	5，u	1，u	∞	∞
1	ux	2，u	4，x		2，x	∞
2	uxy	2，u	3，y			4，y
3	$uxyv$		3，y			4，y
4	$uxyvw$					4，y
5	$uxyvwz$					

- 在初始化步骤，从 u 到与其直接相连的邻居 v、x、w 的当前已知最低开销路径分别初始化为 2、1 和 5。特别值得注意的是，到 w 的开销被设为 5（尽管我们很快就会看见确实存在一条开销更小的路径），因为这是从 u 到 w 的直接（一跳）链路开销。到 y 与 z 的开销被设为无穷大，因为它们不直接与 u 连接。
- 在第一次迭代中，我们观察那些还未加到集合 N' 中的节点，并且找出在前一次迭代结束时具有最低开销的节点。那个节点便是 x，其开销是 1，因此 x 被加到集合 N' 中。于是 LS 算法中的第 12 行中的程序被执行，以更新所有节点 v 的 $D(v)$，产生表 5-1 中第 2 行（步骤 1）所示的结果。到 v 的路径开销未变。经过节点 x 到 w（在初始化结束时其开销为 5）的路径开销被发现为 4。因此这条具有更低开销的路径被选中，且沿从 u 开始的最短路径上 w' 的前一节点被设为 x。类似地，到 y（经过 x）的开销被计算为 2，且该表也被相应地更新。
- 在第二次迭代时，节点 v 与 y 被发现具有最低开销路径（2），并且我们任意改变次序将 y 加到集合 N' 中，使得 N' 中含有 u、x 和 y。到仍不在 N' 中的其余节点（即节点 v、w 和 z）的开销通过 LS 算法中的第 12 行进行更新，产生如表 5-1 中第 3 行所示的结果。
- 如此等等。

当 LS 算法终止时，对于每个节点，我们都得到从源节点沿着它的最低开销路径的前一节点。对于每个前一节点，我们又有它的前一节点，以此方式我们可以构建从源节点到所有目的节点的完整路径。通过对每个目的节点存放从 u 到目的地的最低开销路径上的下一跳节点，在一个节点（如节点 u）中的转发表则能够根据此信息而构建。图 5-4 显示了对于图 5-3 中的网络产生的最低开销路径和 u 中的转发表。



目的地	链路
v	(u, v)
w	(u, x)
x	(u, x)
y	(u, x)
z	(u, x)

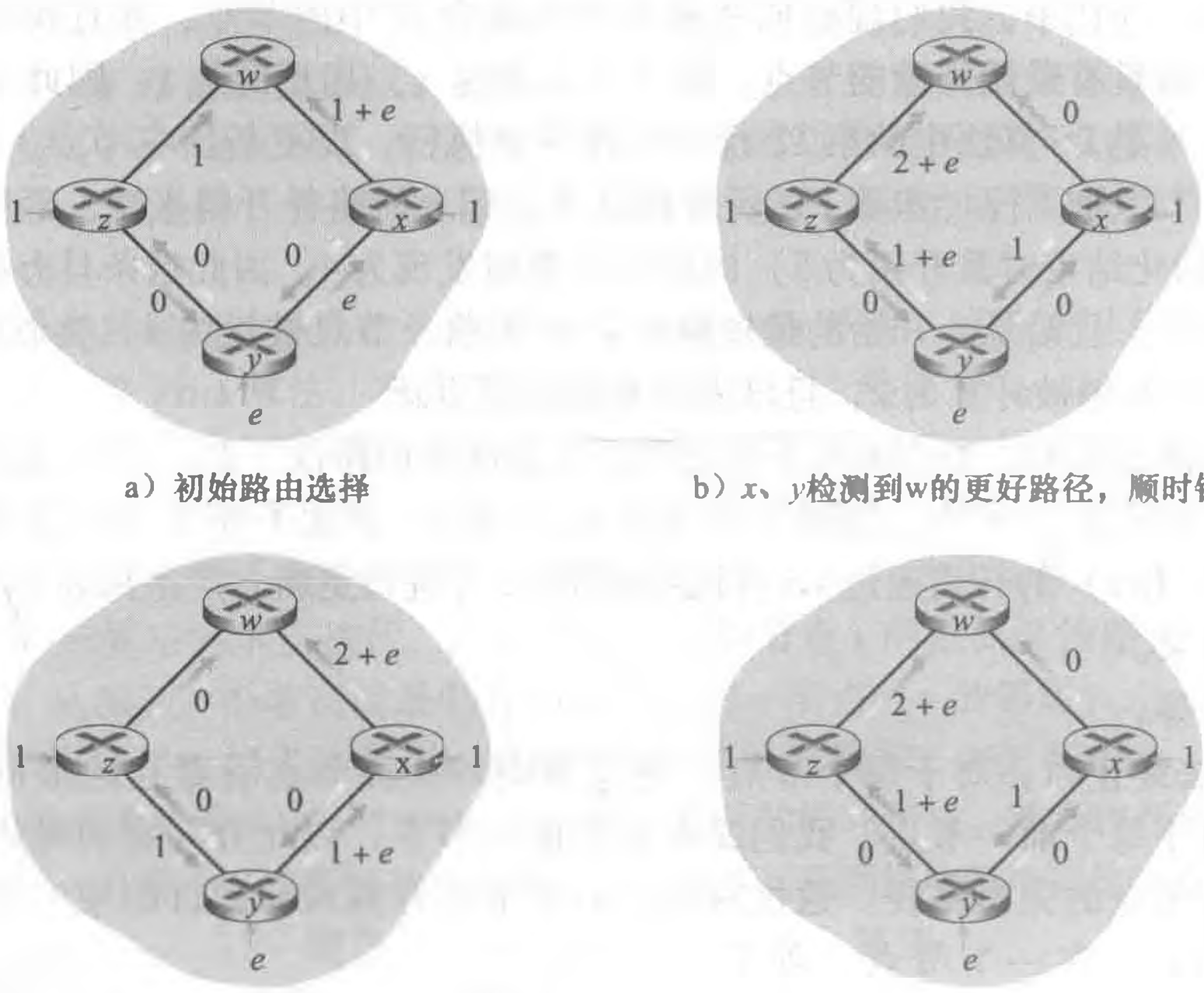
图 5-4　对于节点 u 的最低开销路径和转发表

该算法的计算复杂性是什么？即给定 n 个节点（不算源节点），在最坏情况下要经过多少次计算，才能找到从源节点到所有目的节点的最低开销路径？在第一次迭代中，我们需要搜索所有的 n 个节点以确定出不在 N' 中且具有最低开销的节点 w。在第二次迭代时，我们需要检查 $n-1$ 个节点以确定最低开销。第三次对 $n-2$ 个节点迭代，依次类推。总之，我们在所有迭代中需要搜寻的节点总数为 $n(n+1)/2$，因此我们说前面实现的链路状态算法在最差情况下复杂性为 $O(n^2)$。（该算法的一种更复杂的实现是使用一种称为堆的数据结构，能用对数时间而不是线性时间得到第 9 行的最小值，因而减少其复杂性。）

在完成 LS 算法的讨论之前，我们考虑一下可能出现的问题。图 5-5 显示了一个简单的网络拓扑，图中的链路开销等于链路上承载的负载，例如反映要历经的时延。在该例中，链路开销是非对称的，即仅当在链路（u，v）两个方向所承载的负载相同时 $c(u, v)$ 与 $c(v, u)$ 才相等。在该例中，节点 z 产生发往 w 的一个单元的流量，节点 x 也产生发往 w 的一个单元的流量，并且节点 y 也产生发往 w 的一个数量为 e 的流量。初始路由选择情况如图 5-5a 所示，其链路开销对应于承载的流量。

当 LS 算法再次运行时，节点 y 确定（基于图 5-5a 所示的链路开销）顺时针到 w 的路径开销为 1，而逆时针到 w 的路径开销（一直使用的）是 $1+e$。因此 y 到 w 的最低开销路

径现在是顺时针的。类似地，x 确定其到 w 的新的最低开销路径也是顺时针的，产生如图 5-5b 中所示的开销。当 LS 算法下次运行时，节点 x、y 和 z 都检测到一条至 w 的逆时针方向零开销路径，它们都将其流量引导到逆时针方向的路由上。下次 LS 算法运行时，x、y 和 z 都将其流量引导到顺时针方向的路由上。



a）初始路由选择

b）x、y检测到w的更好路径，顺时针

c）x、y、z检测到w的更好路径，逆时针

d）x、y、z检测到w的更好路径，顺时针

图 5-5　拥塞敏感的路由选择的振荡

如何才能防止这样的振荡（它不只是出现在链路状态算法中，而且也可能出现在任何使用拥塞或基于时延的链路测度的算法中）。一种解决方案可能强制链路开销不依赖于所承载的流量，但那是一种不可接受的解决方案，因为路由选择的目标之一就是要避开高度拥塞（如高时延）的链路。另一种解决方案就是确保并非所有的路由器都同时运行 LS 算法。这似乎是一个更合理的方案，因为我们希望即使路由器以相同周期运行 LS 算法，在每个节点上算法执行的时机也将是不同的。有趣的是，研究人员近来已注意到了因特网上的路由器能在它们之间进行自同步［Floyd Synchronization 1994］。这就是说，即使它们初始时以同一周期但在不同时刻执行算法，算法执行时机最终会在路由器上变为同步并保持之。避免这种自同步的一种方法是，让每台路由器发送链路通告的时间随机化。

学习过 LS 算法之后，我们接下来考虑目前在实践中使用的其他重要的路由选择算法，即距离向量路由选择算法。

5.2.2　距离向量路由选择算法

距离向量（Distance-Vector，DV）算法是一种迭代的、异步的和分布式的算法，而 LS 算法是一种使用全局信息的算法。说它是分布式的，是因为每个节点都要从一个或多个直接相连邻居接收某些信息，执行计算，然后将其计算结果分发给邻居。说它是迭代的，是

因为此过程一直要持续到邻居之间无更多信息要交换为止。（有趣的是，此算法是自我终止的，即没有计算应该停止的信号，它就停止了。）说它是异步的，是因为它不要求所有节点相互之间步伐一致地操作。我们将看到一个异步的、迭代的、自我终止的、分布式的算法比一个集中式的算法要有趣得多！

在我们给出 DV 算法之前，有必要讨论一下存在于最低开销路径的开销之间的一种重要关系。令 $d_x(y)$ 是从节点 x 到节点 y 的最低开销路径的开销。则该最低开销与著名的 Bellman-Ford 方程相关，即

$$d_x(y) = \min_v \{c(x,v) + d_v(y)\} \tag{5-1}$$

方程中的 $\min_v$ 是对于 x 的所有邻居的。Bellman-Ford 方程是相当直观的。实际上，从 x 到 v 遍历之后，如果我们接下来取从 v 到 y 的最低开销路径，则该路径开销将是 $c(x, v) + d_v(y)$。因此我们必须通过遍历某些邻居 v 开始，从 x 到 y 的最低开销是对所有邻居 v 的 $c(x, v) + d_v(y)$ 的最小值。

但是对于那些可能怀疑该方程正确性的人，我们核查在图 5-3 中的源节点 u 和目的节点 z。源节点 u 有 3 个邻居：节点 v、x 和 w。通过遍历该图中的各条路径，容易看出 $d_v(z) = 5$、$d_x(z) = 3$ 和 $d_w(z) = 3$。将这些值连同开销 $c(u, v) = 2$、$c(u, x) = 1$ 和 $c(u, w) = 5$ 代入方程（5-1），得出 $d_u(z) = \min\{2 + 5, 5 + 3, 1 + 3\} = 4$，这显然是正确的，并且对同一个网络来说，这正是 Dijkstra 算法为我们提供的结果。这种快速验证应当有助于消除你可能具有的任何怀疑。

Bellman-Ford 方程不止是一种智力上的珍品，它实际上具有重大的实践重要性。特别是对 Bellman-Ford 方程的解为节点 x 的转发表提供了表项。为了理解这一点，令 v^* 是取得方程（5-1）中最小值的任何相邻节点。接下来，如果节点 x 要沿着最低开销路径向节点 y 发送一个分组，它应当首先向节点 v^* 转发该分组。因此，节点 x 的转发表将指定节点 v^* 作为最终目的地 y 的下一跳路由器。Bellman-Ford 方程的另一个重要实际贡献是，它提出了将在 DV 算法中发生的邻居到邻居通信的形式。

其基本思想如下。每个节点 x 以 $D_x(y)$ 开始，对在 N 中的所有节点 y，估计从 x 到 y 的最低开销路径的开销。令 $\boldsymbol{D}_x = [D_x(y): y \in N]$ 是节点 x 的距离向量，该向量是从 x 到在 N 中的所有其他节点 y 的开销估计向量。使用 DV 算法，每个节点 x 维护下列路由选择信息：

- 对于每个邻居 v，从 x 到直接相连邻居 v 的开销为 $c(x, v)$。
- 节点 x 的距离向量，即 $\boldsymbol{D}_x = [D_x(y): y \in N]$，包含了 x 到 N 中所有目的地 y 的开销估计值。
- 它的每个邻居的距离向量，即对 x 的每个邻居 v，有 $\boldsymbol{D}_v = [D_v(y): y \in N]$。

在该分布式、异步算法中，每个节点不时地向它的每个邻居发送它的距离向量副本。当节点 x 从它的任何一个邻居 v 接收到一个新距离向量，它保存 v 的距离向量，然后使用 Bellman-Ford 方程更新它自己的距离向量如下：

$$D_x(y) = \min_v \{c(x,v) + D_v(y)\} \quad \text{对 } N \text{ 中的每个节点}$$

如果节点 x 的距离向量因这个更新步骤而改变，节点 x 接下来将向它的每个邻居发送其更新后的距离向量，这继而让所有邻居更新它们自己的距离向量。令人惊奇的是，只要所有的节点继续以异步方式交换它们的距离向量，每个开销估计 $D_x(y)$ 收敛到 $d_x(y)$，$d_x(y)$ 为从节点 x 到节点 y 的实际最低开销路径的开销［Bersekas 1991］！

距离向量（DV）算法

在每个节点 x：

```
1   Initialization:
2     for all destinations y in N:
3        D_x(y)= c(x,y)/* if y is not a neighbor then c(x,y)= ∞ */
4     for each neighbor w
5        D_w(y) = ? for all destinations y in N
6     for each neighbor w
7        send distance vector  D_x = [D_x(y): y in N] to w
8
9   loop
10     wait  (until I see a link cost change to some neighbor w or
11              until I receive a distance vector from some neighbor w)
12
13     for each y in N:
14          D_x(y) = min_v{c(x,v) + D_v(y)}
15
16  if Dx(y) changed for any destination y
17        send distance vector D_x  = [D_x(y): y in N] to all neighbors
18
19  forever
```

在该 DV 算法中，当节点 x 发现它的直接相连的链路开销变化或从某个邻居接收到一个距离向量的更新时，它就更新其距离向量估计值。但是为了一个给定的目的地 y 而更新它的转发表，节点 x 真正需要知道的不是到 y 的最短路径距离，而是沿着最短路径到 y 的下一跳路由器邻居节点 $v^*(y)$。如你可能期望的那样，下一跳路由器 $v^*(y)$ 是在 DV 算法第 14 行中取得最小值的邻居 v。（如果有多个取得最小值的邻居 v，则 $v^*(y)$ 能够是其中任何一个有最小值的邻居。）因此，对于每个目的地 y，在第 13～14 行中，节点 x 也决定 $v^*(y)$ 并更新它对目的地 y 的转发表。

前面讲过 LS 算法是一种全局算法，在于它要求每个节点在运行 Dijkstra 算法之前，首先获得该网络的完整信息。DV 算法是分布式的，它不使用这样的全局信息。实际上，节点具有的唯一信息是它到直接相连邻居的链路开销和它从这些邻居接收到的信息。每个节点等待来自任何邻居的更新（第 10～11 行），当接收到一个更新时计算它的新距离向量（第 14 行）并向它的邻居分布其新距离向量（第 16～17 行）。在实践中许多类似 DV 的算法被用于多种路由选择协议中，包括因特网的 RIP 和 BGP、ISO IDRP、Novell IPX 和早期的 ARPAnet。

图 5-6 举例说明了 DV 算法的运行，应用场合是该图顶部有三个节点的简单网络。算法的运行以同步的方式显示出来，其中所有节点同时从其邻居接收报文，计算其新距离向量，如果距离向量发生了变化则通知其邻居。学习完这个例子后，你应当确信该算法以异步方式也能正确运行，异步方式中可在任意时刻出现节点计算与更新的产生/接收。

该图最左边一列显示了这 3 个节点各自的初始**路由选择表**（routing table）。例如，位于左上角的表是节点 x 的初始路由选择表。在一张特定的路由选择表中，每行是一个距离向量——特别是每个节点的路由选择表包括了它的距离向量和它的每个邻居的距离向量。因此，在节点 x 的初始路由选择表中的第一行是 $\boldsymbol{D}_x = [D_x(x), D_x(y), D_x(z)] = [0, 2, 7]$。在该表的第二和第三行是最近分别从节点 y 和 z 收到的距离向量。因为在初始化时节点 x 还没有从节点 y 和 z 收到任何东西，所以第二行和第三行表项中被初始化为无穷大。

初始化后，每个节点向它的两个邻居发送其距离向量。图 5-6 中用从表的第一列到表的第二列的箭头说明了这一情况。例如，节点 x 向两个节点 y 和 z 发送了它的距离向量

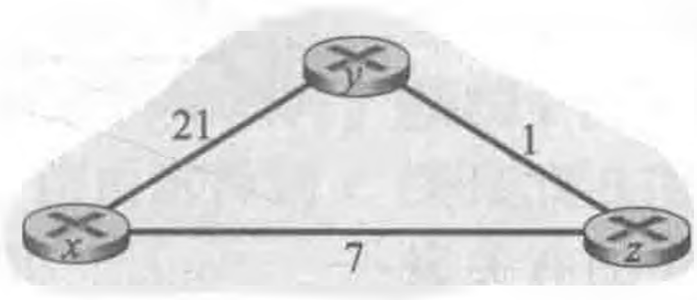
$D_x=[0, 2, 7]$。在接收到该更新后，每个节点重新计算它自己的距离向量。例如，节点 x 计算

$$D_x(x)=0$$

$$D_x(y)=\min\{c(x,y)+D_y(y),c(x,z)+D_z(y)\}=\min\{2+0,7+1\}=2$$

$$D_x(z)=\min\{c(x,y)+D_y(z),c(x,z)+D_z(z)\}=\min\{2+1,7+0\}=3$$

第二列因此为每个节点显示了节点的新距离向量连同刚从它的邻居接收到的距离向量。注意到，例如节点 x 到节点 z 的最低开销估计 $D_x(z)$ 已经从7变成了3。还应注意到，对于节点 x，节点 y 在该 DV 算法的第 14 行中取得了最小值；因此在该算法的这个阶段，我们在节点 x 得到了 $v^*(y)=y$ 和 $v^*(z)=y$。



节点x的表

到…开销	x	y	z
x	0	2	7
从 y	∞	∞	∞
z	∞	∞	∞

到…开销	x	y	z
x	0	2	3
从 y	2	0	1
z	7	1	0

到…开销	x	y	z
x	0	2	3
从 y	2	0	1
z	3	1	0

节点y的表

到…开销	x	y	z
x	∞	∞	∞
从 y	2	0	1
z	∞	∞	∞

到…开销	x	y	z
x	0	2	7
从 y	2	0	1
z	7	1	0

到…开销	x	y	z
x	0	2	3
从 y	2	0	1
z	3	1	0

节点z的表

到…开销	x	y	z
x	∞	∞	∞
从 y	∞	∞	∞
z	7	1	0

到…开销	x	y	z
x	0	2	7
从 y	2	0	1
z	3	1	0

到…开销	x	y	z
x	0	2	3
从 y	2	0	1
z	3	1	0

时间

图 5-6　DV 算法

在节点重新计算它们的距离向量之后，它们再次向其邻居发送它们的更新距离向量（如果它们已经改变的话）。图 5-6 中由从表第二列到表第三列的箭头说明了这一情况。注意到仅有节点 x 和节点 z 发送了更新：节点 y 的距离向量没有发生变化，因此节点 y 没有发送更新。在接收到这些更新后，这些节点则重新计算它们的距离向量并更新它们的路由选择表，这些显示在第三列中。

从邻居接收更新距离向量、重新计算路由选择表项和通知邻居到目的地的最低开销路径的开销已经变化的过程继续下去，直到无更新报文发送为止。在这个时候，因为无更新报文发送，将不会出现进一步的路由选择表计算，该算法将进入静止状态，即所有的节点将执行 DV 算法的第 10～11 行中的等待。该算法停留在静止状态，直到一条链路开销发生改变，如下面所讨论的那样。

1. 距离向量算法：链路开销改变与链路故障

当一个运行 DV 算法的节点检测到从它自己到邻居的链路开销发生变化时（第 10～11 行），它就更新其距离向量（第 13～14 行），并且如果最低开销路径的开销发生了变化，向邻居通知其新的距离向量（第 16～17 行）。图 5-7a 图示了从 y 到 x 的链路开销从 4 变为 1 的情况。我们在此只关注 y 与 z 到目的地 x 的距离表中的有关表项。该 DV 算法导致下列事件序列的出现：

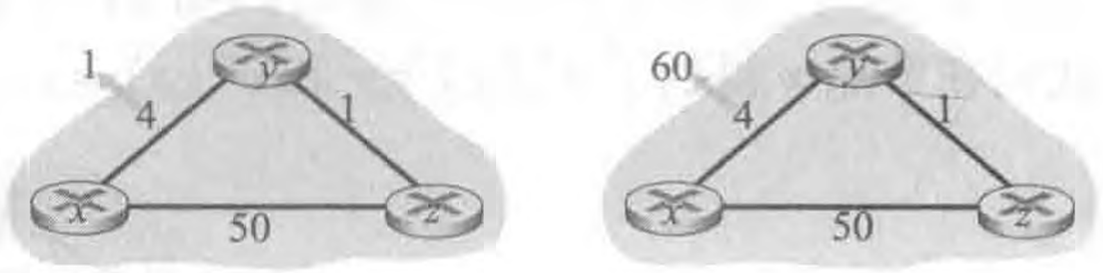


图 5-7　链路开销改变

- 在 t_0 时刻，y 检测到链路开销变化（开销从 4 变为 1），更新其距离向量，并通知其邻居这个变化，因为最低开销路径的开销已改变。
- 在 t_1 时刻，z 收到来自 y 的更新报文并更新了其距离表。它计算出到 x 的新最低开销（从开销 5 减为开销 2），它向其邻居发送了它的新距离向量。
- 在 t_2 时刻，y 收到来自 z 的更新并更新其距离表。y 的最低开销未变，因此 y 不发送任何报文给 z。该算法进入静止状态。

因此，对于该 DV 算法只需两次迭代就到达了静止状态。在 x 与 y 之间开销减少的好消息通过网络得到了迅速传播。

我们现在考虑一下当某链路开销增加时发生的情况。假设 x 与 y 之间的链路开销从 4 增加到 60，如图 5-7b 所示。

1）在链路开销变化之前，$D_y(x)=4$，$D_y(z)=1$，$D_z(y)=1$ 和 $D_z(x)=5$。在 t_0 时刻，y 检测到链路开销变化（开销从 4 变为 60）。y 计算它到 x 的新的最低开销路径的开销，其值为

$$D_y(x) = \min\{c(y,x) + D_x(x), c(y,z) + D_z(x)\} = \min\{60+0, 1+5\} = 6$$

当然，从网络全局的视角来看，我们能够看出经过 z 的这个新开销是错误的。但节点 y 仅有的信息是：它到 x 的直接开销是 60，且 z 上次已告诉 y，z 能以开销 5 到 x。因此，为了到达 x，y 将通过 z 路由，完全期望 z 能以开销 5 到达 x。到了 t_1 时刻，我们遇到**路由选择环路**（routing loop），即为到达 x，y 通过 z 路由，z 又通过 y 路由。路由选择环路就像一个黑洞，即目的地为 x 的分组在 t_1 时刻到达 y 或 z 后，将在这两个节点之间不停地（或直到转发表发生改变为止）来回反复。

2）因为节点 y 已算出到 x 的新的最低开销，它在 t_1 时刻将该新距离向量通知 z。

3）在 t_1 后某个时间，z 收到 y 的新距离向量，它指示了 y 到 x 的最低开销是 6。z 知道它能以开销 1 到达 y，因此计算出到 x 的新最低开销 $D_z(x)=\min\{50+0, 1+6\}=7$。因为 z 到 x 的最低开销已增加了，于是它便在 t_2 时刻通知 y 其新开销。

4）以类似方式，在收到 z 的新距离向量后，y 决定 $D_y(x)=8$ 并向 z 发送其距离向量。接下来 z 确定 $D_z(x)=9$ 并向 y 发送其距离向量，等等。