第5章 网络层：控制平面

目的:

- 了解网络层服务背后的原则，专注于控制平面：
  - **传统路由算法**
  - ICMP协议
  - 网络管理
- 因特网的网络层的实现
  - OSPF、BGP、ICMP、SNMP
- SDN

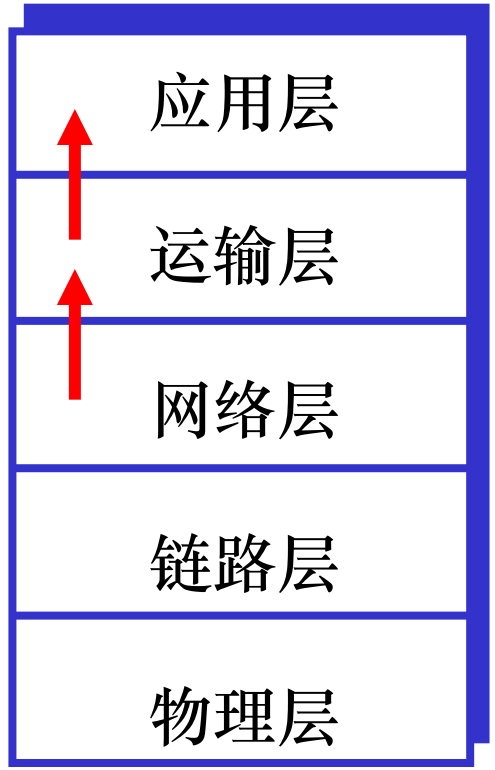
第5章 主要内容

- 5.1 概述
- 5.2 路由选择算法
  - 链路状态路由选择算法
  - 距离向量路由选择算法
- 5.3 ISP域内路由选择：OSPF
- 5.4 ISP域间路由选择：BGP
  - BGP的作用
  - 通告BGP路由信息
  - 确定最好的路由
  - IP任播
  - 路由选择策略
- 5.5 SDN控制平面
- 5.6 ICMP协议
- 5.7 网络管理和SNMP
  - 网络管理框架
  - 简单网络管理协议

5.1 概述

- 运输层和网络层提供服务的区别
  - 运输层为主机上的两个进程之间提供通信服务；
  - 运输层只在网络的主机中出现；
  - 网络层提供的主机到主机的通信服务；
  - 网络层在网络的主机和路由器中出现。

本章主要讨论网络层如何实现主机到主机的通信服务



网络层功能

回顾：网络层的两个功能

- *转发：*转发分组从路由器的输入端口到正确的输出端口

data plane

- *路由：*为分组确定从源端到目的端的路径

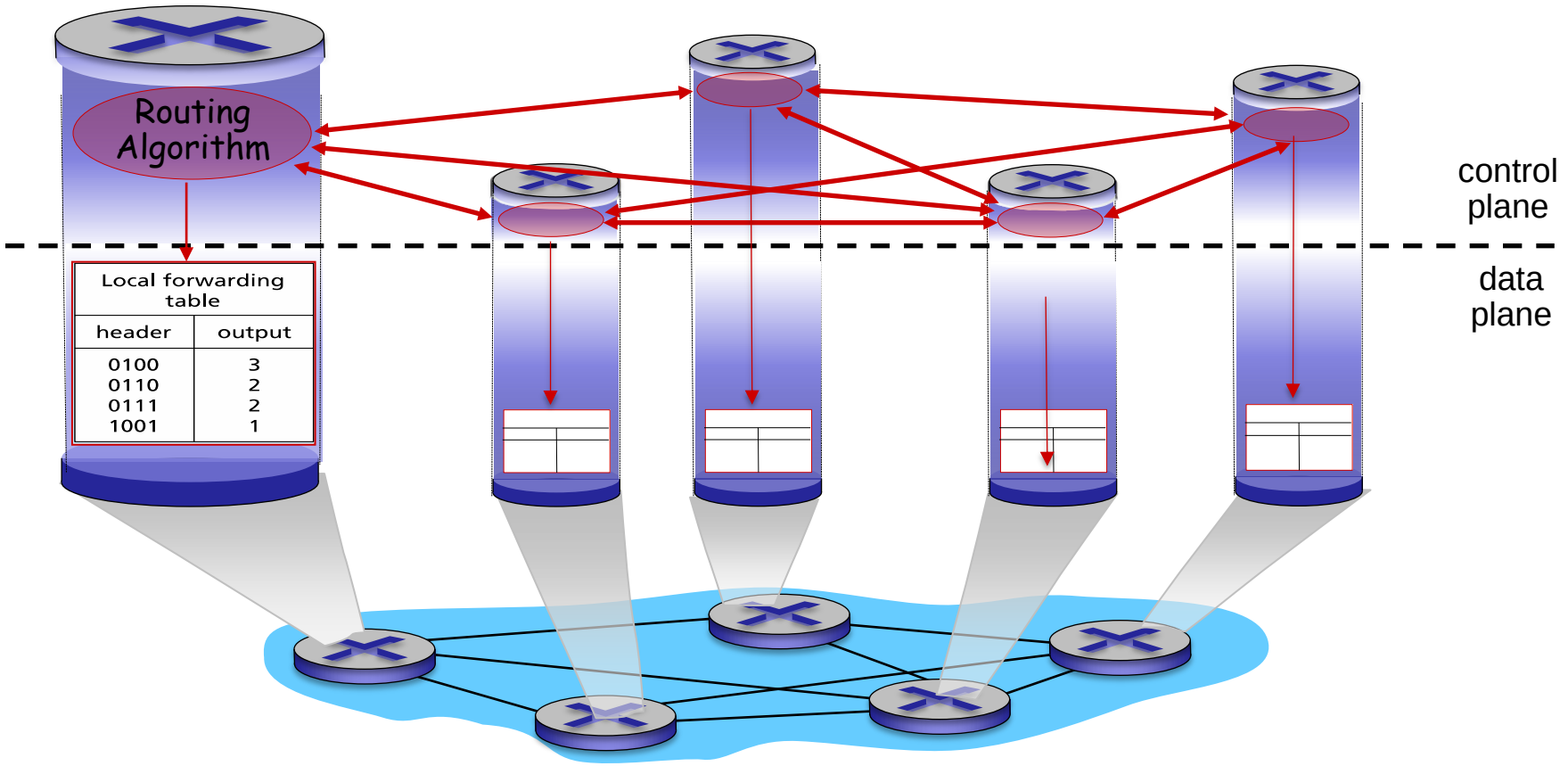
control plane

有两种方法构建网络控制平台：

- 在每个路由器上控制(传统方法)
- 在逻辑上集中地控制(SDN)

在每个路由器上实现控制平台

每个路由器中的各个路由组件在控制平面中相互作用，以计算转发表。



5.2 路由选择算法

- 分组通过路由器转发，每个路由器都有一张转发表，路由选择算法决定转发表内容。
- 路由选择：确定分组从发送方传送到接收方所要通过的路径(路由)。
  - 数据报服务：在给定源和目的地之间传输不同分组可能采用不同的路由。
  - 虚电路服务：在给定源和目的地之间的所有分组采用同一路径。

基本术语

- 默认路由器：与主机直接相连的一台路由器，又叫**第一跳路由器**。
- 每当主机发送一个到局域网以外的分组时，需要将分组传送给它的默认路由器。
  - 源路由器：源主机的默认路由器。
  - 目的路由器：目的主机的默认路由器。
- 从源主机到目的主机的选路归结为从源路由器到目的路由器的选路。

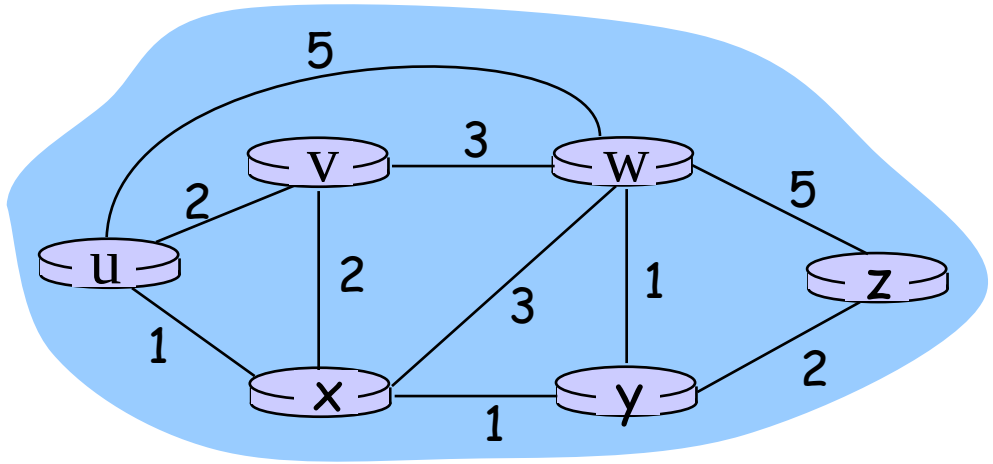
选路算法

- 确定一个分组从源路由器到目的路由器所经路径的算法。
- 即在给定的一组路由器以及连接路由器的链路中，找到一条从源路由器到目的路由器的“好”路径。
- “好”路径：具有“最低费用”的路径。
  - 由许多因素决定，如链路长度、传播时延、价格等。

网络的抽象图模型

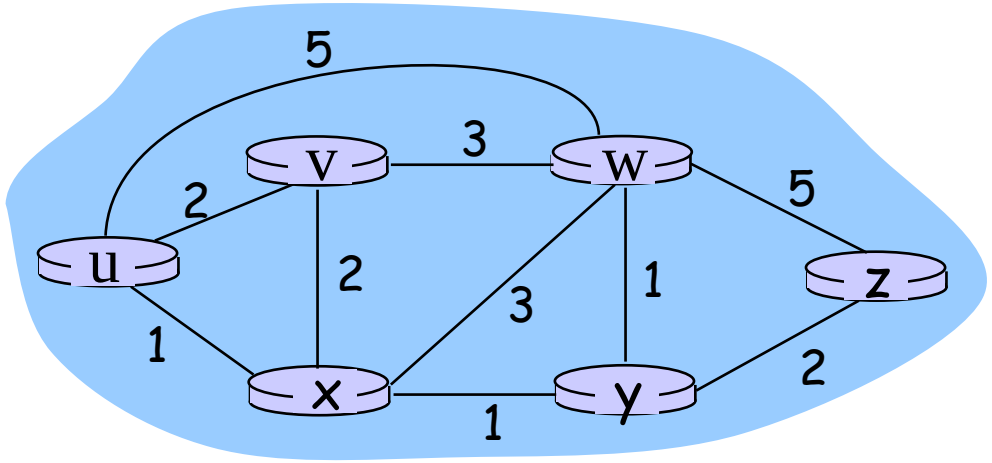
- 用节点图表示网络的结构。
- 图G = (N，E)表示N个节点和E条边的集合，每条边是来自N的一对节点。
  - 节点：表示路由器(做出分组转发判决的点)。
    节点集合{u，v，w，x，y，z}
  - 边：连接节点的线段，表示路由器之间的物理链路。

边的集合
{(u,v), (u,x), (v,x), (v,w), (x,w), (x,y),
(w,y), (w,z), (y,z)}



边的权值(费用)

- 边的权值表示对应链路的物理长度、或链路速度、或与链路相关的费用。
  - c(x, y)表示节点x和节点y之间边的费用
- 若节点x与节点y直接相连，则c(x, y) = 链路费用
  - 如c(u, v) = 2，节点u与节点v互为邻居
- 若节点x与节点y不直接相连，则c(x, y) = ∞
  - 如c(u, y) = ∞，c(u, z) = ∞
- c(x, y) = c(y, x)，如c(u, v) = c(v, u) = 2



选路算法：根据给定的网络抽象图，找出从源节点到目的节点间的最低费用路径。

路径

- 路径表示：用路径上的节点来描述
  - 如图G = (N, E)中的一条路径，是一个节点序列$(x_1,x_2,\cdots,x_p)$，其中每个相邻节点$x_1,x_2,\cdots,x_p$依次连成边
- 路径$(x_1,x_2,\cdots,x_p)$费用：沿途所有边的费用之和。即

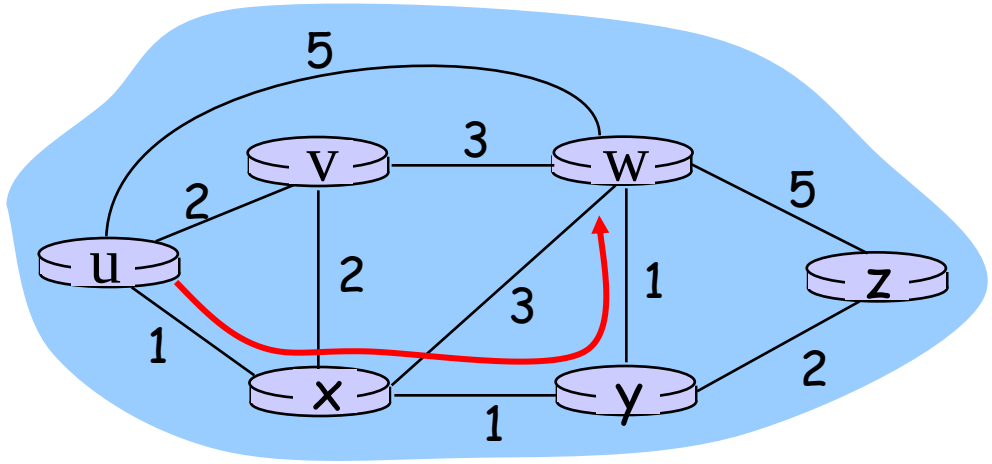
$$c(x_1,x_2) + c(x_2,x_3) + \ldots + c(x_{p-1},x_p)$$

  - 通常任意两个节点x和y 之间有多条路径，费用不一定相同
- 最低费用路径：该路径上链路费用之和最小。若所有链路的费用相同，则最低费用路径就是最短路径，即在源和目的地之间经过链路最少的路径。

例子

如图，源节点*u*和目的节点*w*之间的最低费用路径：

(*u*, *x*, *y*, *w*)，费用为3。



路由算法分类

- 根据算法是全局性(global)还是分布性(decentralized)划分
  - 全局算法：用完整的、全局性的网络信息来计算最低费用路径
    - 所有路由器拥有完整的拓扑和链路费用信息；
    - 链路状态路由算法(link state)；
  - 分布式算法：以迭代的、分布式的方式计算最低费用路径
    - 节点只知道与其直接相连的链路的路由器信息和链路的费用信息，不需拥有所有网络链路费用的完整信息；
    - 通过迭代计算，并与相邻节点(邻居节点)交换信息；
    - 逐步计算出到达某目的节点或一组目的节点的最低费用路径；

距离向量算法DV：每个节点维护到网络中所有其他节点的费用(距离)的估计向量。

路由算法分类

- 根据算法是静态(static)还是动态(dynamic)划分
  - 静态算法：路由确定后基本不再变化，当人工干预调整时，可能有一些变化；
  - 动态算法：当网络的流量负载或拓扑发生变化时，改变路径。可以周期性地或直接地响应拓扑或链路费用的变化。易受选路循环、路由振荡之类问题的影响。
- 因特网常用的两种选路算法
  - 动态全局链路状态算法
  - 动态分散式距离向量算法

5.2.1 链路状态选路算法

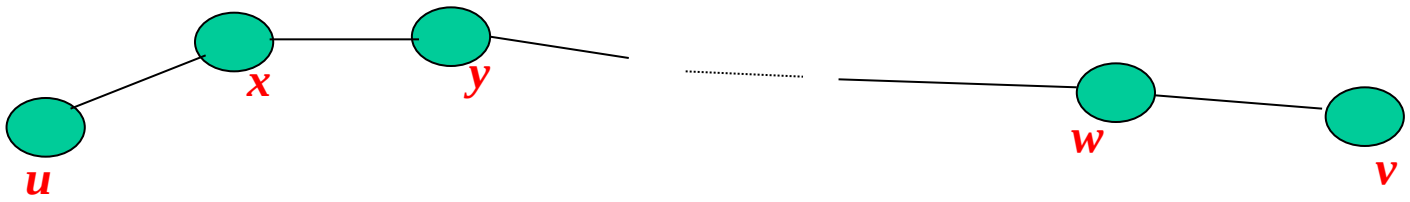
- 前提条件：已知网络拓扑和所有链路的费用，作为算法的输入；
- 获取方法：每个节点向网络中广播链路状态分组(含有它所连接的链路的费用)，由链路状态广播算法实现，最终使所有节点都有一个相同且完整的网络视图；
- 每个节点都可以运行链路状态算法并计算出最低费用路径集。

Dijkstra算法

- 计算从某节点(源节点，如u)到网络中所有其他节点的最低费用路径。
- Dijkstra算法是一种迭代算法，即经第k次迭代后，可知道到k个目的节点的最低费用路径。
- 基本思想：以源节点为起点，每次找出一个到源节点费用最低的节点，直到把所有目的节点都找到为止。

符号定义

- c(x, y)：从节点x到y的链路费用;如果不是直接邻居 = ∞
- D(v)：从源节点u到目的节点*v*的当前费用；
- p(v)：从源节点u到节点*v*的路径上的前一节点(节点*v*的邻居)；
- N‘：节点集合，从源节点到这些节点的最低费用路径已被求出。



Dijkstra算法

```
1   初始化:
2    N' = {u}
3    对所有节点v
4      if v 临近 u
5          then D(v) = c(u,v)
6      else D(v) = ∞
7
8   Loop
9     找出w不在N‘中使得D(w)最小
10    将w加入N'
11    对于所有v临近w并不在N’中，更新D(v):
12       D(v) = min( D(v), D(w) + c(w,v) )
13    /* 到v的新费用或是到v的老费用或到w加上从w到v的已知最短路费用*/
15  until 所有节点在 N‘中
```

算法描述

（1）初始化（1#～6#）

N' ={**源节点*u***};

对所有不在N'中的节点v，标出其$D(v)$值：

- 节点v与源节点u直接相连，$D(v) = c(u, v)$
- 节点v与源节点u不直接相连，$D(v) = \infty$

```
1   初始化：
2     N' = {u}
3     对所有节点v
4       if v 临近 u
5           then D(v) = c(u,v)
6       else D(v) = ∞
```

（2）找出一个到源节点的费用最低的节点*w*，并以此更新其它点D(*v*) 值；

- 从不在*N'* 中的节点中找出一个*D(w)*值最小的节点*w*，并将*w*加入到*N'* 中。(9#～#10)
- 对不在*N'* 中，但与节点*w*是邻居的节点*v*，用新的值更新

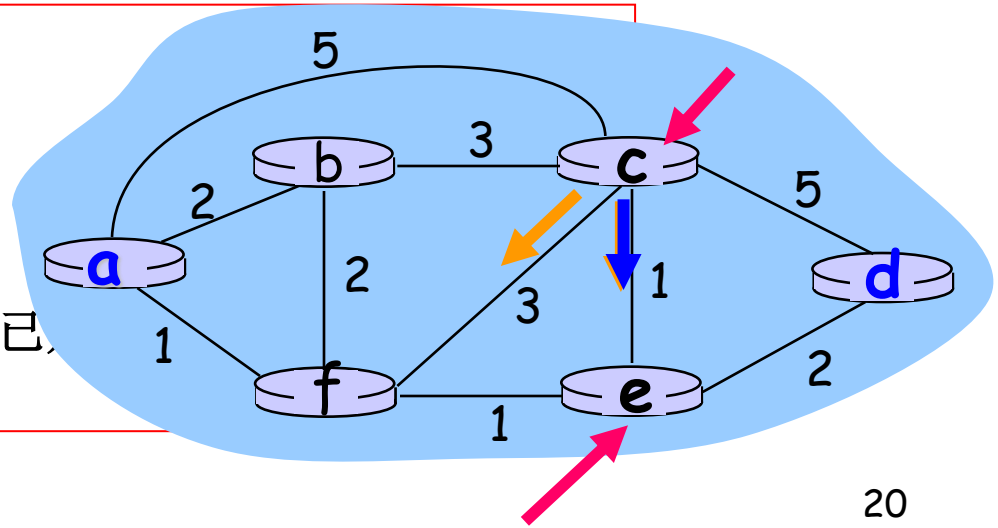
$$D(v)=min[D(v),\ D(w)+c(w,v)]$$

将节点*v*原值与节点*v*现经节点*w*到源节点的值比较，取小值。

（3）重复步骤(2)

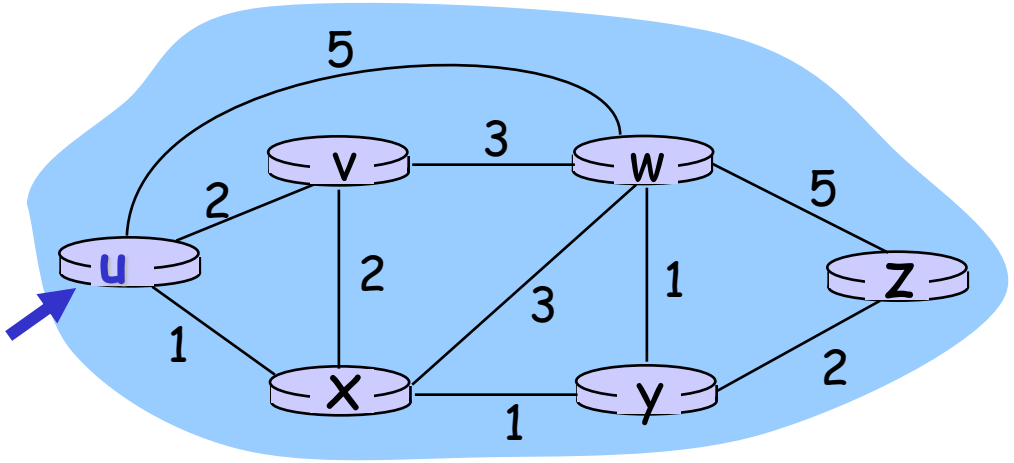
直到所有的网络节点都在*N'* 中为止。

```
8   Loop
9     找出w不在N'中使得D(w)最小
10    将w加入N'
11    对于所有v临近w并不在N'中，更新D(v):
12       D(v) = min( D(v), D(w) + c(w,v) )
13    /* 到v的新费用或是到v的老费用或到w加上从w到v的已
15  until 所有节点在 N'中
```



例子

- 计算从*u*到所有可能目的节点的最低费用路径。
- 计算过程如表，表中的每一行表示一次迭代结束时的算法变量值。

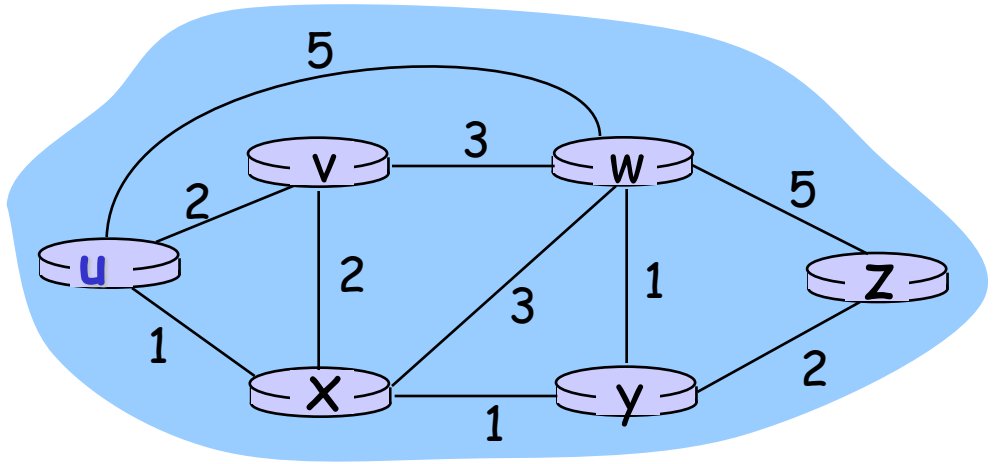


例子

D(w): min(5，1+3)=4

D(y): min(∞，1+1)=2

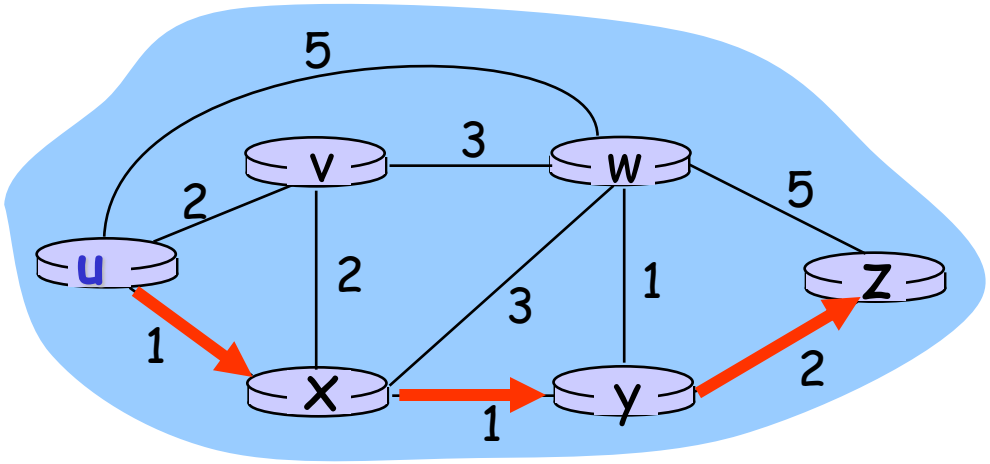
步骤	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	**1,u**	∞	∞
1	u**x**	2,u	**4,x**		**2,x**	∞
2	ux**y**	**2,u**	3,y			4,y
3	uxy**v**		**3,y**			4,y
4	uxyv**w**					4,y
5	uxyvw**z**					



构建从源节点到所有目的节点的最短路径树

- 对于每个节点，都得到从源节点沿着它的最低费用路径的前一节点；
- 每个前一节点，又可得到它的前一节点；
- 以此继续，可以得到到所有目的节点的完整路径。

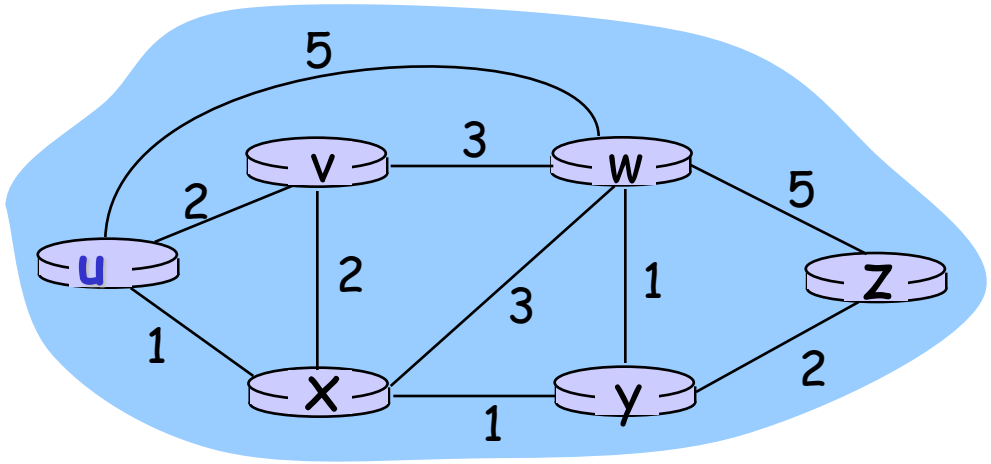
  如节点z的前一节点依次为：$z \rightarrow y \rightarrow x \rightarrow u$
- 得出从源节点u到节点z的最低费用路径为：$uxyz$，费用为4。

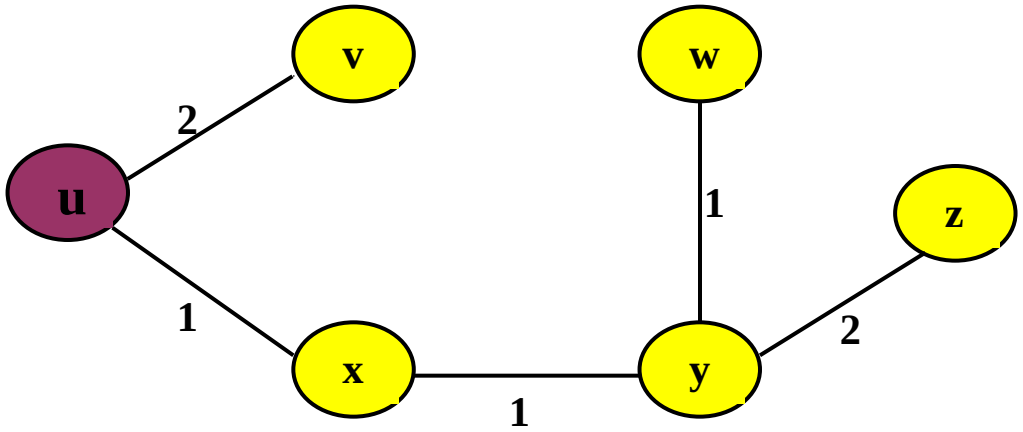


根据目的节点的找出顺序和其费用以及前一节点，可以画出源节点u到所有目的节点的*最低费用路径树*。

根据得到的所有目的节点的完整路径，或最低费用路径树，可以生成源节点的转发表。

- 转发表：存放从源节点到每个目的节点的最低费用路径上的下一跳节点。即指出对于发往某个目的节点的分组，从该节点发出后的下一个节点。





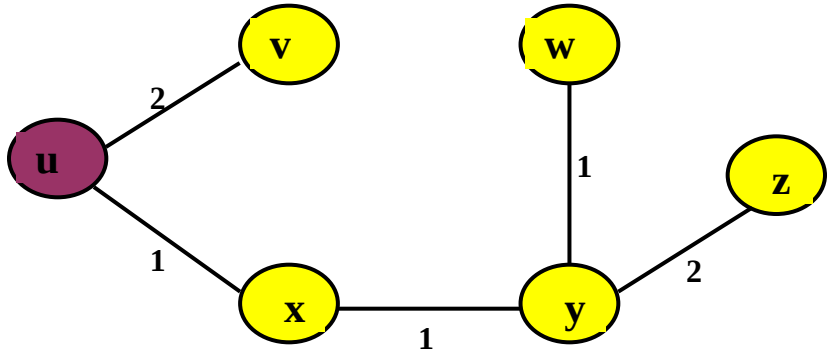
- 默认路由 * ：
  - 表示所有具有相同“下一跳”的表项。即将“下一跳”相同的项合并为一项，目的节点用“*”表示。
  - 优先级最低，转发分组时，当找不到对应表项时，才使用默认路由。

u的转发表

目的点	下一跳
u	—
v	*v*
w	*x*
x	*x*
y	*x*
z	*x*

u的转发表

目的点	下一跳
u	—
v	*v*
*	*x*



将网络中每个节点作为源点，分别用上述方法计算，可以得到每一个节点的转发表

算法的计算复杂性

- 设n个节点(除源节点)，最坏情况下要经多少次计算才能找到从源节点到所有目的节点的最低费用路径?

第1次迭代： 搜索所有的n个节点以确定最低费用节点

第2次迭代： 检查n-1个节点；

第3次迭代： 检查n-2个节点；

第n次迭代：检查1个节点；

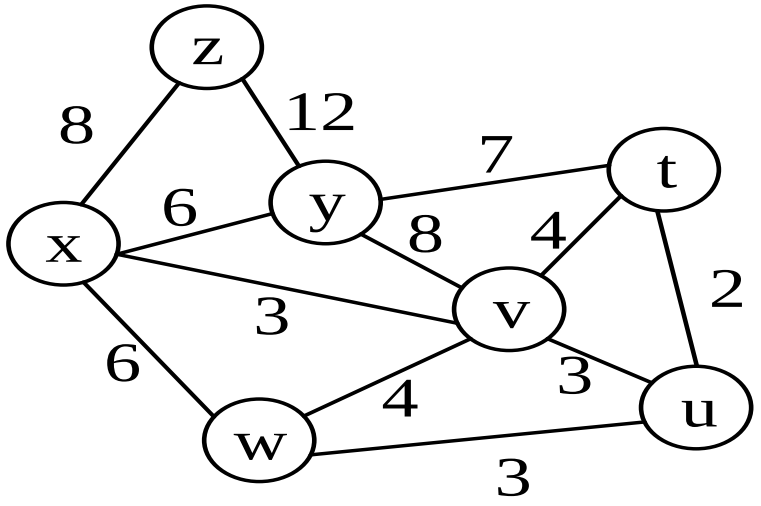
依次类推。

所有迭代中需要搜寻的节点总数为$n(n+1)/2$。

- 算法复杂性为n平方阶序$O(n^2)$。

课堂练习

- 考虑下面的网络。对于标明的链路费用，用Dijkstra的最短路径算法计算从x到所有网络节点的最短路径。通过完成下表，说明算法是如何工作的。



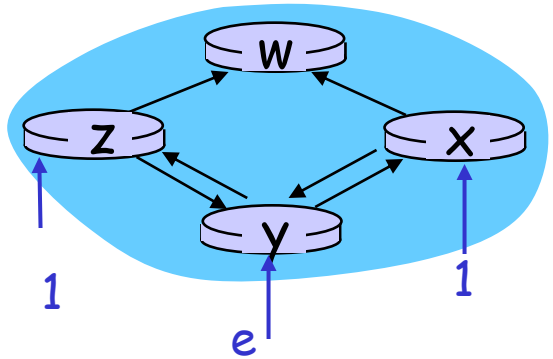
Step	N'	D(t),p(t)	D(u),p(u)	D(v),p(v)	D(w),p(w)	D(y),p(y)	D(z),p(z)
0	x	∞	∞	3,x	6,x	6,x	8,x
1	xv	7,v	6,v	3,x	6,x	6,x	8,x
2	xvu	7,v	6,v	3,x	6,x	6,x	8,x
3	xvuw	7,v	6,v	3,x	6,x	6,x	8,x
4	xvuwy	7,v	6,v	3,x	6,x	6,x	8,x
5	xvuwyt	7,v	6,v	3,x	6,x	6,x	8,x
6	xvuwytz	7,v	6,v	3,x	6,x	6,x	8,x

LS算法的缺陷

可能产生“振荡”。

例图4-29,一个简单网络拓扑。设:

- 链路费用等于链路上的负载;
- 链路费用是非对称的:
- $c(u，v)$与$c(v，u)$仅当在链路$(u，v)$两个方向的负载相同时才相等。
- 有三个同时发往w的注入流量:
  - 节点z: 一个单元;
  - 节点x: 一个单元;
  - 节点y: e的流量。

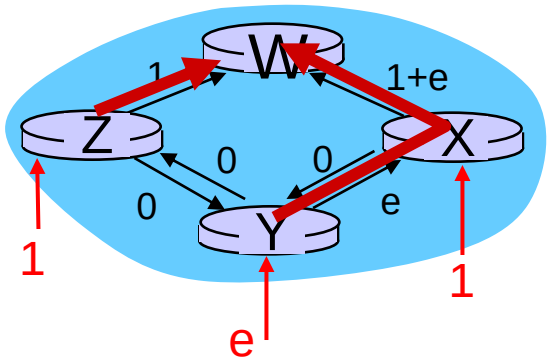


初始：图(a)，其链路费用相当于承载的流量

第一次：节点x、y、z都确定顺时针方向到w的路径费用最低，为1。产生图(b)费用。

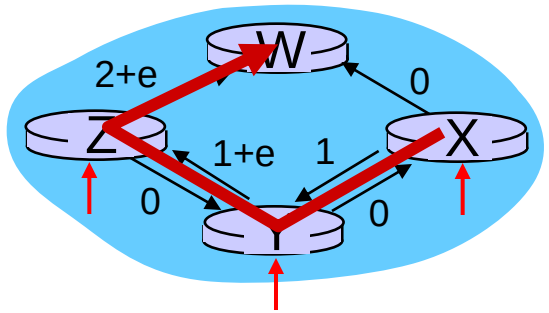
第二次：节点x、y、z都确定逆时针方向到w的路径费用最低，为0。产生图(c)费用。

第三次：节点x、y、z都确定顺时针方向到w的路径费用最低，为0。产生图(d)费用。



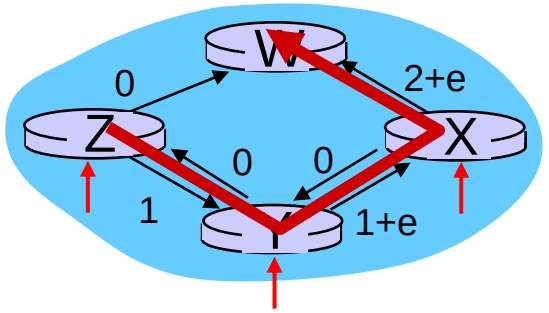
initially

(a)



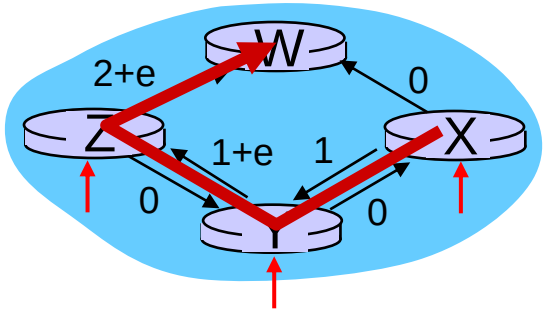
given these costs,
find new routing….
resulting in new costs

(b)



given these costs,
find new routing….
resulting in new costs

(c)



given these costs,
find new routing….
resulting in new costs

(d)

避免振荡

- 使链路费用不依赖于所承载的流量：不可行；
- 避免所有的路由器同时运行LS算法：比较合理。
  - 即使路由器以相同周期(每30分钟)运行LS算法，在每个节点上算法的执行时刻也不同。

因特网的自同步：

- 即使路由器初始时以同一周期但不同时刻执行算法，算法执行时刻最终会同步并保持。采用随机时间的方法，即每台路由器随机发送链路通告。

5.2.2 距离向量选路算法DV

- DV是一种迭代的、异步的和分布式选路算法。
  - 分布式：每个节点都从其直接相连邻居接收信息，进行计算，再将计算结果分发给邻居。
  - 迭代：计算过程一直持续到**邻居之间无更多信息交换为止**。
  - 自我终结：算法能自行停止。
  - 异步：不要求所有节点相互之间步伐一致地操作。

最低费用表示

- $d_x(y)$：节点x到节点y的最低费用路径的费用。
  - 用Bellman-Ford方程表示

$$d_x(y) = \min_v\{c(x, v) + d_v(y)\} \qquad \text{(B-F)}$$

- **$c(x, v) + d_v(y)$**：x与某个邻居v之间的直接链路费用$c(x, v)$加上邻居v到y的最小费用，即**x经v到节点y的最小的路径费用**。
- **min_v**：从所有经直接相连邻居到节点y的费用中选取的最小路径费用。

例子

源节点*u*到目的节点*z*的最低费用路径。

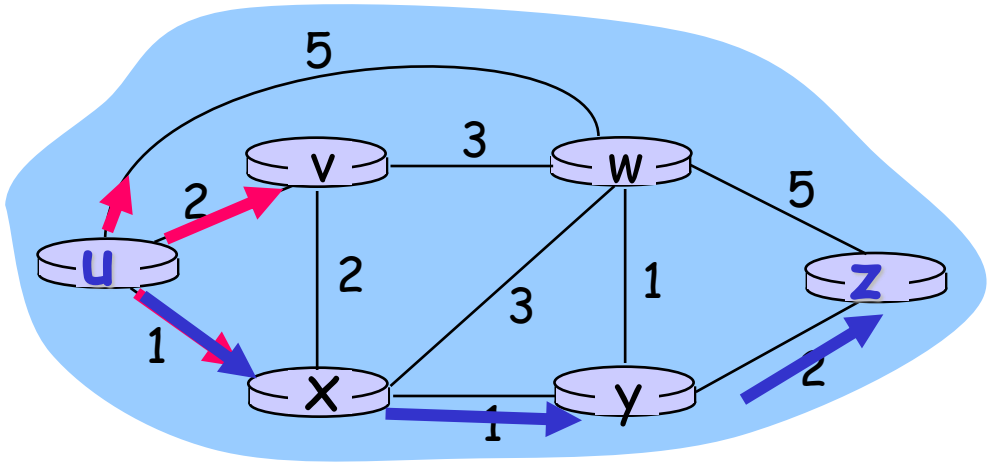
源节点*u*有三个邻居：

- 邻居*v*：$d_v(z) = 5$ 、 $c(u,v) = 2$
- 邻居*w*：$d_w(z) = 3$ 、 $c(u,w) = 5$
- 邻居*x*：$d_x(z) = 3$ 、 $c(u,x) = 1$

$$d_u(z) = \min\{\, c(u,v) + d_v(z),\ c(u,w) + d_w(z),\ c(u,x) + d_x(z)\}$$

$$= \min\{2+5,\ 5+3,\ 1+3\} = 4$$

即源节点*u*经相邻节点*x*到目的节点*z*的路径费用最低，为4。



得到节点*u*的转发表中到目的节点*z*的下一跳是节点*x*

距离向量算法

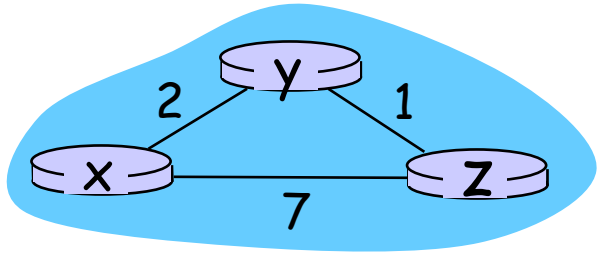
- $D_x(y)$ = 节点x到y的最小费用估计值
  - x 维护距离向量 $\mathbf{D}_x = [D_x(y): y \in N]$
- 节点 x:
  - 知道x到每个邻居节点v的花费: c(x, v)
  - 维护邻居节点的距离向量。对于每个邻居节点v，x维护 $\mathbf{D}_v = [D_v(y): y \in N]$

列：网络中所有的节点

行：该节点的距离向量*Dx*和其邻居的距离向量*Dv*

		x	y	z
	x	$D_x(x)$	$D_x(y)$	$D_x(z)$
邻居	y	$D_y(x)$	$D_y(y)$	$D_y(z)$
邻居	z	$D_z(x)$	$D_z(y)$	$D_z(z)$

	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞



注意提醒同学：y，z对应的行的数据是从y节点和z节点获取的距离向量，而不是由x节点计算的，x只依赖于第二行和第三行的数据，更新自己的距离向量

距离向量算法

- 核心思想
  - 每个节点不断向邻居节点发送自己的距离向量估计值Dx
  - 当x从邻居节点收到新的距离向量估计值Dv，它使用B-F方程更新自己到其他节点的距离向量Dx。

$$D_x(y) \leftarrow \min_v\{c(x,v) + D_v(y)\} \quad \text{for each node } y \in N$$

  - 最终$D_x(y)$会收敛到实际的最小花费$d_x(y)$

距离向量算法

- *iterative, asynchronous:*

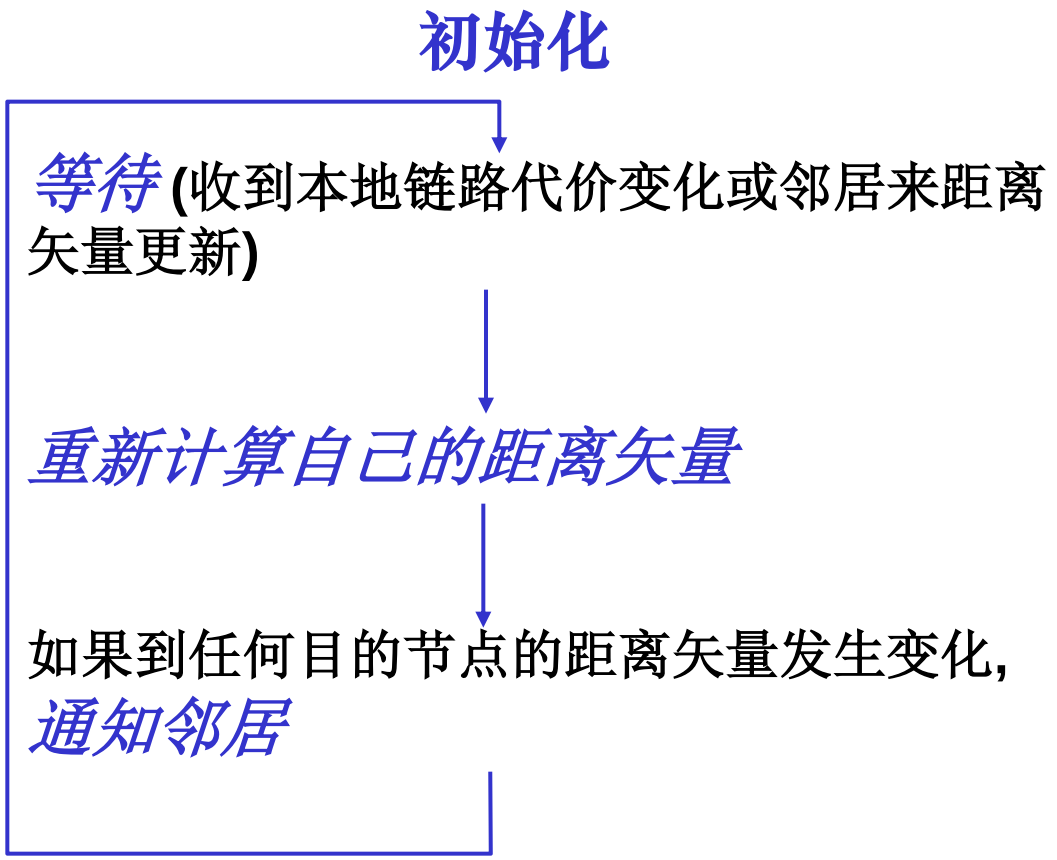
每次本地迭代由如下事件触发：

  - 本地链路的费用发生改变；
  - 收到来自邻居的DV更新消息；

- *distributed:*

当节点的DV发生改变，它会通知它的邻居节点

  - 如果存在必要，邻居节点会继续通知它的邻居



距离向量算法

在所有的节点，X：

初始化

```
1  Initialization:
2     for all destinations y in N:
3          Dx (y) = c( x, y )        /* if y is not a neighbor then c( x, y )= ∞ */
4     for each neighbor w
5          Dw (y) = ∞ for all destinations y in N
6     for each neighbor w
7          send DV Dx = [Dx (y) :y in N]  to w

9     loop
10        wait (until I see a link cost change to neighbor w or
11                 until I receive update from neighbor w)
13        for each  y in N:
14           Dx(y) = c(x,v) + Dv(y)
15
16        if Dx(y) changed for any destination y
17           Send DV Dx = [Dx (y) : y in N]  to all neighbors
19  forever
```

$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$
$= \min\{2+0 , 7+1\} = 2$
直接到y费用最小。X到y的下一跳为y

$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$
$= \min\{2+1 , 7+0\} = 3$
x经y到z费用最小。x到z的下一跳为y

节点x的距离向量表

目的节点

邻居

y 的新距离向量

z 的新距离向量

向邻居发送

稳定

节点 z

	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

	y	z
x	∞	∞
y	2	1
z	∞	∞

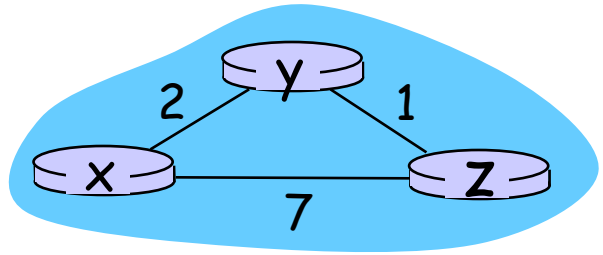
	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

	y	z
x	∞	∞
y	∞	∞
z	7	1

	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0



初始化

第一次

第二次

time

$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$
$= \min\{2+0 , 7+1\} = 2$
直接到y费用最小。X到y的下一跳为y

$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$
$= \min\{2+1 , 7+0\} = 3$
x经y到z费用最小。x到z的下一跳为y

节点x的距离矢量表

目的节点

节点 y

节点 z

	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

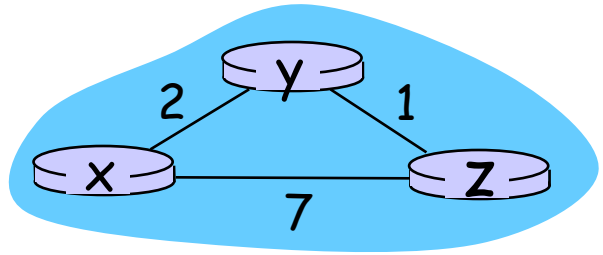
	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0

	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0



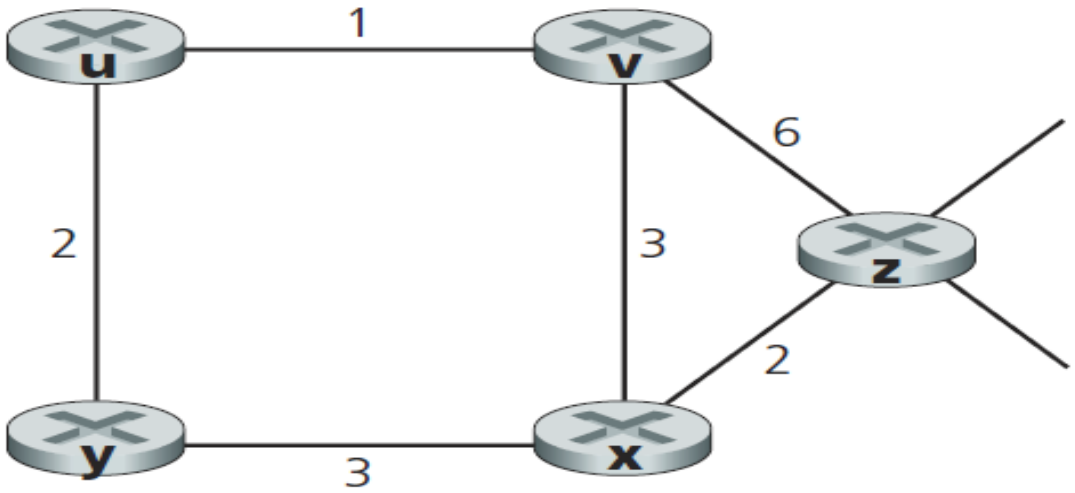
初始化 → 第一次 → 第二次 → time

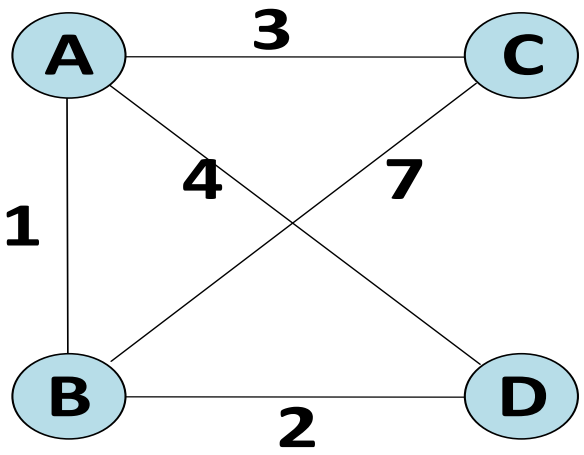
距离向量算法

- 多次重复从邻居接收更新距离向量、重新计算自己的距离向量、并向邻居发送更新通知的过程，一直持续到没有更新报文发出为止。算法进入静止状态，直到某个链路费用发生改变为止。

课堂练习

P28. Consider the network shown below, and assume that each node initially knows the costs to each of its neighbors. Consider the distance-vector algorithm and show the distance table entries at node z.





节点A

	A	B	C	D
A	0	1	3	4
B	∞	∞	∞	∞
C	∞	∞	∞	∞
D	∞	∞	∞	∞

（0.5分）

	A	B	C	D
A	0	1	3	3
B	1	0	7	2
C	3	7	0	∞
D	4	2	∞	0

（1分）

	A	B	C	D
A	0	1	3	3
B	1	0	4	2
C	3	4	0	7
D	3	2	7	0

（1分）

	A	B	C	D
A	0	1	3	3
B	1	0	4	2
C	3	4	0	6
D	3	2	6	0

（0.5分）

节点B

	A	B	C	D
A	∞	∞	∞	∞
B	1	0	7	2
C	∞	∞	∞	∞
D	∞	∞	∞	∞

（0.5分）

	A	B	C	D
A	0	1	3	4
B	1	0	4	2
C	3	7	0	∞
D	4	2	∞	0

（1分）

	A	B	C	D
A	0	1	3	3
B	1	0	4	2
C	3	4	0	7
D	3	2	7	0

（1分）

	A	B	C	D
A	0	1	3	3
B	1	0	4	2
C	3	4	0	6
D	3	2	6	0

（0.5分）

节点C

	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	∞	∞
C	3	7	0	∞

（0.5分）

	A	B	C	D
A	0	1	3	4
B	1	0	7	2
C	3	4	0	7

（1分）

	A	B	C	D
A	0	1	3	3
B	1	0	4	2
C	3	4	0	6

（1分）

	A	B	C	D

节点D

	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	∞	∞
D	4	2	∞	0

（0.5分）

初始化

	A	B	C	D
A	0	1	3	4
B	1	0	7	2
D	3	2	7	0

（1分）

第一次迭代

	A	B	C	D
A	0	1	3	3
B	1	0	4	2
D	3	2	6	0

（1分）

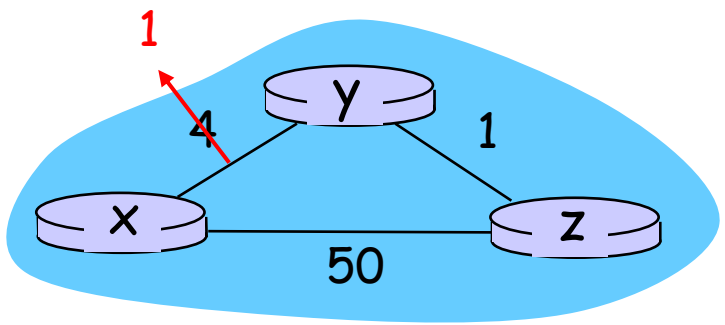
第二次迭代

	A	B	C	D

第三次迭代

1、链路费用改变与链路故障

- 当一个节点检测到从它到邻居的链路费用发生变化时，就更新其距离向量，如果最低费用路径的费用发生变化，通知其邻居。
- **某链路费用减少时情况**
  - 如图，当y 到x的链路费用从4变为1的情况。

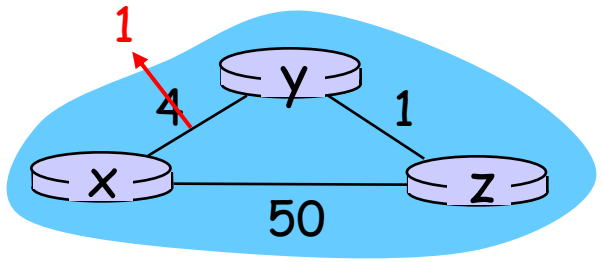


考虑y与z到目的节点x的距离表变化

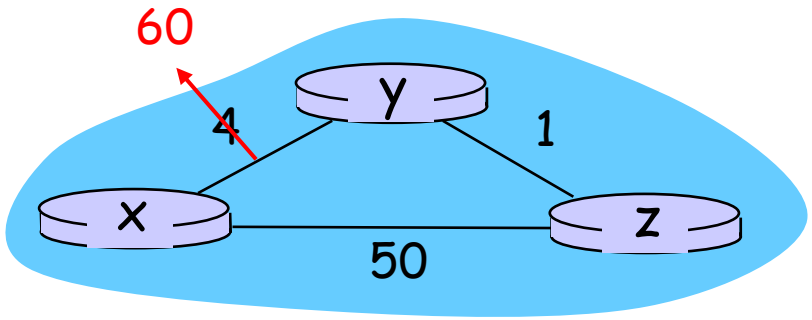
- *t0*：y 检测到x的链路费用从4变为1，更新其距离向量，并通知其邻居z；
- *t1*：z收到来自y的更新报文，并更新自己的距离表，此时到节点x的最低费用减为2，并通知其邻居y；
- *t2*：y收到来自z的更新报文，并更新自己的距离表，此时到节点x的最低费用不变仍为1。不发送更新报文，算法静止。

当x与y之间费用减少，DV算法只需两次迭代到达静止状态。

节点之间链路费用减少的“好消息”在网络中能迅速传播。



1、链路费用改变与链路故障



- **某链路费用增加时情况**
  - 如图，设*x*与*y*之间的链路费用从4增加到60。
- 链路费用变化前

  $D_y(x)=4,\ D_y(z)=1,\ D_z(x)=5$

- ✓ *t0 时刻*: *y*检测到链路费用从4变为60。更新到*x*的最低路径费用

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

经节点*z*到x费用最低，此新费用错误，发给节点*z*。

✓ *t1时刻*：z收到新费用，更新其到x的最低路径费用

$$Dz(x)=\min\{c(z,x)+Dx(x),\ c(z,y)+Dy(x)\}$$
$$=\min\{50+0,1+6\}=7$$

经节点 *y* 到 *x* 费用最低，发给节点 *y*。

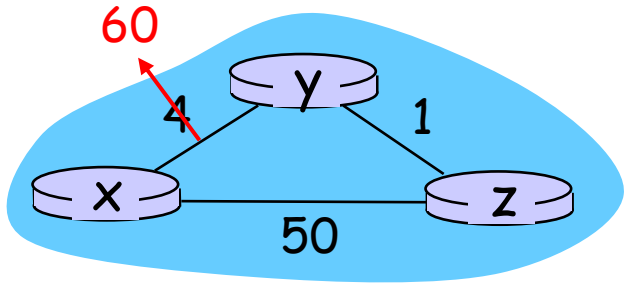
✓ *t2 时刻*：y收到新费用，更新到x的最低路径费用

$$Dy(x)=\min\{c(y,x)+Dx(x),\ c(y,z)+Dz(x)\}$$
$$=\min\{60+0,1+7\}=8$$

经节点z到x费用最低，发给节点z。

……

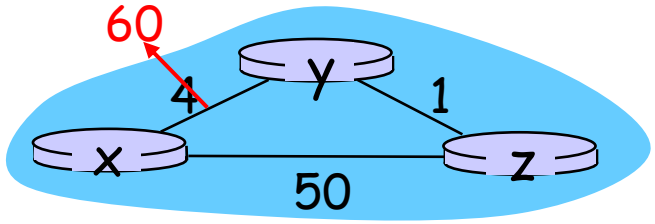
节点y或z的最低费用不断更新。



❒ 产生“选路回环”：为到达x，y通过z选路，z又通过y选路。即目的地为x的分组到达y或z后，将在这两个节点之间不停地来回反复，直到转发表发生改变为止。

1、链路费用改变与链路故障

- 上述循环将持续44次迭代 (*y*与*z*之间的报文交换)，直到*z*最终算出它经由*y*的路径费用大于50为止。并确定：
  - *z*到*x*的最低费用路径：*zx*
  - *y*到*x*的最低费用路径：*yzx*

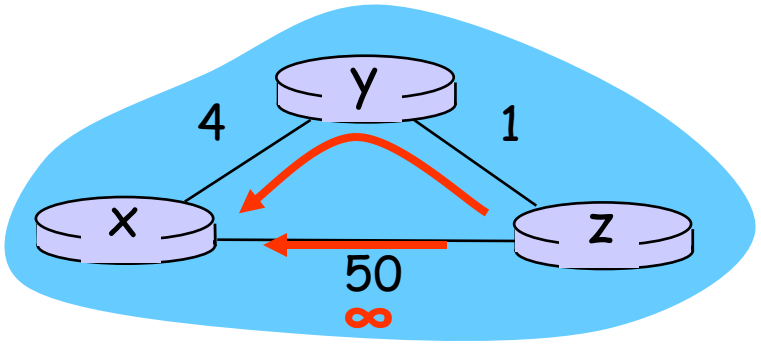


- 说明：链路费用增加的“坏消息”传播很慢!
- 当链路费用增加很大，会出现“计数到无穷”问题。
- 如链路费用c(y, x)变为10000，c(z, x)为9999时。

2、增加毒性逆转

避免出现“选路回环”现象。

- 基本思想：如果 z 通过 y 选路到达目的地 x，则 z 将告诉 y，它到 x 的距离是无穷大（即 z 将告诉 y，$Dz(x)=\infty$）；y 认为 z 没有到 x 的路径，就不会试图经由z选路到 x。“毒性逆转”用来解决图中的特定环路。



设使用“毒性逆转”后，y距离表显示$Dz(x)=\infty$。

✓ *t0 时刻*: (x,y)链路费用从4变为60。y更新其到x的最低费用

$$Dy(x)=\min\{c(y,x)+Dx(x), c(y,z)+Dz(x)\}$$
$$=\min\{60+0,1+\infty\}=60$$

y直接选路到x，将新费用$Dy(x)=60$，通知z 。

✓ *t1时刻*：z收到新费用，计算其到x的新的最低路径费用

$$Dz(x)=\min\{c(z,x)+Dx(x), c(z,y)+Dy(x)\}$$
$$=\min\{50+0,1+60\}=50$$

z直接选路到x，将新费用$Dz(x)=50$，通知y。

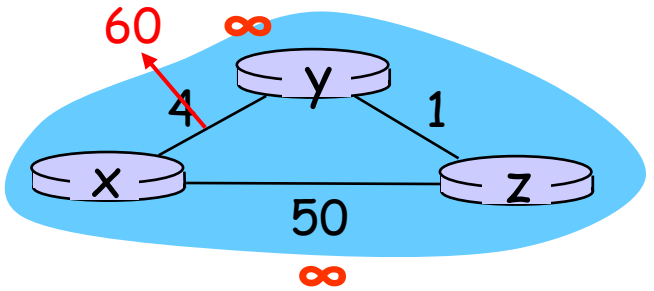
✓ *t2 时刻*：y收到新费用，计算其到x的新的最低路径费用

$$Dy(x)=\min\{c(y,x)+Dx(x), c(y,z)+Dz(x)\}$$
$$=\min\{60+0,1+50\}=51$$

*y*经*z*到*x*费用最低，使用“毒性逆转”抑制从*y*到*x*方向的路径

✓ *t3 时刻*：*y*通知z ，*y*到*x*费用无穷大，$Dy(x)=\infty$。

算法静止。



3、LS算法与DV算法比较

- *DV算法：*每个节点只与邻居互相交流，得到邻居节点到网络中其他节点的最低费用，并告知邻居节点自己当前到其他节点的最低费用。
- *LS算法：*每个节点与所有其他节点广播交流，只告知与其直接相连链路的费用。
- ***报文复杂性：***
  - *LS算法：*需要知道网络每条链路的费用，需发送O(nE)个报文；当一条链路的费用变化时，必须通知所有节点；

3、LS算法与DV算法比较

- ***报文复杂性：***
  - *DV算法*：迭代时，在两个直接相连邻居之间交换报文；收敛时间受许多因素影响；当链路费用改变时，只有该链路相连的节点的最低费用路径发生改变时，才传播已改变的链路费用。
- ***收敛速度：***
  - *LS算法*：需要O(nE)个报文和O(n2)的搜寻。
  - *DV算法：*收敛较慢。可能会遇到选路回环，或计数到无穷的问题。

3、LS算法与DV算法比较

- ***健壮性(robust)：*** 当一台路由器发生故障、操作错误或受到破坏时，会发生什么情况?
  - ***LS算法：*** 路由器向其连接的链路广播不正确费用。每个路由器独立计算自己的转发表，因此提供一定的健壮性。
  - ***DV算法：*** 一个节点可向任意或所有目的节点发布其不正确的最低费用路径。在DV中，一个节点的计算值会传递给它的邻居，并间接地传递给邻居的邻居。一个不正确的计算值会扩散到整个网络。

5.3 因特网中自治系统内部的路由选择：OSPF

- 使路由可具有扩展性
  - 我们目前的研究是非常理想化的
    - 所有路由器都是相同的
    - 网络是扁平的
- 扩展性问题：数十亿的目的主机
  - 在路由转发表中无法存储所有的目的主机的地址
  - 路由表的交换将淹没链路的带宽容量
- 行政上的自治问题：
  - 因特网是由各种网络互联而成的
  - 每个网络的管理者会希望由自己来控制自己网络中的路由信息！

但在实际中，这并不是真实的

支持扩展性路由的因特网方案

可以将路由器聚合为若干的区域，称为自治系统或自治域(autonomous systems, AS)

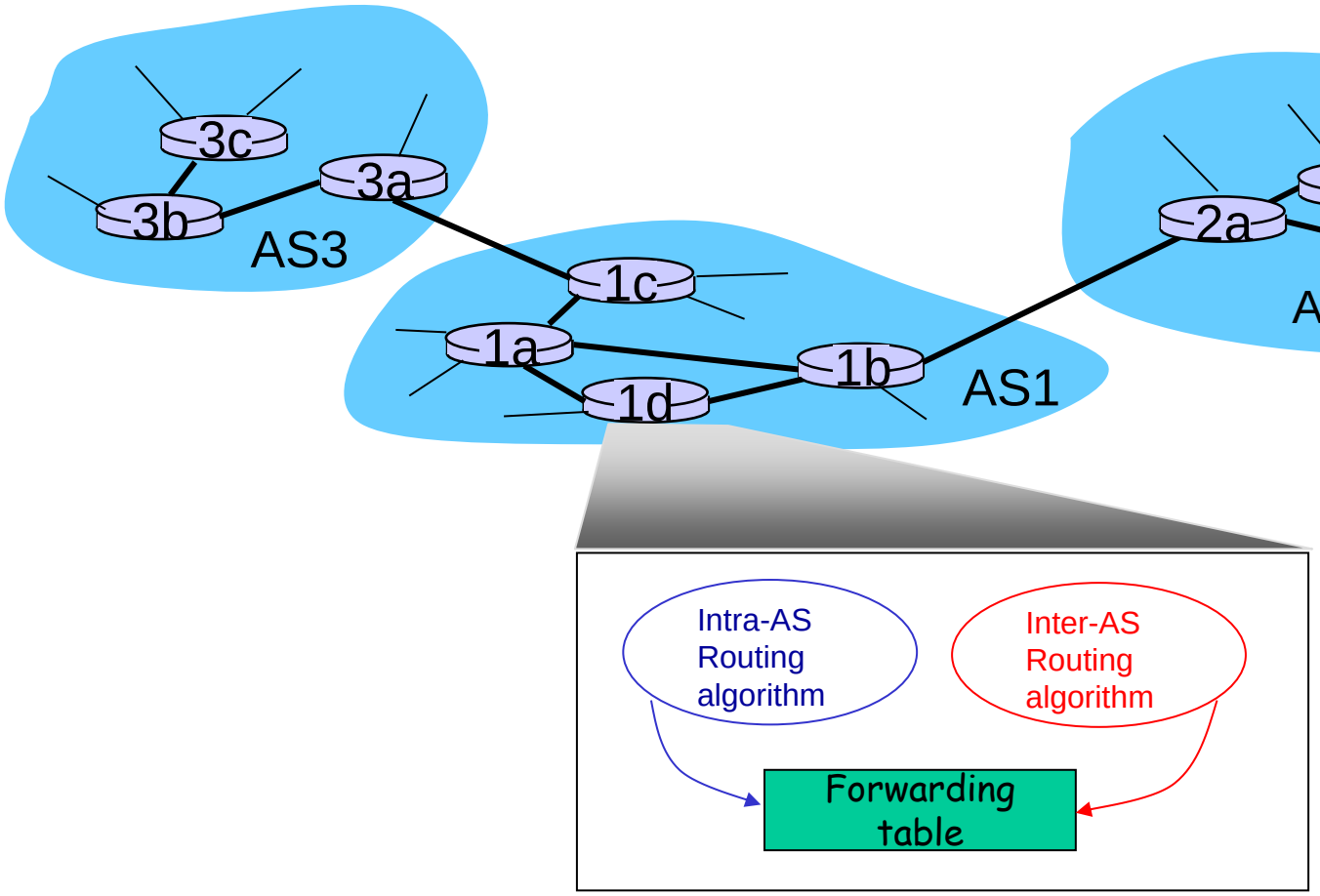
intra-AS 路由(域内)

- 位于相同AS的主机和路由器之间的路由
- 在AS内部的所有路由器必须运行相同的域内路由协议
- 位于不同AS的路由器可以运行不同的域内路由协议
- 网关路由器：位于AS的边缘，有链路连接到其他AS的路由器

inter-AS 路由(域间)

- AS之间的路由
- 网关执行域间路由协议

互联的ASes



- 转发表的配置由域内和域间路由算法完成
  - intra-AS 路由确定AS内部的目的地址的转发表项
  - inter-AS & intra-AS 确定外部目的地址的转发表项

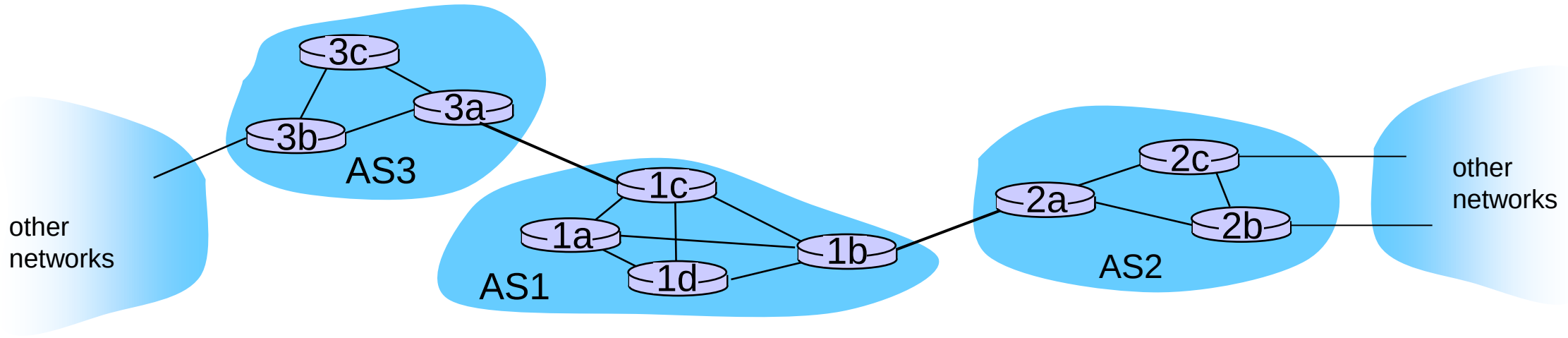
Inter-AS 任务

- 假设AS1中的路由器收到目的地址是AS1之外的数据报：
  - 路由器将转发分组到网关路由器，但是哪个呢?

AS1 必须:

1. 学习通过AS2可以到达哪些目的地址，通过AS3可以达到哪些目的地址
2. 将可达信息转播给AS1中的所有路由器

job of inter-AS routing!



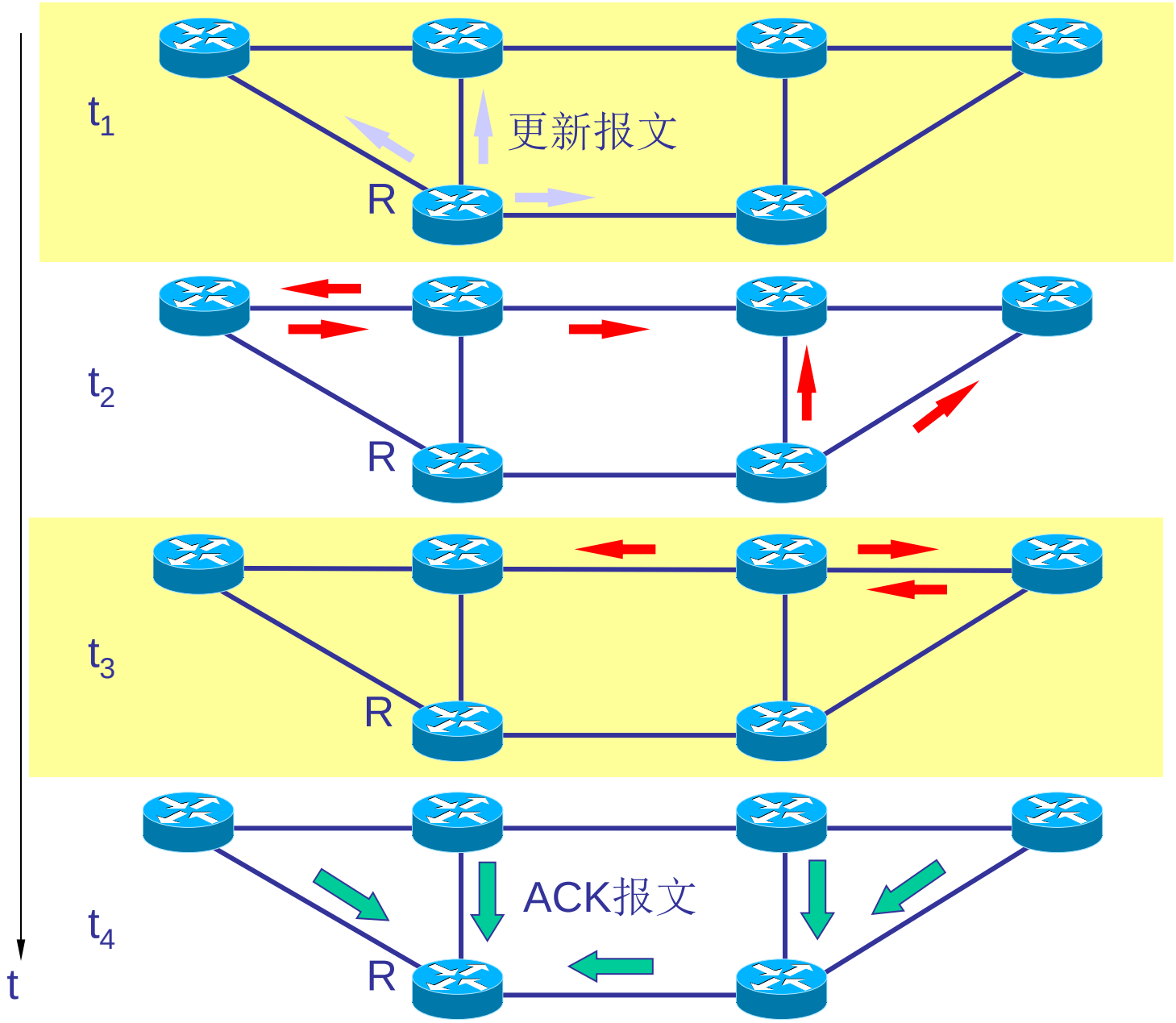
Intra-AS 路由

- 域内路由协议也称为内部网关协议(interior gateway protocols, IGP)
- 目前应用最广泛的域内路由协议包括:
  - RIP: Routing Information Protocol
  - OSPF: Open Shortest Path First (IS-IS protocol essentially same as OSPF)
  - IGRP: Interior Gateway Routing Protocol (Cisco proprietary for decades, until 2016)

开放最短路径优先OSPF

Open Shortest Path First

- “open”：公开可用
- uses link-state algorithm
  - link state packet 扩散
  - 每个节点维护拓扑图
  - 使用Dijkstra算法计算路由
- 路由器泛洪OSPF link-state 公告给所有的位于同一个AS的路由器
  - OSPF消息直接封装在IP报文中(而不是TCP或UDP报文段)
  - link state: 每条直接相连的链路

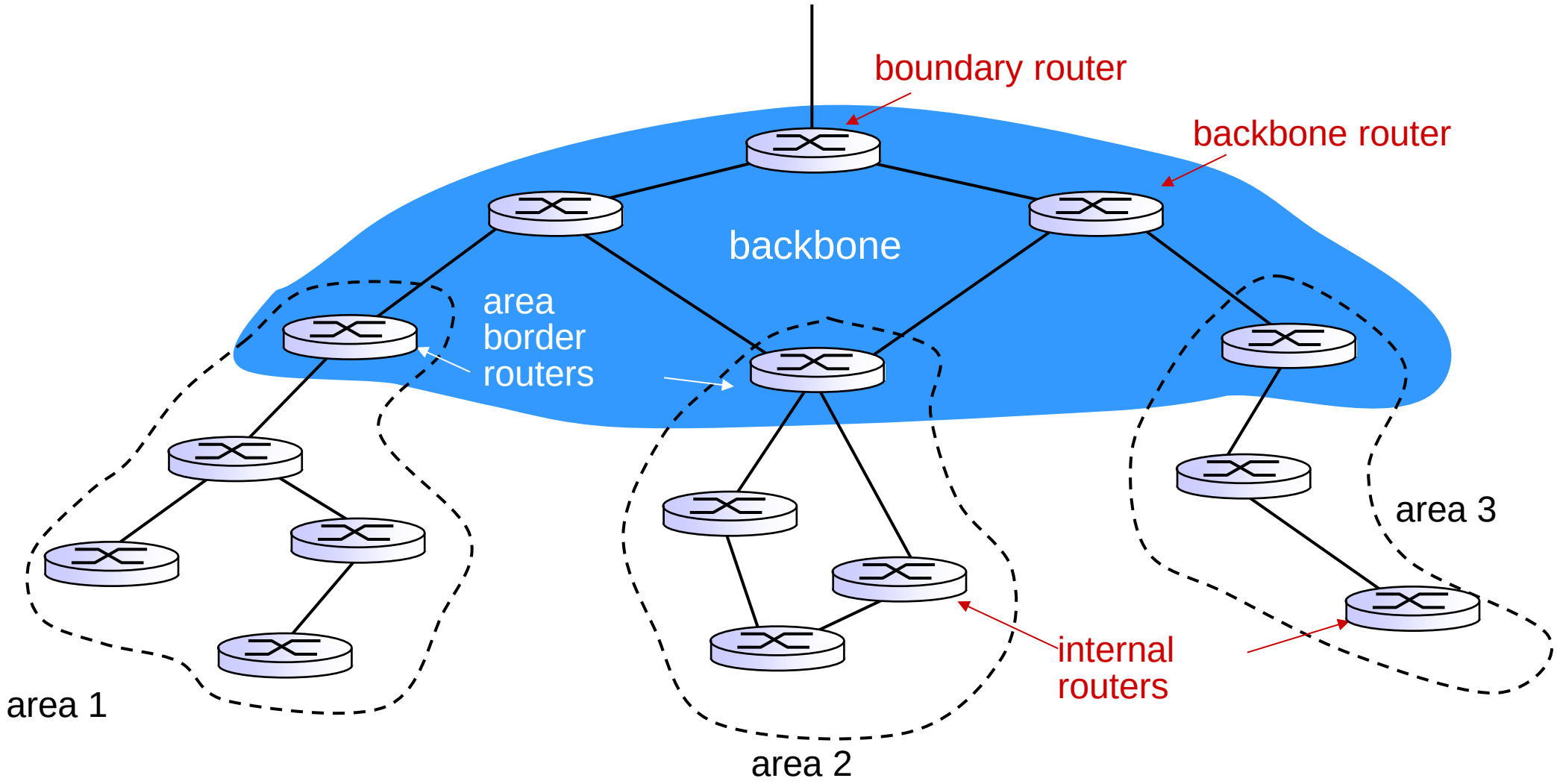


OSPF 使用的是可靠的洪泛法

OSPF “advanced” features

- 安全: 所有OSPF消息都是加密的(以防止恶意入侵)
- 允许多条有相同开销(cost)的路径(在RIP协议中只允许1条)
- 对于每条链路，针对不同的TOS可以设置不同的cost(例如，对于best effort ToS 可以将cost设得较低，而对于real-time ToS 可以设得较高)
- 支持在单个AS中的层次结构

Hierarchical OSPF



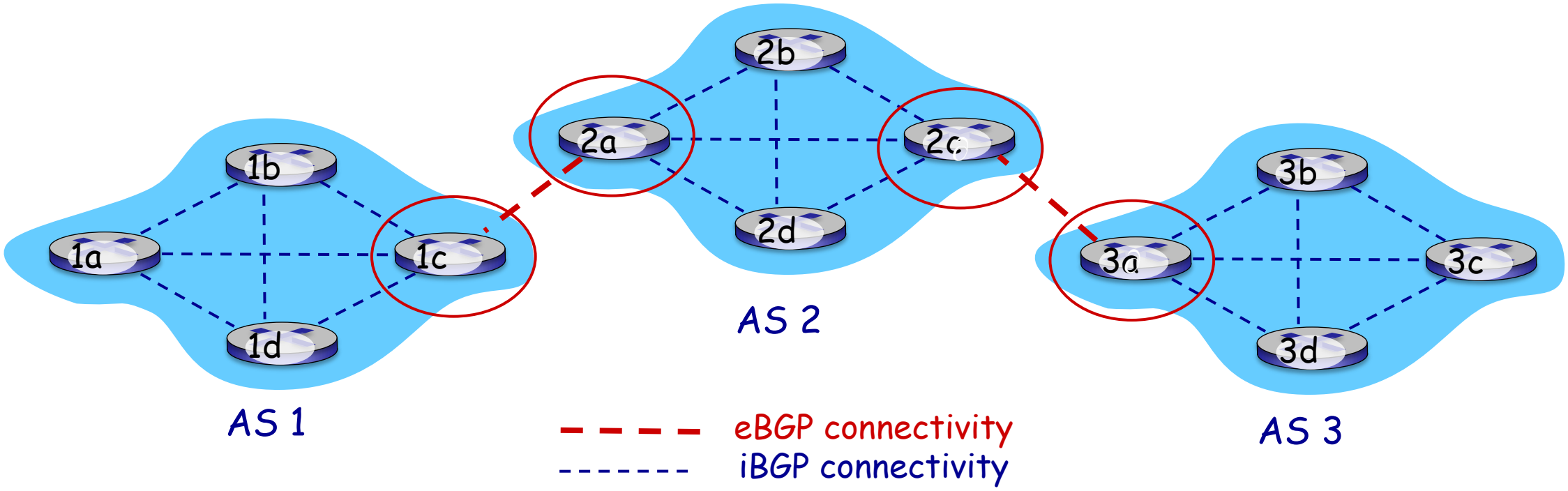
Hierarchical OSPF

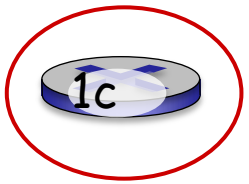
- *two-level hierarchy:* local area, backbone.
  - link-state advertisements only in area
  - each nodes has detailed area topology; only know direction (shortest path) to nets in other areas.
- *area border routers:* "summarize" distances to nets in own area, advertise to other Area Border routers.
- *backbone routers:* run OSPF routing limited to backbone.
- *boundary routers:* connect to other AS'es.

5.4 ISP之间的路由选择：BGP

- BGP (Border Gateway Protocol): 域间路由协议
  - 粘合数以千计的ISP
- BGP为每个AS提供如下功能:
  - eBGP(外部): 从其邻居AS获得子网的可达信息
  - iBGP(内部): 传播可达信息给AS内部的所有路由器.
  - 基于可达信息和策略，确定到其他网络的好路由（并不一定是最优）
- 允许子网(subnet)向英特网公告它自己的存在信息: *“I am here”*

eBGP, iBGP connections



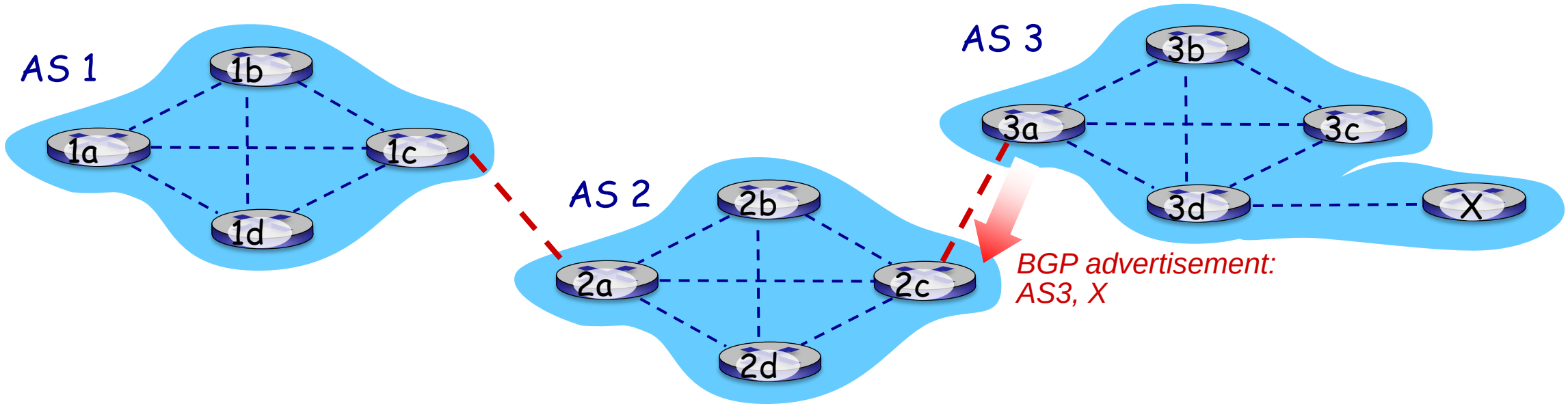


gateway routers run both eBGP and iBGP protools
(网关路由器同时运行边界BGP和内部BGP协议)

BGP 基础

- BGP 会话: 两个BGP路由器(peers)交换BGP消息，通过半永久的TCP连接:
  - advertising *paths* to different destination network prefixes (BGP is a "path vector" protocol)

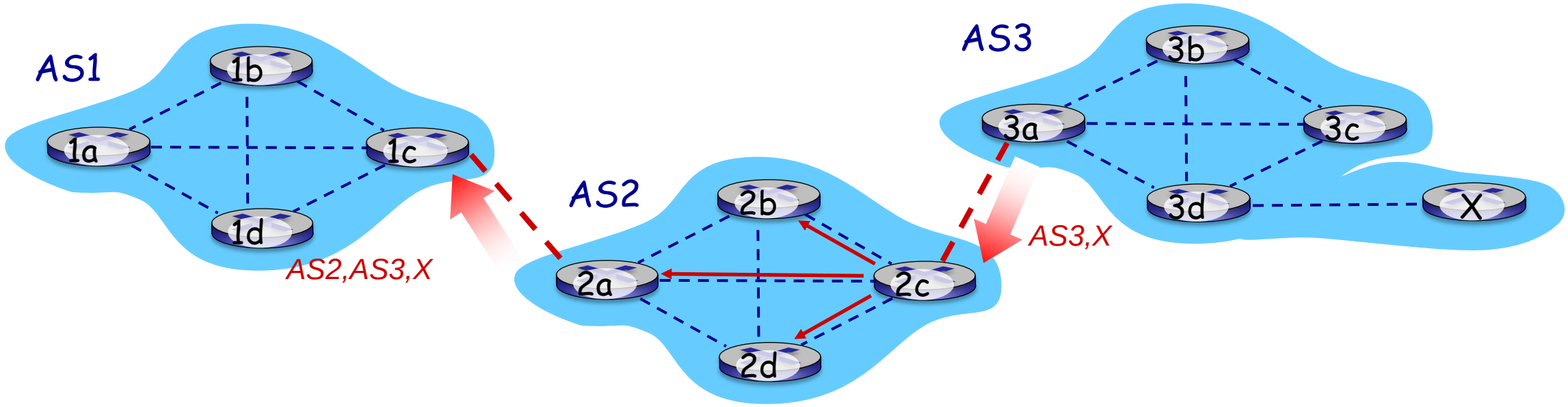
- when AS3 gateway router 3a advertises path (AS3, X) to AS2 gateway router 2c:
  - AS3 *promises* to AS2 it will forward datagrams towards X



Path属性和BGP路由

- 公告前缀包括BGP属性
  - prefix + attributes = “route”
- 两个重要的属性：
  - AS-PATH: 该前缀公告已经通过的AS的列表
  - NEXT-HOP: 指出连接下一跳AS的域间路由器的地址

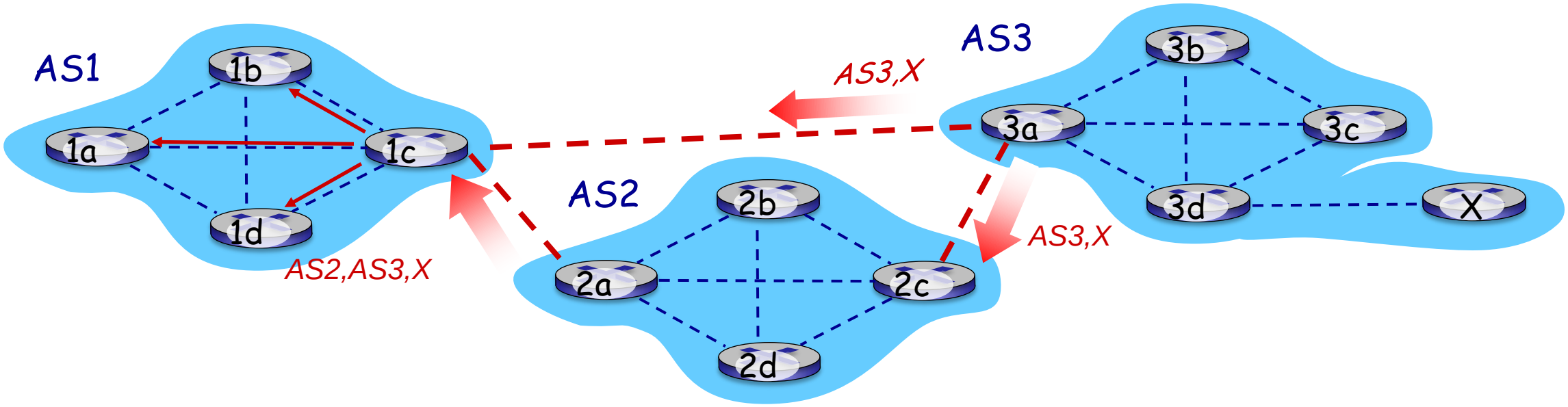
BGP 路径公告的例子



- AS2 router 2c receives path advertisement AS3,X (via eBGP) from AS3 router 3a

- Based on AS2 policy, AS2 router 2c accepts path AS3,X, propagates (via iBGP) to all AS2 routers
- Based on AS2 policy, AS2 router 2a advertises (via eBGP) path AS2, AS3, X to AS1 router 1c

BGP 路径公告的例子

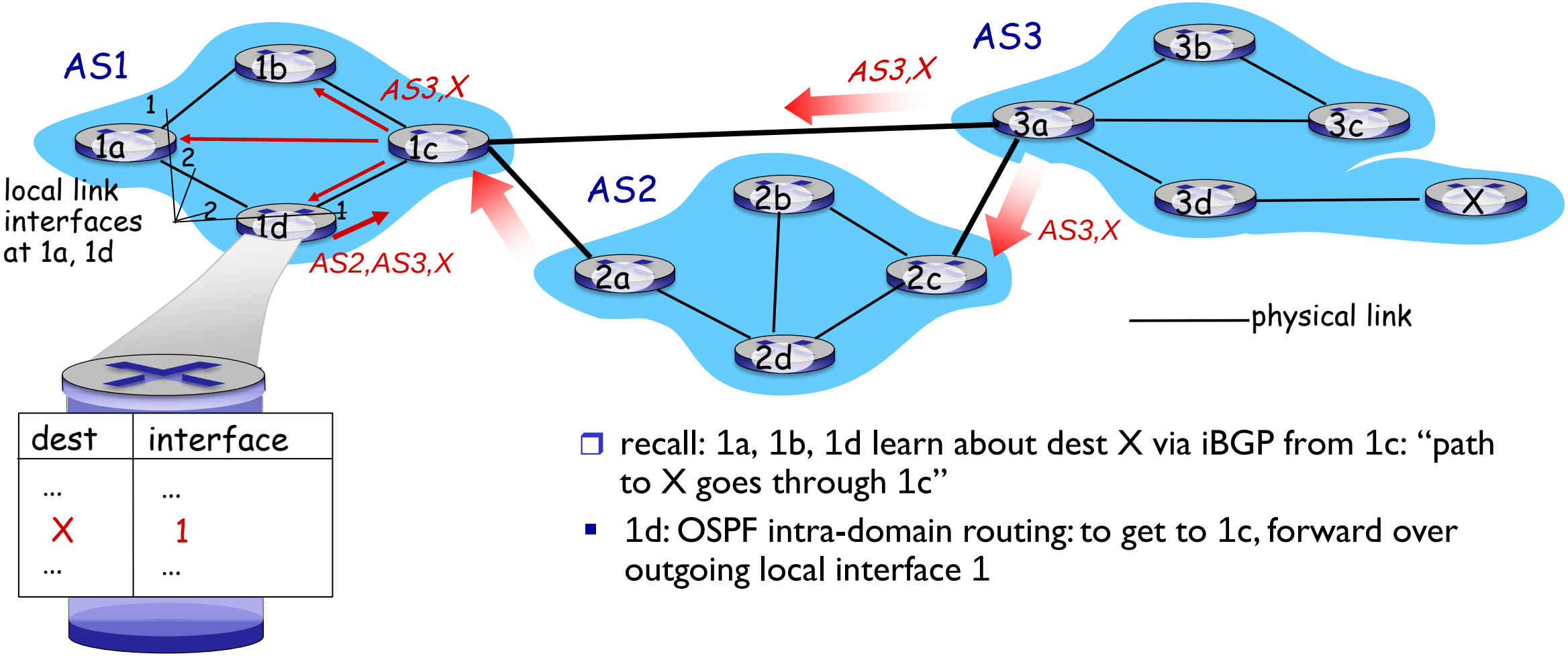


gateway router may learn about multiple paths to destination:

- AS1 gateway router 1c learns path *AS2,AS3,X* from 2a
- AS1 gateway router 1c learns path *AS3,X* from 3a
- Based on policy, AS1 gateway router 1c chooses path *AS3,X, and advertises path within AS1 via iBGP*

BGP, OSPF, 转发表的表项

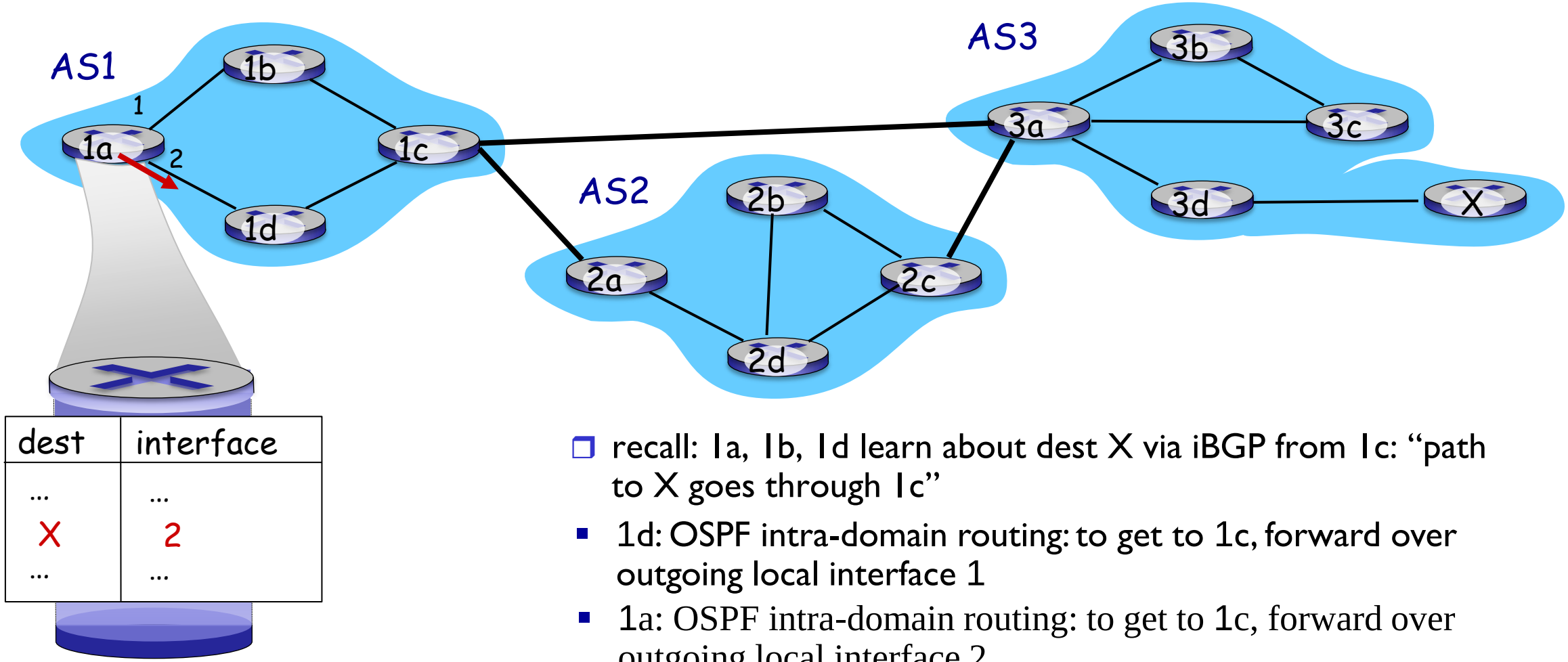
Q: how does router set forwarding table entry to distant prefix?



- recall: 1a, 1b, 1d learn about dest X via iBGP from 1c: "path to X goes through 1c"
- 1d: OSPF intra-domain routing: to get to 1c, forward over outgoing local interface 1

BGP, OSPF, 转发表的表项

Q: how does router set forwarding table entry to distant prefix?

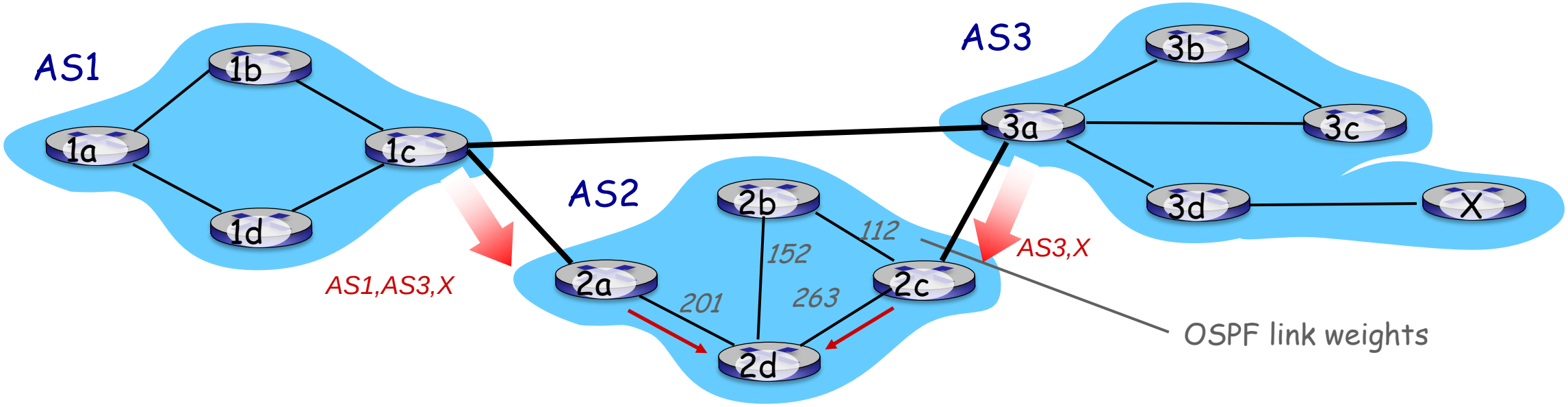


- recall: 1a, 1b, 1d learn about dest X via iBGP from 1c: "path to X goes through 1c"
  - 1d: OSPF intra-domain routing: to get to 1c, forward over outgoing local interface 1
  - 1a: OSPF intra-domain routing: to get to 1c, forward over outgoing local interface 2

BGP 路由选择

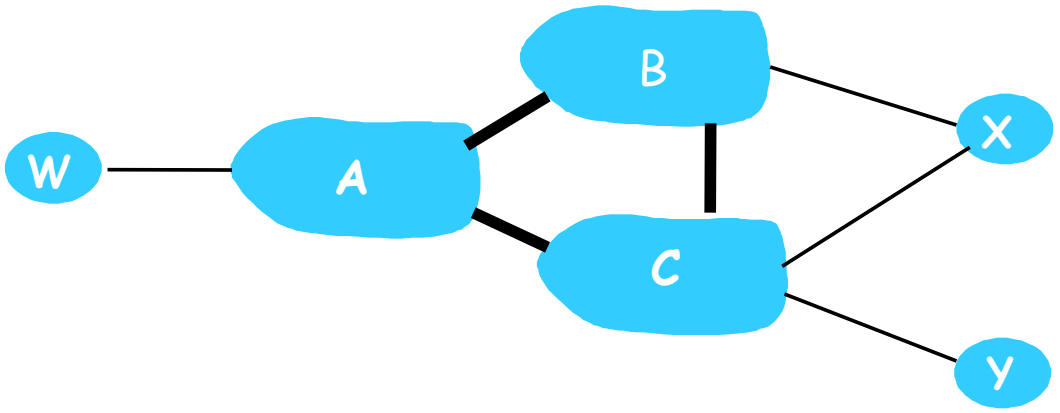
- 路由器可能学习得到有多于1条路由到达目的AS，可以基于以下原则选择路由：
  1. 基于本地偏好值属性选择：策略决策
  2. 最短的AS-PATH选择
  3. 最近的NEXT-HOP路由器：hot potato routing
  4. 额外的规则

Hot Potato Routing(热土豆路由)



- 2d learns (via iBGP) it can route to X via 2a or 2c
- *hot potato routing:* **choose local gateway that has least intra-domain cost** (e.g., **2d chooses 2a**, even though more AS hops to *X*): don't worry about inter-domain cost!

BGP: 通过公告实现策略

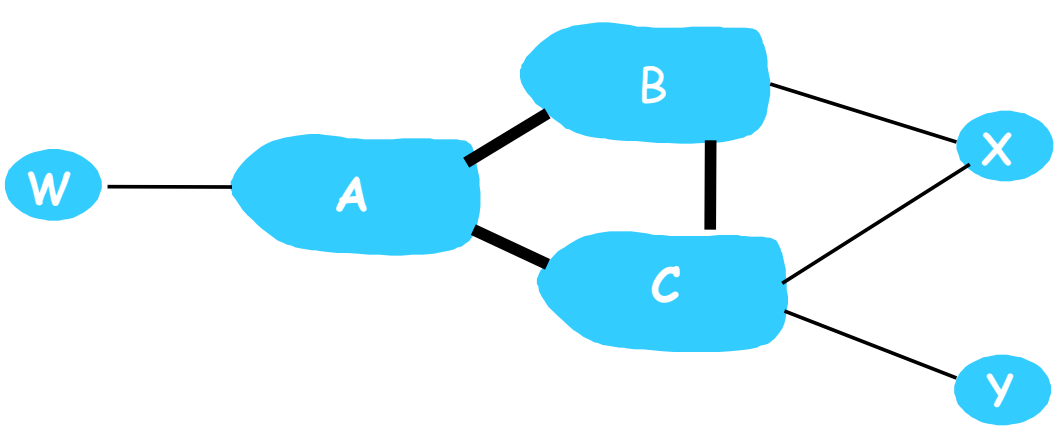


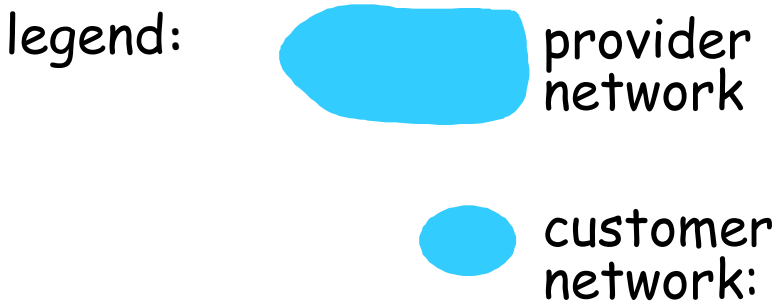
图例:

Suppose an ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs)

- A advertises path (A, w) to B and to C
- B ***chooses not to advertise*** (B, A, w) to C:
  - B gets no "**revenue**" for routing (C, B, A, w), since none of C, A, w are B's customers
  - C does not learn about (C, B, A, w) path
- C will route C, A, w) (not using B) to get to w

BGP: 通过公告实现策略





Suppose an ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs)

- A,B,C are *provider networks*
- X,W,Y are customer (of provider networks)
- X is *dual-homed:* attached to two networks
- *policy to enforce:* X does not want to route from B to C via X
  - .. so X will not advertise to B a route to C (B, x, C)

5.6 ICMP: 因特网控制报文协议

- 用于主机或路由器相互之间交换网络层的信息
  - 错误报告**(error reporting): unreachable host, network, port, protocol**
  - echo request/reply (used by ping)
- network-layer “above” IP:
  - ICMP msgs carried in IP datagrams
- ICMP message: type, code plus first 8 bytes of IP datagram causing error

Type	Code	description
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control - not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header

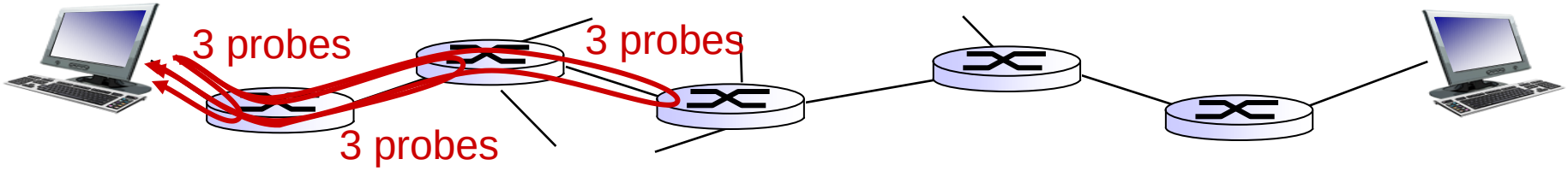
Traceroute and ICMP

- source sends series of UDP segments to destination
  - first set has TTL =1
  - second set has TTL=2, etc.
  - unlikely port number
- when datagram in *n*th set arrives to nth router:
  - router discards datagram and sends source ICMP message (type 11, code 0)
  - ICMP message include name of router & IP address

- when ICMP message arrives, source records RTTs

stopping criteria:

- UDP segment eventually arrives at destination host
- destination returns ICMP "port unreachable" message (type 3, code 3)
- source stops



什么是SDN

- 软件定义网络（SDN，Software Defined Network）源自美国斯坦福大学Clean State研究组提出的一种新型网络创新架构，可通过软件编程的形式定义和控制网络，具有控制平面和转发平面分离及开放性可编程的特点。
- SDN的核心理念是，希望应用软件可以参与对网络的控制管理，满足上层业务需求，通过自动化业务部署，简化网络运维。
- SDN并不是一个具体的技术，它是一种网络设计理念，规划了网络的各个组成部分（软件、硬件、转发面和控制面）及相互之间的互动关系。

Google SDN案例

- 将分布于全球的11个数据中心用SDN技术互联
2010年试点，2012年完成全网部署

- 广域网带宽利用率提升至接近100%
- 故障收敛时间从9s减低到1s

SDN的发展驱动力和优势

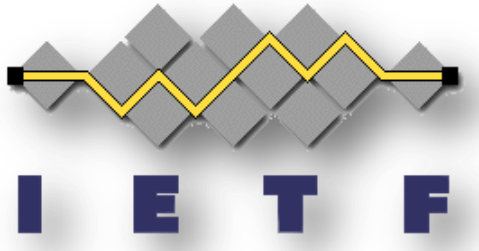
发展驱动力

- 计算虚拟化驱动：静态到动态的网络变化
- 云计算对资源的垂直整合：独立演进到协同
- 云计算时代IT业务的发展
- 数据中心资源：需要随业务跨地域整合，并使数据中心间广域流量增大

优势

- 统一便捷的管理
  - ✓ 解决网络中设备越来越多样化问题
- 无缝的版本升级
  - ✓ 解决设备版本升级对业务的影响
- 网络数据可视化
- 整体的流量调度
- ……

SDN的相关组织



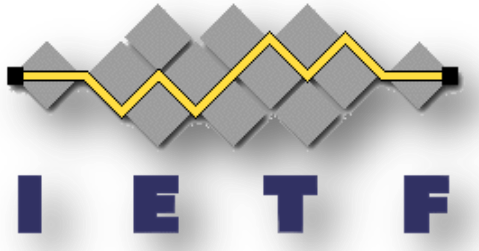
- ONF（open network foundation ）：OpenFlow

  该组织的发起者为google、facebook 、微软等由客户驱动的组织，负责推动SDN网络的部署

- IETF I2RS interface to route system

  I2RS实现路由系统的开放访问接口标准化，通过外部控制平面对设备控制平面进行扩展，也不是完全取代现有控制平面，可以实现基于控制层面的hybrid SDN

SDN的相关组织



- 国际主流运营商发起成立的ETSI： 网络功能虚拟化工作组（Network Function Virtualization，NFV）

  NFV的目标是利用当前的一些IT虚拟化技术，讲多种网络设备虚拟到大量符合行业标准的物理服务器、交换机或者存储设备上，然后在这些标准的硬件上运行各种执行这些网络功能（路由、安全功能、负载均衡、SBC等）

- Opendaylight

  由各软硬件厂商成立目标打造一个开源的基于SDN的平台框架

SDN技术的应用方向

数据中心

- 虚拟机联动/多租户
- 流量监视/拥塞发现/自动分流
- 可编程/自定义路由
- 自动化管理与运维

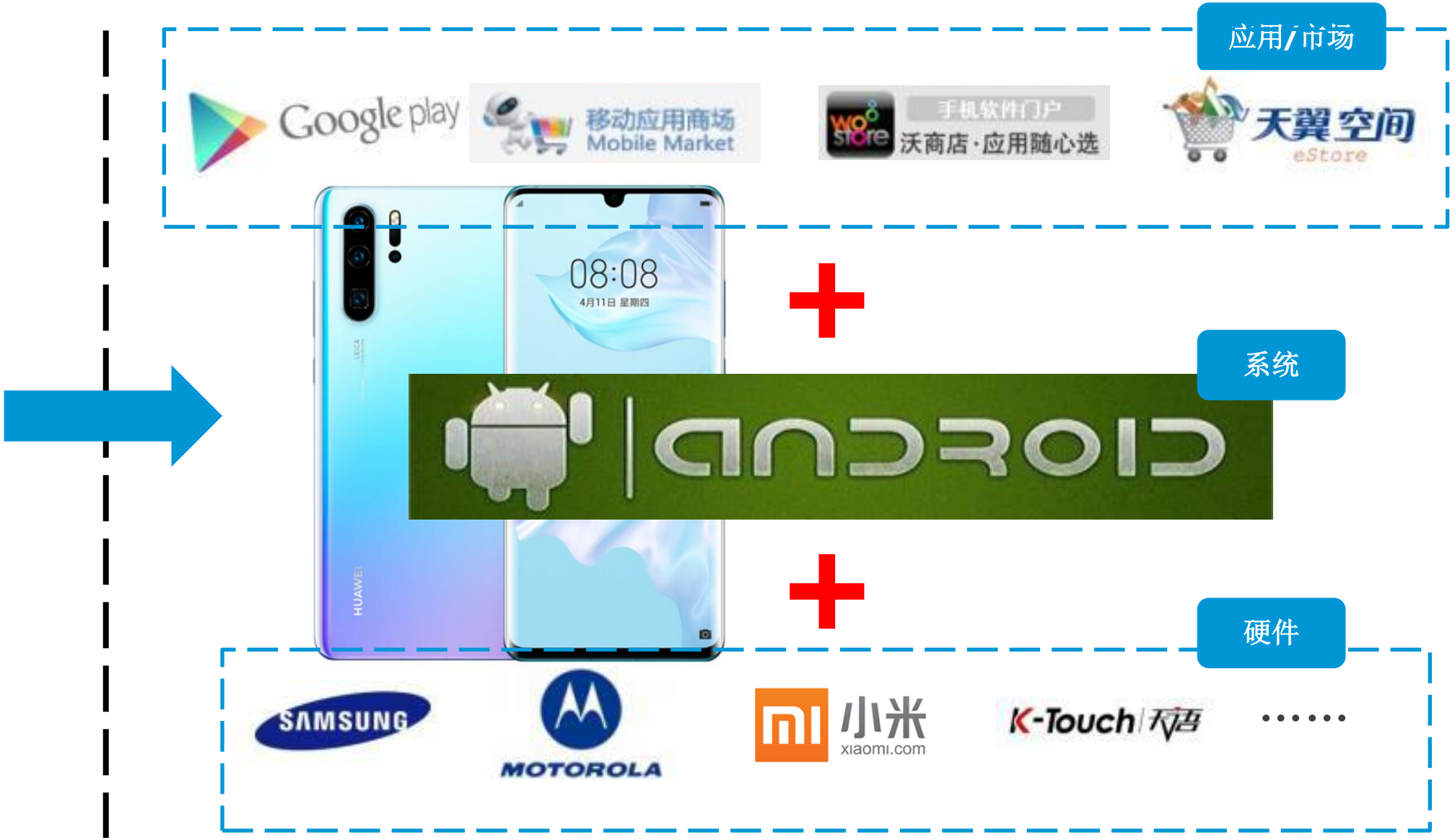
广域网

- 流量工程(TE)
- 流量识别与差异化调度
- 业务QoS自动部署与保证

无线、安全等

- 业务动态隔离及权限控制
- 统一安全防护
- BYOD
- AC云

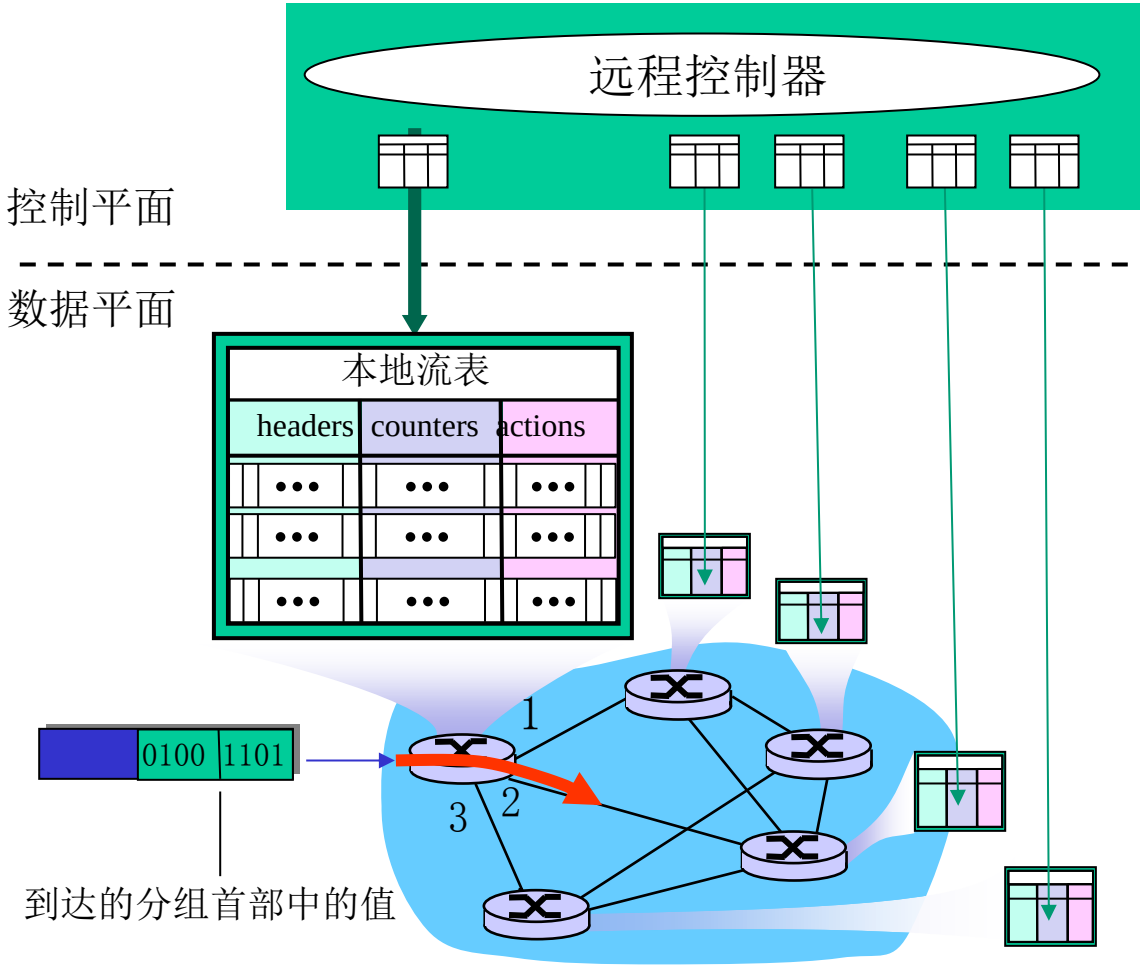
一个简单的类比


应用/市场
Google play
移动应用商场
Mobile Market
手机软件门户
沃商店·应用随心选
天翼空间
eStore
08:08
4月11日 星期四
HUAWEI
系统
ANDROID
硬件
SAMSUNG
MOTOROLA
小米
xiaomi.com
K-Touch
……

SDN：通用转发

- SDN的核心思想是建立一个通用转发体系

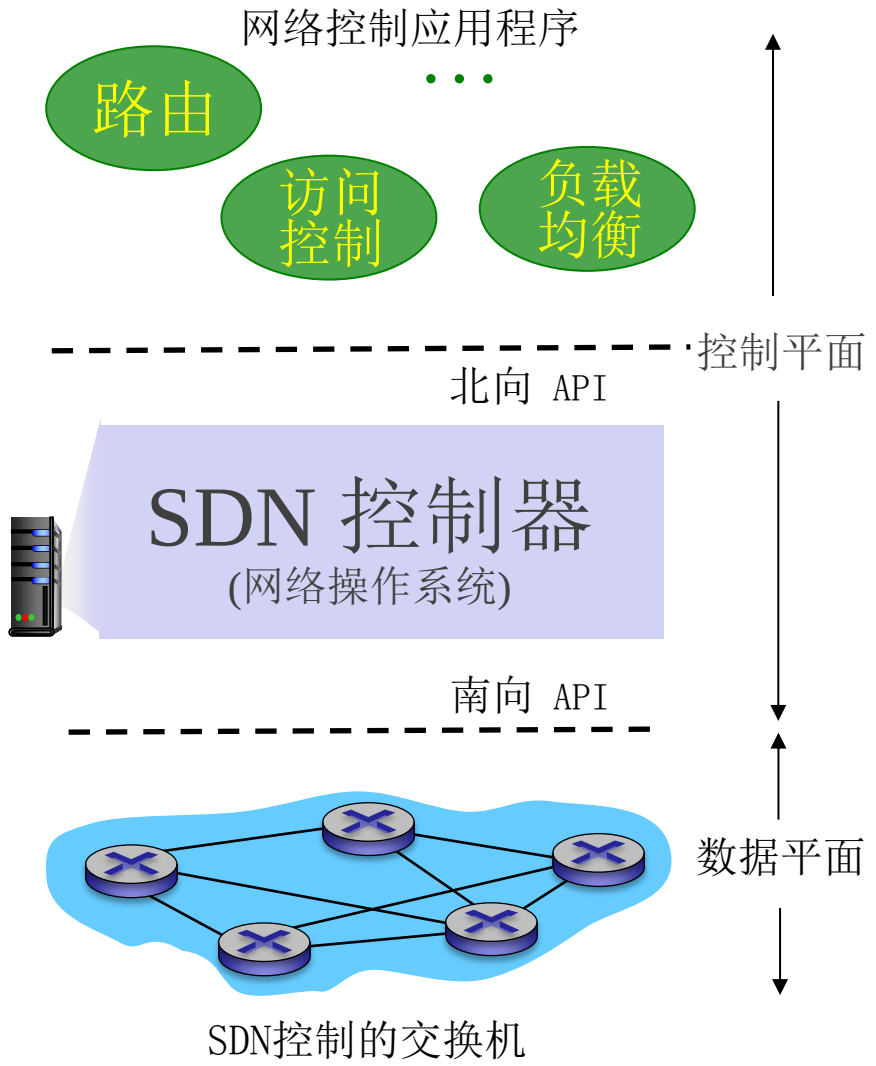
——每个交换设备包含一个流表(flow table). 流表由一个逻辑上中心化的控制器（远程控制器）来计算和分发



SDN体系结构及特征

特　征

- 基于流的转发
- 数据平面与控制平面分离
- 网络控制功能：位于数据平面交换机外部
- 可编程的网络



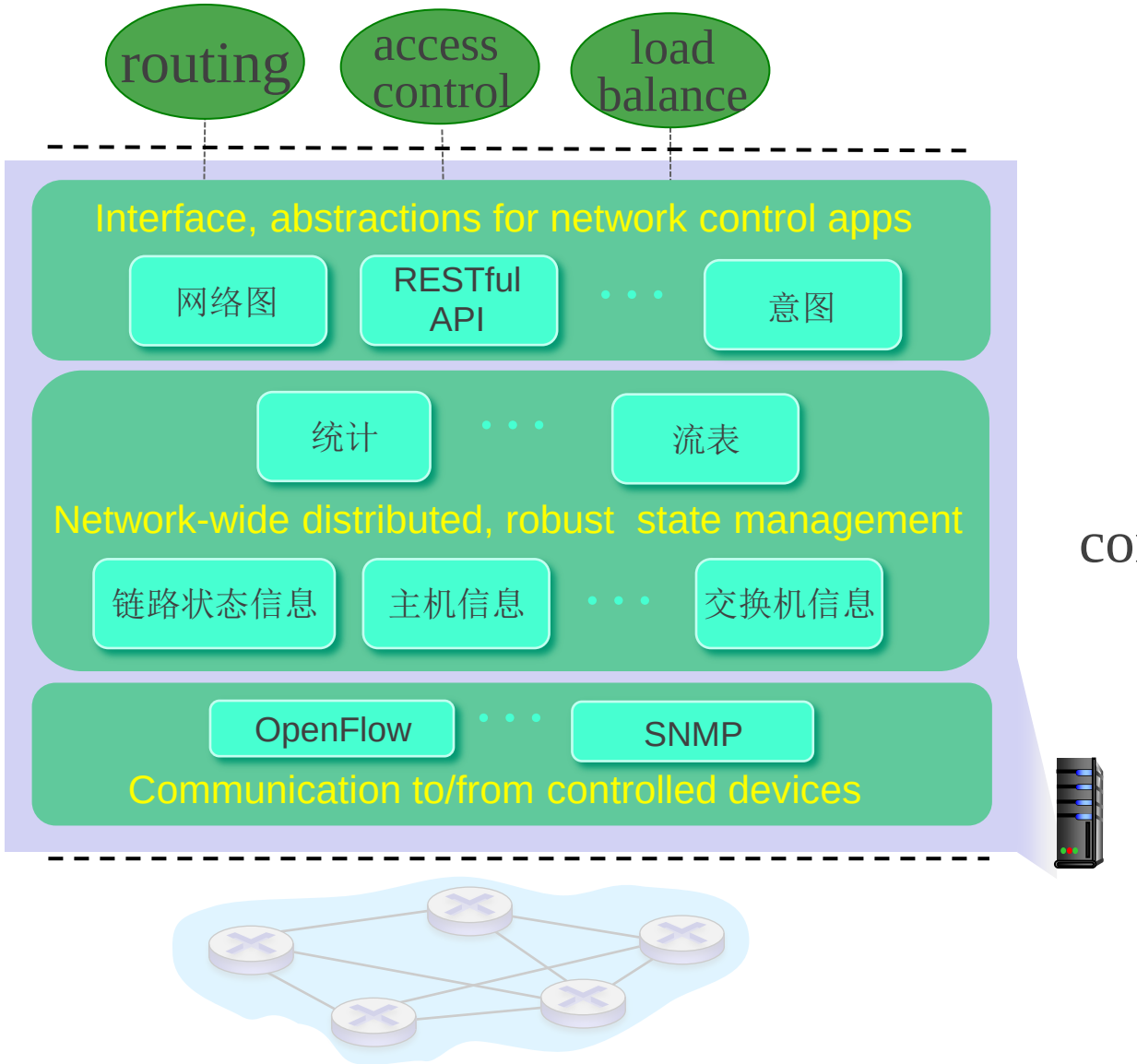
SDN体系结构

SDN控制器的组件

针对网络控制应用的接口层：抽象API

负责维护网络状态一致视图的状态管理层: 链路、交换机、服务等的状态: 一个分布式数据库

通信层： 在SDN控制器和受控交换设备间通信



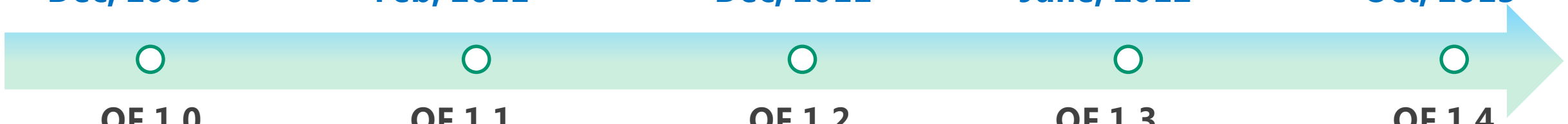
SDN controller

OpenFlow

- 起源于斯坦福大学的Clean Slate项目，目前得到广泛使用和认可
- 是SDN中控制器与转发层之间的通信接口标准
- 允许直接访问和操作网络设备的转发平面
- 分离控制平面和数据平面，二者间使用标准的协议通信
- 数据平面采用基于流的方式进行转发
- OpenFlow网络由 OpenFlow网络设备（OpenFlow Switch）和控制器（OpenFlow Controller）通过OpenFlow通道（OpenFlow Channel）组成

OpenFlow版本演进

OPEN NETWORKING FOUNDATION



功能：

- 单表
- IPv4

- 多表
- MPLS、VLAN
- group
- ECMP

- IPv6

- 重构了能力协商
- PBB
- IPv6扩展头
- 多Controller
- Per flow meter

- 增强协议扩展性
- Flow Monitoring
- Eviction
- Bundles

核心思想： OpenFlow 交换机基于流进行转发。同时，传统的控制层面从转发设备中剥离出来，“迁移”到集中控制器上

OpenFlow消息(不考)

- OpenFlow使用TCP在控制器和交换设备之间交换消息（可选加密）
- 三种消息类型：

Controller-to-Switch	Asynchronous	Symmetric
• Features: 获取交换机特性 • Configuration: 配置 OpenFlow 交换机 • Modify-State: 修改交换机状态修改流表） • Read-States: 用来获取交换机状态 • Send-Packet: 用来发送数据包 • Barrier 阻塞消息	• Packet-in 用来告知控制器交换机接收到数据包 • Flow-Removed 用来告知控制器交换机的流表被删除 • Port-Status y用来告知控制器交换机端口状态更新 • Error 用来告知控制器交换机发生错误	• Hello 用来建立 OpenFlow 连接 • Echo 用来确认控制器与交换机之间的连接状态 • Vendor 厂商自定义消息
控制器到交换机 此类消息是由控制器主动发出	异步消息 此类消息由交换机主动发出	对称消息 控制器和交换机都可以发起

本章小结

- 传统的路由算法
  - LS
  - DV
- 传统的路由协议
  - OSPF
  - BGP
  - RIP
- SDN