网络层：路由算法与协议

刘志敏

liuzm@pku.edu.cn

提纲

- 路由算法
  - 泛洪 Flooding
  - 最短路径算法 Dijkstra Algorithm
  - 距离矢量算法 Bellman-Ford Algorithm
- 路由协议
  - RIP
  - OSPF

网络中的路由选择

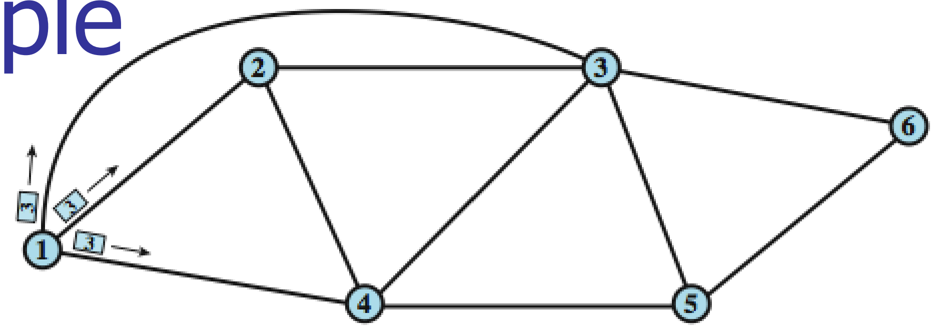
- 根据分组的目的地址选择路径
  - 数据报方式，每个分组要在途径的节点上被单独选路；
  - 虚电路方式，在建立连接时要进行选路
- 路由选择考虑的主要因素：
  - 正确性、简洁性、稳健性（处理故障以及高负载）、公平性、最优性（获得最大的平均吞吐量）、有效性
  - 性能评估准则：用于路由选择，一般为最小代价，最简单的为最小跳数

Flooding——泛洪

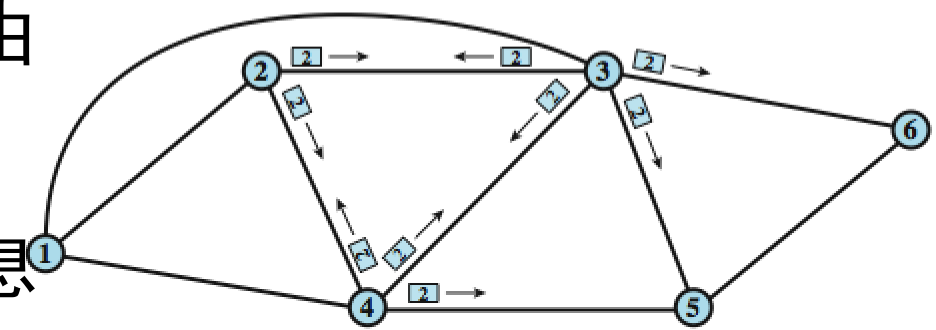
- 不需要网络信息
- 由源结点将分组发送给其邻近结点
- 结点接收到分组后，在除接收链路之外的所有链路上转发
- 最终，分组的多个备份将到达目的结点
- 每个分组有唯一的序号，以消除重复的分组
- 结点可以记住哪个分组曾被转发过，将网络负载控制在一定范围
- 在分组中可以包含有关跳数的信息

Flooding Example

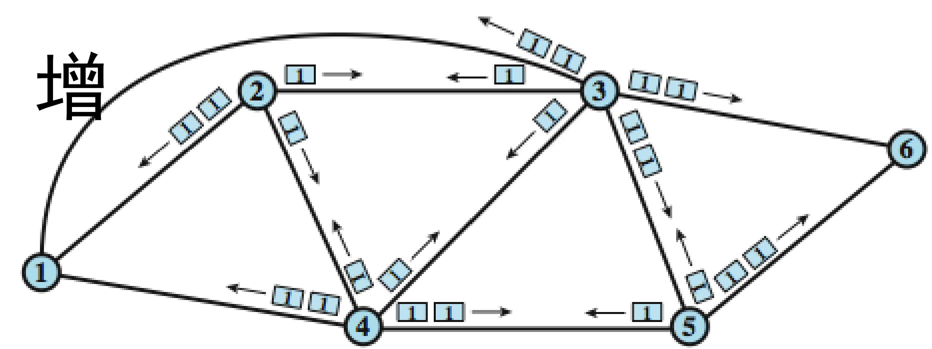
- 尝试所有可能的路由
  - 很稳健
- 可以获得最小跳数路由
  - 可用于建立虚电路
- 可以获得所有结点信息
  - 用于分组转发
- 多次复制并转发分组，增加了网络负载



(a) First hop



(b) Second hop

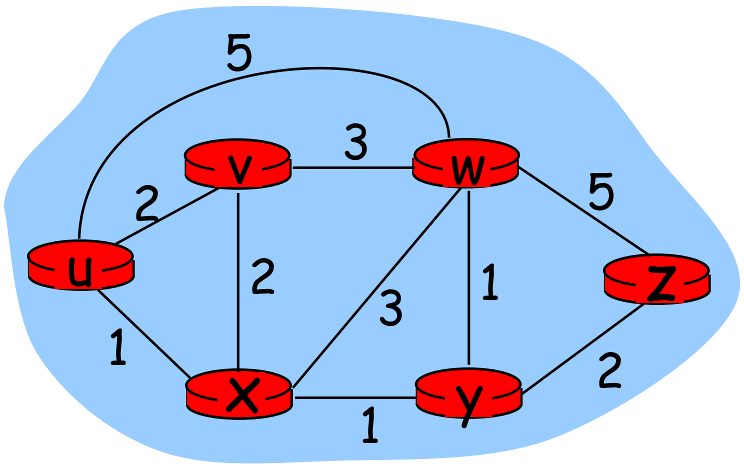


(c) Third hop

用图来描述网络

图 G = (N,E)，其中，N = 结点（路由器）集合 = { u, v, w, x, y, z }，E = 链路集合=

{ (u,v), (u,x), (u,w),(v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) }



- c（x, x'） = 链路（x, x'）的代价

例如，c（w, z） = 5

- 代价可为1，或与带宽、延迟、拥塞有关

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

问题：从u到z的最小代价路径是？代价是？

路由算法：用于寻找最小代价路径的算法

路由选择分类

集中或分布式?

集中:

- 路由器有全部的拓扑及链路代价的完整信息
- "Dijkstra" 算法

分布式:

- 路由器知道其邻居节点，以及到邻居节点的代价
- 计算：与邻居节点交换信息，通过迭代过程逐渐获得最小代价的路径
- "Bellman-Ford"算法

静态或动态?

静态:

- 路由随时间的变化缓慢

动态:

- 路由随时间的变化很快
  - 周期性更新
  - 随链路代价的改变而变化

最短路径算法

Dijkstra (迪杰斯特拉)算法

- 所有节点已知网络拓扑及链路代价
  - 通过"链路状态广播”获得
  - 所有节点具有相同信息
- 计算从一个源节点到其他所有节点的最小代价路径
  - 给出对应节点的转发表
- 迭代：经过k次迭代，获得到k个目的节点的最小代价路径

最短路径算法

定义

- c（x, y）： 从结点x到y的链路代价
  - c$(i, i) = 0$;
  - c$(i, j) = \infty$ 若两结点不直连；
  - c$(i, j) > 0$ 两结点直连
- D（v）：从源结点到结点v当前的最小代价路径的代价，随着迭代而变化
- p（v）：从源结点到结点v沿最小代价路径的前一结点
- N‘： 为算法处理的结点集合，若到结点v的最小路径已知，则v在N‘中

最短路径算法

```
Initialization:
  N' = {u}
  对所有节点 v
    if v 为 u 的邻居节点
        then D(v) = c(u,v)
    else D(v) = ∞

Loop
  找到不在 N‘中节点 w， 且 D(w) 最小
  将 w 加入 N'
  用所有与w相邻但不在 N‘中的节点v 更新D(v):
      D(v) = min( D(v), D(w) + c(w,v) )
  /* 是旧的，或是到w的代价加上从w到v的代价*/
until all nodes in N'
```

最短路径算法(1)

```
#define MAX_NODES 1024                    /* maximum number of nodes */
#define INFINITY 1000000000               /* a number larger than every maximum path */
int n, dist[MAX_NODES][MAX_NODES];        /* dist[i][j] is the distance from i to j */

void shortest_path(int s, int t, int path[])
{ struct state {                          /* the path being worked on */
      int predecessor;                    /* previous node */
      int length;                         /* length from source to this node */
      enum {permanent, tentative} label;  /* label state */
  } state[MAX_NODES];

  int i, k, min;
  struct state *p;

  for (p = &state[0]; p < &state[n]; p++) {      /* initialize state */
      p->predecessor = –1;
      p->length = INFINITY;
      p->label = tentative;
  }
  state[t].length = 0;  state[t].label = permanent;
  k = t;                                         /* k is the initial working node */
  do {                                           /* Is there a better path from k? */
      for (i = 0; i < n; i++)                    /* this graph has n nodes */
          if (dist[k][i] != 0 && state[i].label == tentative) {
              if (state[k].length + dist[k][i] < state[i].length) {
                  state[i].predecessor = k;
                  state[i].length = state[k].length + dist[k][i];
              }
          }
```

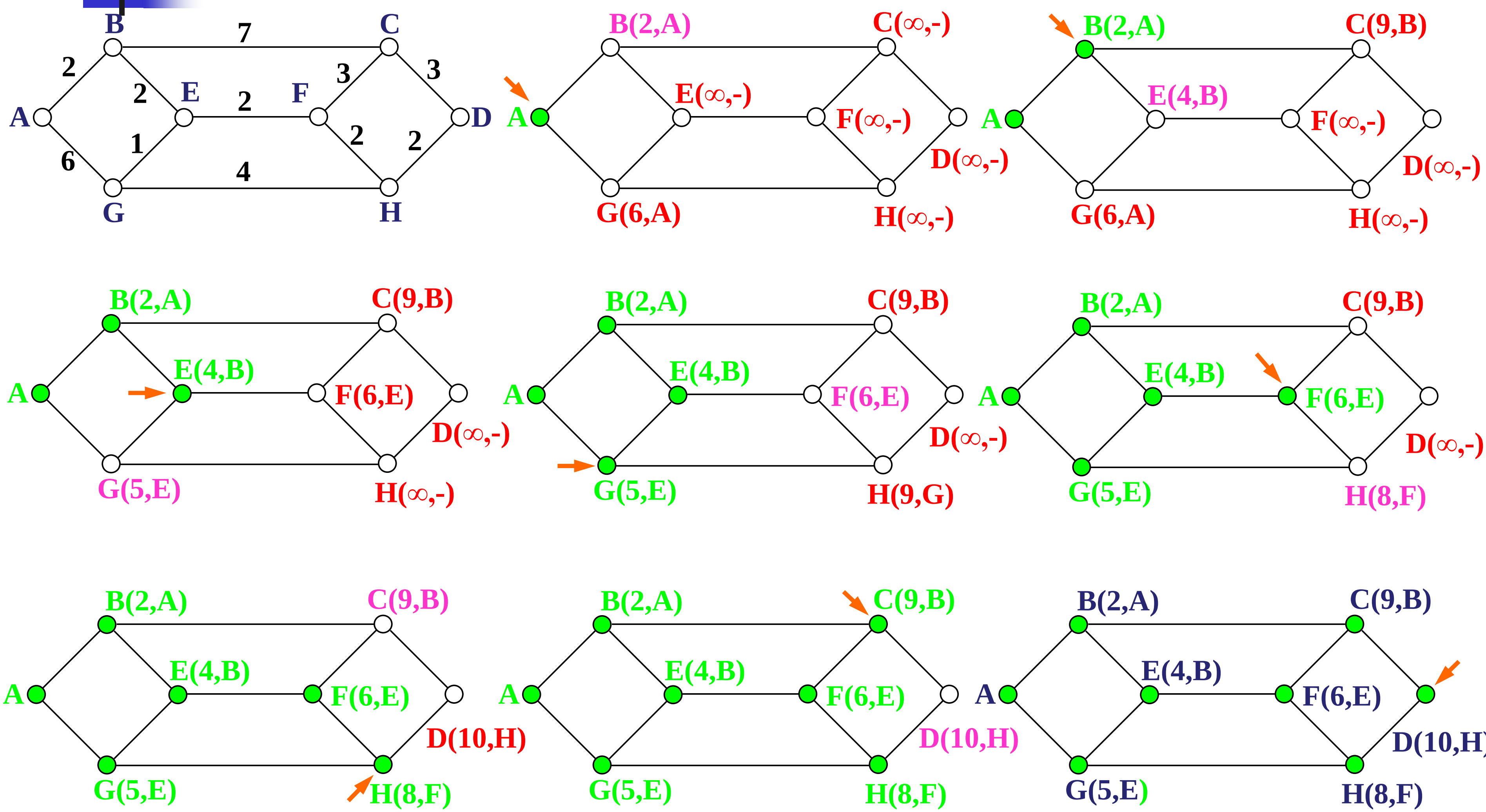
最短路径算法(2)

. . .

```
        /* Find the tentatively labeled node with the smallest label. */
        k = 0; min = INFINITY;
        for (i = 0; i < n; i++)
             if (state[i].label == tentative && state[i].length < min) {
                  min = state[i].length;
                  k = i;
             }
        state[k].label = permanent;
   } while (k != s);

   /* Copy the path into the output array. */
   i = 0;  k = s;
   do {path[i++] = k; k = state[k].predecessor; } while (k >= 0);
}
```

利用最短路径算法求A到D的最短路径

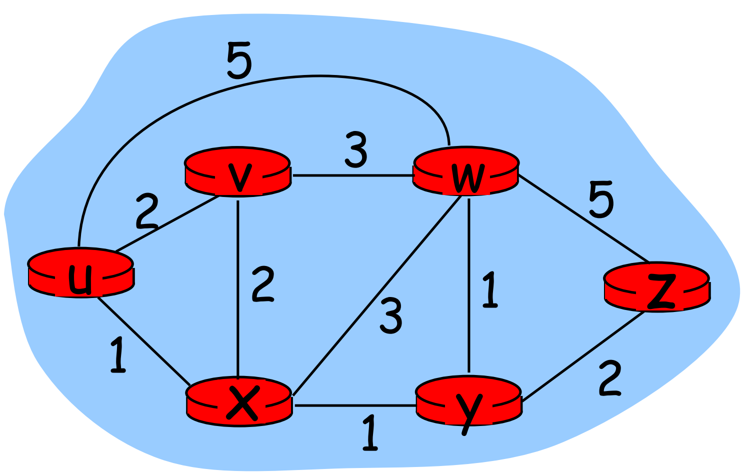


绿色：已处理结点N‘；紫色：更新的结点
红色：未处理的结点；B（DV，PV）

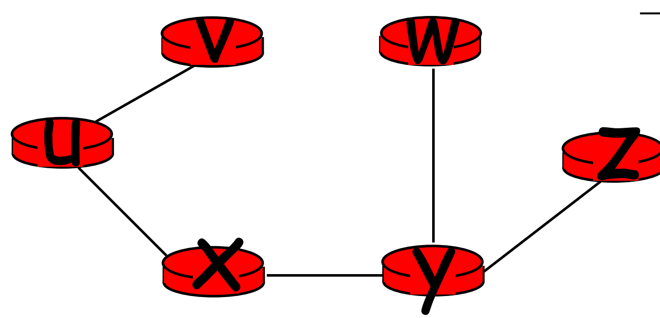
最短路径为A-B-E-F-H-D
代价为10

最短路径算法举例：求u到其他节点的最短路径及其代价

Step	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					



u的最短路径树



u上的转发表：

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

距离矢量（Distance Vector）算法

- 定义$d_x(y)$为从x到y的最小代价
- 遍历x的所有邻居节点v，得到经v到y的最小代价

$$\boxed{d_x(y)= \min_v\{ C(x,v)+d_v(y) \}}$$

B-F方程

- 从x到y的最小代价是对x的所有邻居节点v的$C(x,v)+d_v(y)$ 的最小值。怎样能找到v？

若x希望沿最小代价路径向y发送分组，而v*是使B-F方程取得最小值的相邻节点，则x首先要向v*转发分组，即x的转发表指向v*作为到目的y的下一跳

取最小值的节点作为最短路径上的下一跳➜ 转发表

距离矢量算法

- $C(x,v)$=从x到v直连链路的代价
  - 由结点x维护$C(x, v)$
- $D_x(y)$=从x到y的最小代价
  - 由结点x维护$\mathbf{D_x} = [D_x(y): y \in N]$
- 经过x的邻结点v到y的距离矢量$\mathbf{D_v}$
  - 对每个邻居v，由x维护$\mathbf{D_v} = [D_v(y): y \in N]$
- 每个节点v周期地向其邻居结点发送D_v
  - 邻居结点更新其距离矢量
  - $D_x(y) \leftarrow \min_v\{c(x,v) + D_v(y) : y \in N\}$
- 经历一段时间，D_x收敛（网络寻找最佳路径的过程）

距离矢量算法

迭代：触发迭代的原因：

- 链路代价变化
- 收到邻居的DV更新消息

异步

- 各节点独立计算，无需同步

分布式

- 只有当其DV改变时，才发送消息通知其邻居
- 邻居再发消息通知其邻居节点

每个节点：

等待（从邻居节点获得本地链路代价改变的消息）

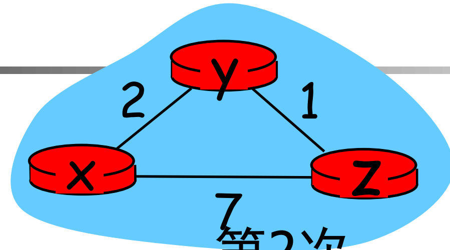
↓

重新计算 代价估计

↓

若到目的节点的DV改变了，则通知 邻居节点

练习题



距离矩阵

第1次

X

目的	X	Y	Z
X	0		
Y	2		
Z	7		

Y

目的	X	Y	Z
X		2	
Y		0	
Z		1	

Z

目的	X	Y	Z
X			7
Y			1
Z			0

第2次

X

目的	X	Y	Z
X	0	4	14
Y	2	2	8
Z	7	3	7

$+C_{xy}$ $+C_{xz}$

Y

目的	X	Y	Z
X	2	2	8
Y	4	0	2
Z	9	1	1

$+C_{yx}$ $+C_{yz}$

Z

目的	X	Y	Z
X	7	3	7
Y	9	1	1
Z	14	2	0

$+C_{zx}$ $+C_{zy}$

路由表

选最小距离及邻结点

X

目的	下一跳	距离
X	X	0
Y	Y	2
Z	Y	3

Y

目的	下一跳	距离
X	X	2
Y	Y	0
Z	Z	1

Z

目的	下一跳	距离
X	Y	3
Y	Y	1
Z	Z	0

练习题

- 初始网络有XYZ三个路由器，路由表如下。之后路由器 r 接入网络，问算法迭代多少次后路由达到稳定？给出每个路由器的距离矩阵和路由表。

路由表

X

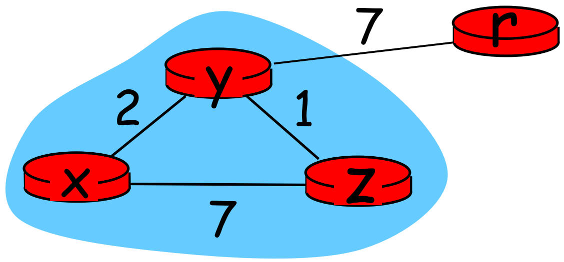
目的	下一跳	距离
X	X	0
Y	Y	2
Z	Y	3

Y

目的	下一跳	距离
X	X	2
Y	Y	0
Z	Z	1

Z

目的	下一跳	距离
X	Y	3
Y	Y	1
Z	Z	0



提纲

- 路由算法
  - 泛洪 Flooding
  - 最短路径算法 Dijkstra Algorithm
  - 距离矢量算法 Bellman-Ford Algorithm
- 路由协议
  - RIP：基于DV
  - OSPF：基于链路状态路由

路由信息协议RIP
(Routing Information Protocol)

- RIP 是一种基于距离矢量的分布式路由协议。
- 每个路由器维护其到每个目的网络的距离记录。
- 定义：路由器到直连网络的距离为1；到非直连网络的距离为经过的路由器数加1。“距离”也称为“跳数”
- 只选择一个具有最少跳数的路由。
- 一条路径最多只包含15个路由器，“距离”为16时则表示不可达。
- 仅与相邻路由器交换路由表信息，路由表的信息：

  目的网络　跳数　下一跳路由器
- 按固定的时间间隔交换路由信息，例如每隔30秒。

距离矢量算法

收到相邻路由器地址为X的RIP报文：

(1) 修改此 RIP 报文中的所有项目：将“下一跳”字段中的地址改为 X，并将所有的“距离”字段的值加 1。

(2) 对RIP 报文中的每一项，重复以下步骤：

若表项中的目的网络不在路由表中，则将该项目加到路由表中。否则，

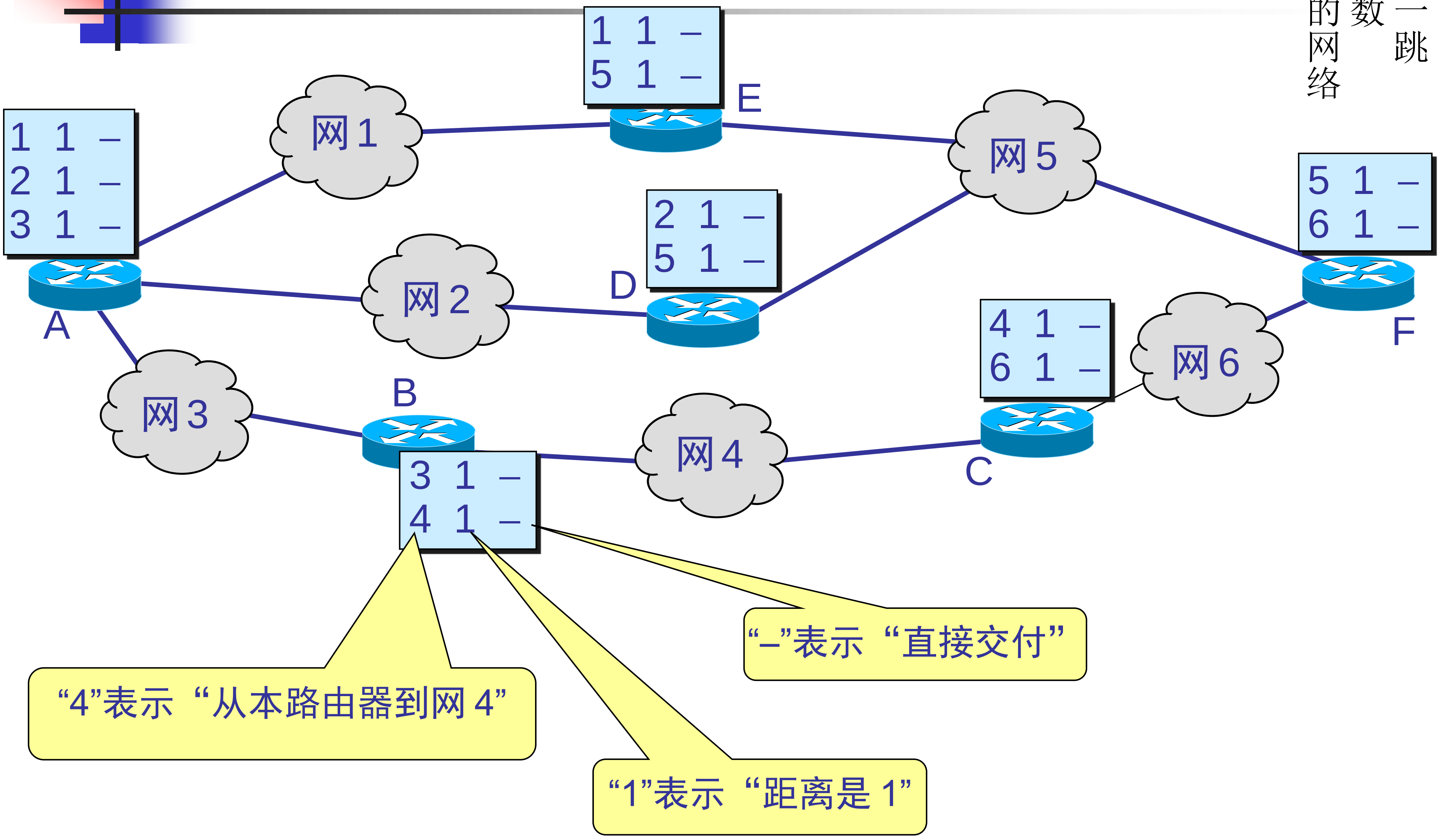
　　若下一跳路由器地址相同，则用收到的项目替换原表项

　　否则，若收到表项中的跳数更小，则更新；否则，忽略。

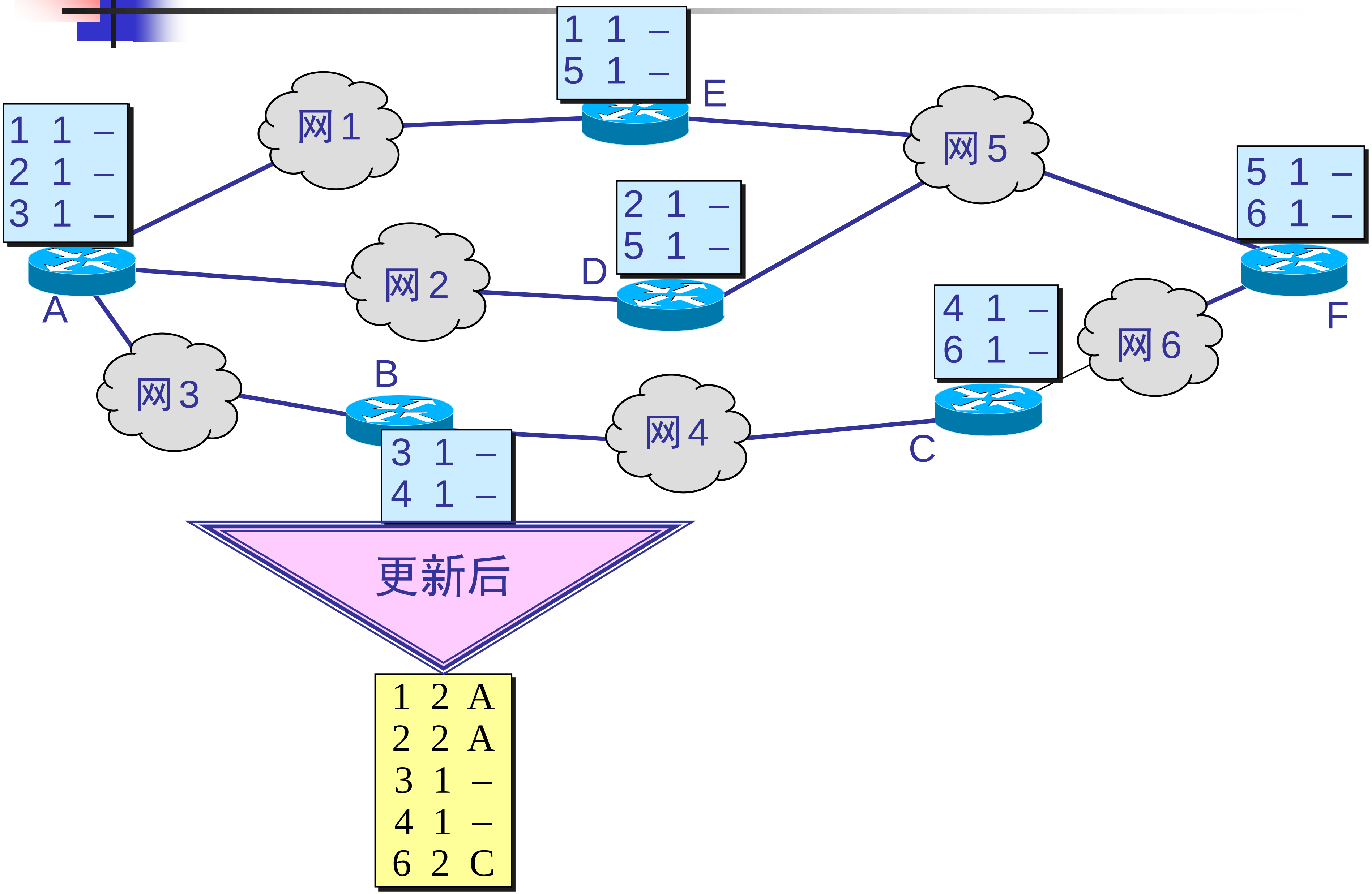
(3) 若 3 分钟还没有收到相邻路由器的更新路由表，则将此路由器记为不可达，即将距离置为16。

(4) 返回。

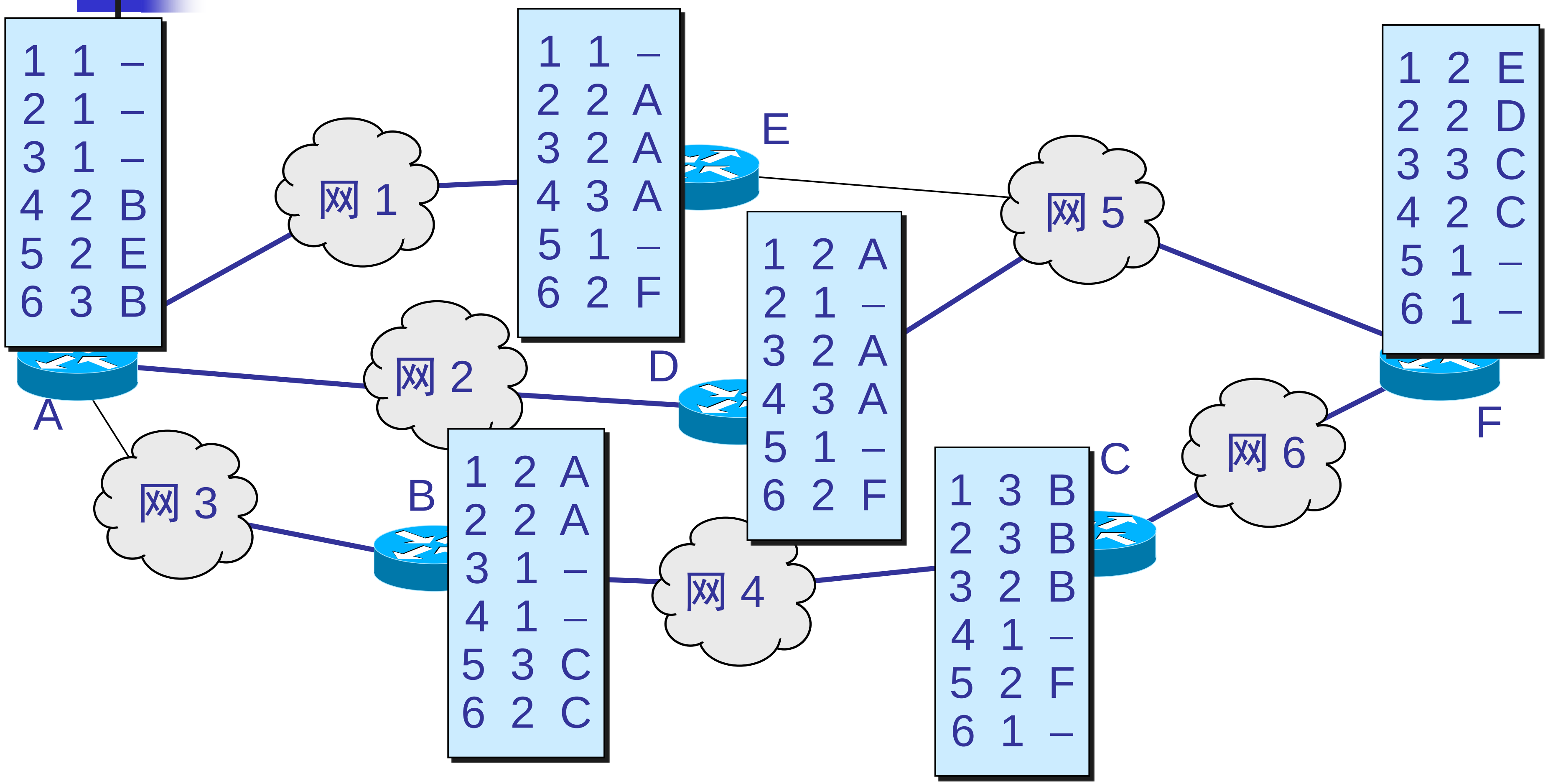
开始，各路由表只有到相邻路由器的信息



路由器 B 收到相邻路由器 A 和 C 的路由表



最终，路由器上所有的路由表都更新了

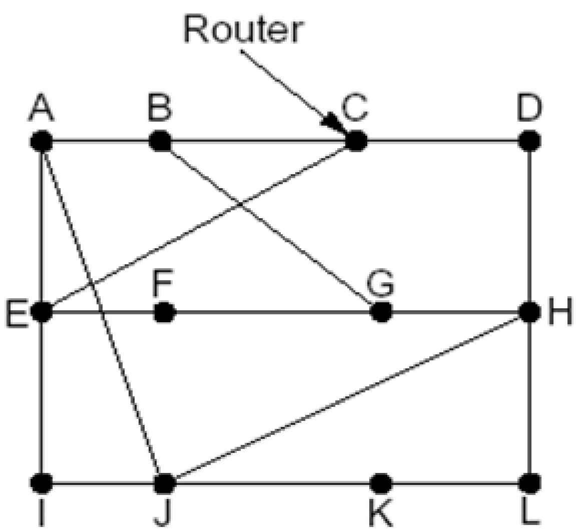


RIP协议的特点

- RIP进程间的通信使用UDP520端口
- RIP协议采用广播或组播进行路由更新，RIPv1使用广播，RIPv2使用组播224.0.0.9
- RIP协议允许主机只接收和更新路由信息而不发送信息
- RIP协议支持默认路由传播
- RIP协议的网络不超过15跳，适合于中小型网络
- RIPv1是有类（A、B、C）路由协议，RIPv2是无类(IP地址+掩码)路由协议，即RIPv2的报文中含有掩码信息

距离矢量路由

- 网络拓扑图如下图。J从邻居路由器收到延迟矢量，J测量到邻居A、I、H、K的延迟分别为8，10，12，6ms，则J的新路由表如下



To	A	I	H	K
A	0	24	20	21
B	12	36	31	28
C	25	18	19	36
D	40	27	8	24
E	14	7	30	22
F	23	20	19	40
G	18	31	6	31
H	17	20	0	19
I	21	0	14	22
J	9	11	7	10
K	24	22	22	0
L	29	33	9	9

距离矢量路由(解题过程)

$J_i=\min_j(A_i+JA, I_i+JI, H_i+JH, K_i+JK)$

其中，$i=A,B,C,\ldots,L$

转发节点：使J_i取极小值对应的j,

其中j=A,I,H,J

例如：i=E

14+8, 7+10, 30+12, 22+6;

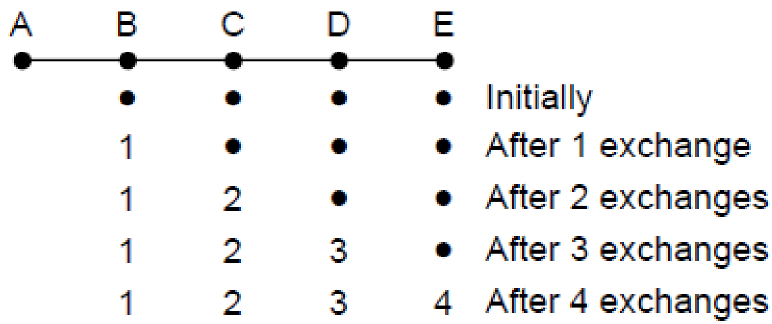
$J_i=17$, j=I

To	A	I	H	K	New routing table for J (delay)	Line
A	0	24	20	21	8	A
B	12	36	31	28	20	A
C	25	18	19	36	28	I
D	40	27	8	24	20	H
E	14	7	30	22	17	I
F	23	20	19	40	30	I
G	18	31	6	31	18	H
H	17	20	0	19	12	H
I	21	0	14	22	10	I
J	9	11	7	10	0	–
K	24	22	22	0	6	K
L	29	33	9	9	15	K
	JA delay is 8	JI delay is 10	JH delay is 12	JK delay is 6		

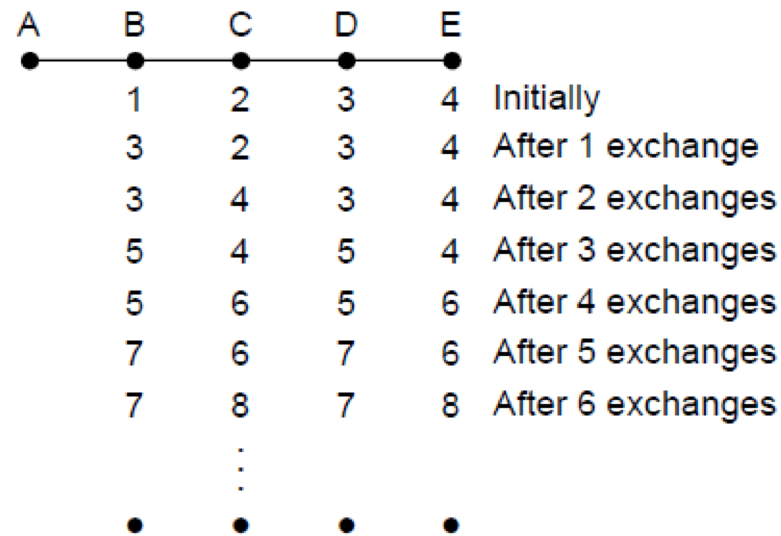
Vectors received from J's four neighbors

New routing table for J

距离矢量路由——无穷计算问题



(a)



(b)

距离矢量路由算法，存在好消息传得快、坏消息传得慢的问题例如：

a) 若A突然启动，则经过4次，收敛

b) 若A突然停机，B没有检测到，则经过N次交换，所有路由器的代价一直在更新。

解决方案：1）将无穷大的路由器的跳数设置为最大跳数+1

2）禁止路由器向邻居返回一个从邻居获得的最佳路径

链路状态路由（LS）

- 距离矢量路由的问题：需要很长时间才收敛
- “最短路径优先”：使用Dijkstra 提出的最短路径算法
- 路由器的工作过程
  - 发现邻居节点：使用HELLO消息
  - 测量链路代价
  - 创建链路状态分组
    - 何时创建？定时或发生事件时。
  - 发布链路状态分组：可靠发布
  - 计算新的路由：采用最短路径算法

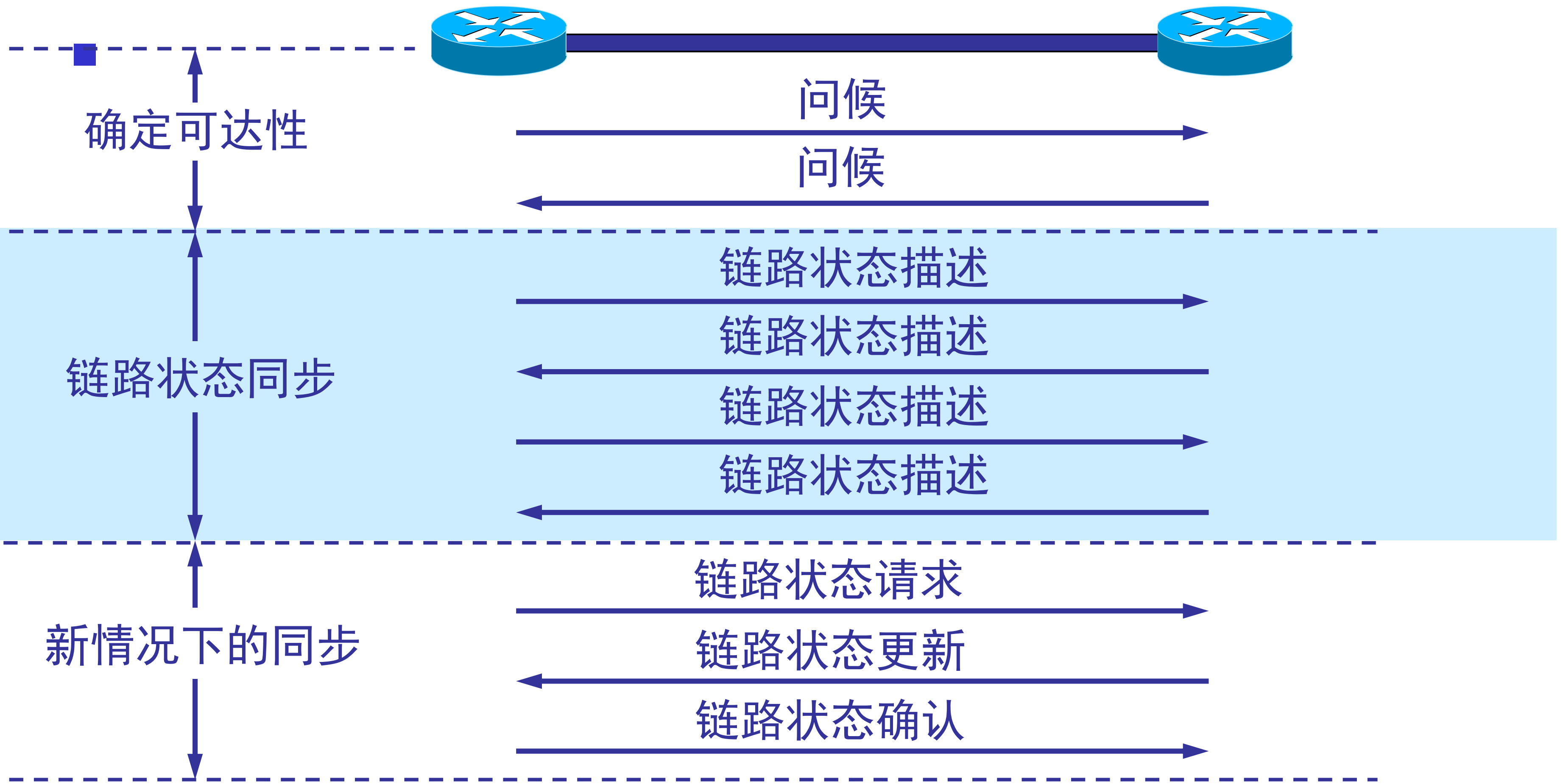
开放最短路径优先 OSPF

(Open Shortest Path First)

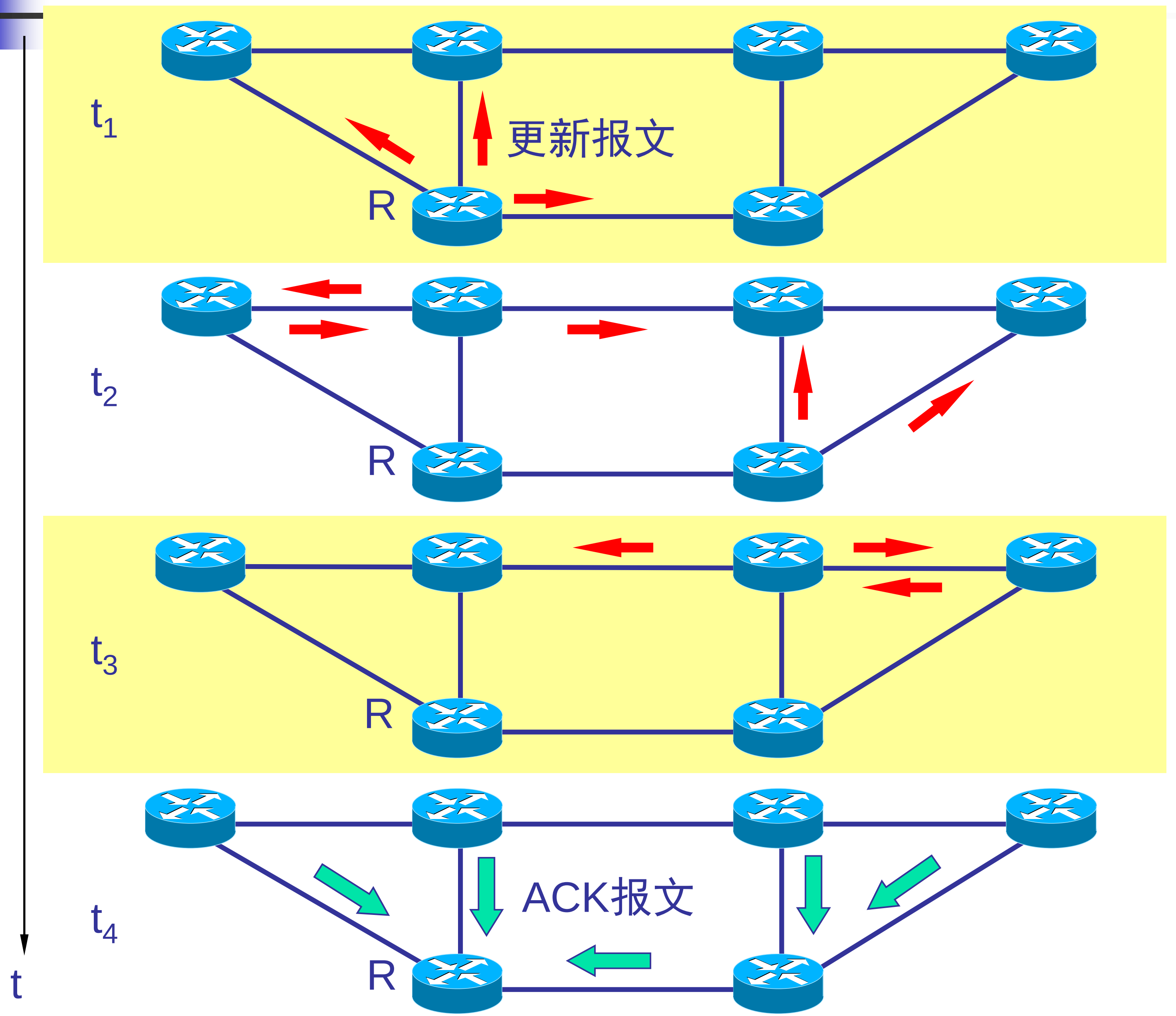
- 使用扩散法向所有（而不仅是临近）路由器发送信息与其相邻路由器的链路状态
- 当链路状态发生变化时发送信息；
- 定期地（至少每隔30分钟）发送一次
- 算法
  - （1）主动测试邻接节点的状态
  - （2）将与其相邻节点的状态信息传送给所有节点
  - （3）每个节点获得完整的网络拓扑信息，然后Dijkstra最短路径算法计算到每个节点的最佳路径

OSPF的基本操作

- 五种分组类型：问候、链路状态描述、链路状态请求、链路状态更新、链路状态确认



OSPF 使用可靠扩散法



OSPF 的特点

- OSPF 的数据报很短，减少路由信息量
- 每隔一段时间，如 30 分钟，刷新一次链路状态信息
- 由于一个路由器的链路状态只涉及与相邻路由器的连通状态，而与互联网的规模无直接关系。当互联网规模很大时，OSPF 协议要比距离向量协议 RIP好得多
- OSPF 没有“坏消息传播得慢”的问题，据统计，其响应网络变化的时间小于 100 ms

LS与DV的比较

- DV仅与邻居交换信息，但提供到其他节点的DV估计；而LS与网络所有节点交换代价信息，每个节点获得网络拓扑及链路代价的全部信息
- 1）发送的报文数：LS为N*E数量级的（N为结点数，E为链路数），只要有一个链路变化，则发送信息到全部节点；DV是在邻居节点之间交换，变化则再交换
- 2）收敛时间：LS的为N*N，DV的收敛时间受到多种因素的影响
- 3）稳健性：LS为分别计算，一个错误不会影响全局，稳定；而DV，一个节点的计算错误，会在节点之间扩散，影响整个网络，稳健性差

小结

- 地址分配：
  - IP地址，三种编址方式；
  - 如何分配IP地址？
  - IP地址数量不够如何解决？
- 分组传送
  - ARP：IP 地址到MAC的映射
  - 各段链路的帧长度不同，如何确定IP分组长度？
- 路由与转发：
  - 路由算法及协议：DV（RIP），LS（OSPF）
  - 组播、移动、自组织网络等路由算法及协议？
- 网络控制：超时控制、差错恢复、状态报告、拥塞检测与控制？