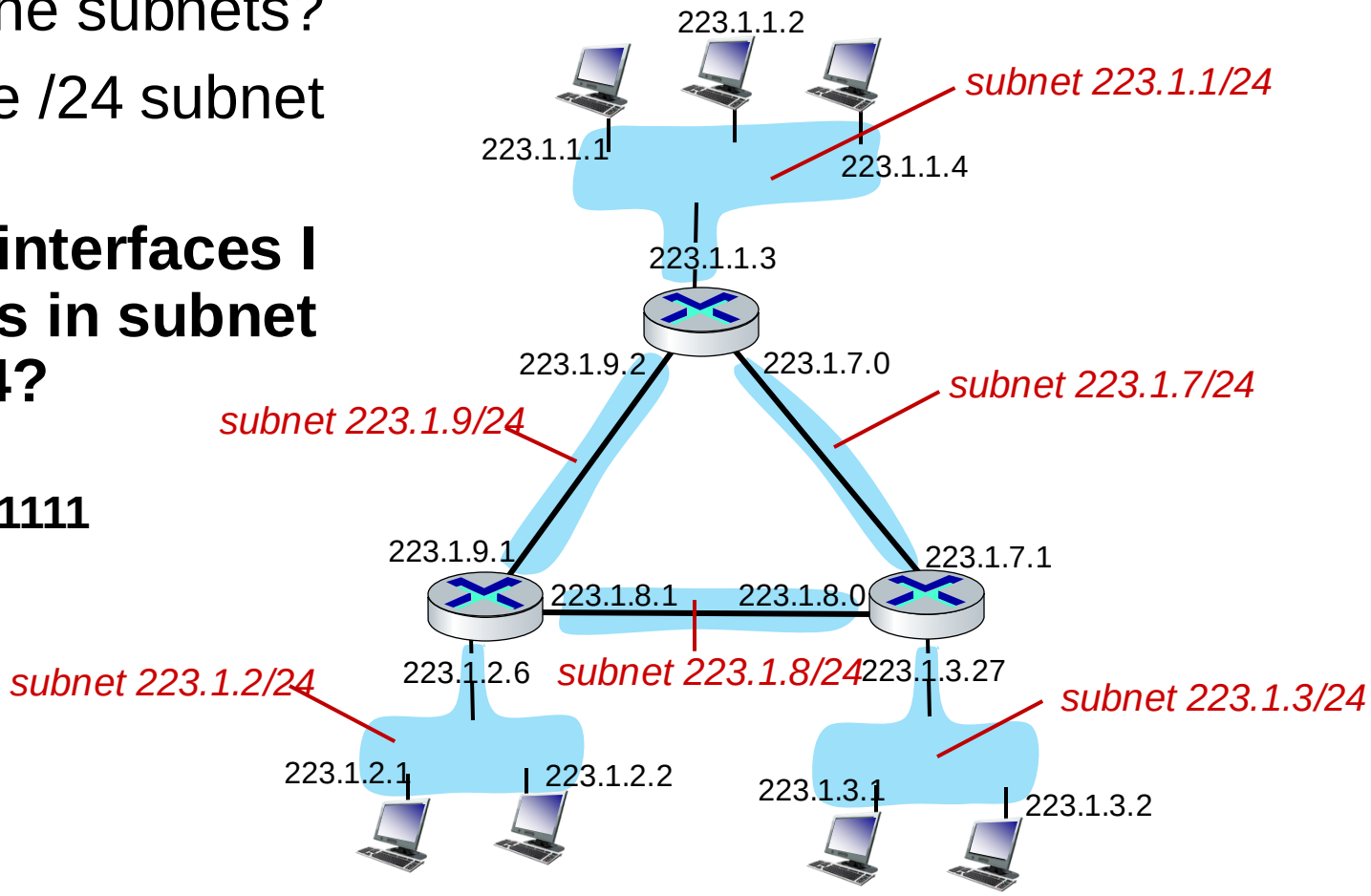
Caveat: Lecture 9 P1

- Where are the subnets?
- What are the /24 subnet addresses?
- **How many interfaces I can address in subnet 223.1.2.1/24?**

00000000 ~ 11111111

256 - 2 = 253



Lecture 12: Where we are in the textbook

5.1 Introduction

5.2 Routing algorithms

- Link state
- Distance vector

today

5.3 Intra-AS routing in the Internet: OSPF

5.4 Routing among the ISPs: BGP

next lecture

5.5 The SDN control plane

5.6 ICMP: The Internet Control Message Protocol (covered briefly in lecture 12)

5.7 Network management and SNMP

Will not cover 5.5-5.7

Network-layer functions

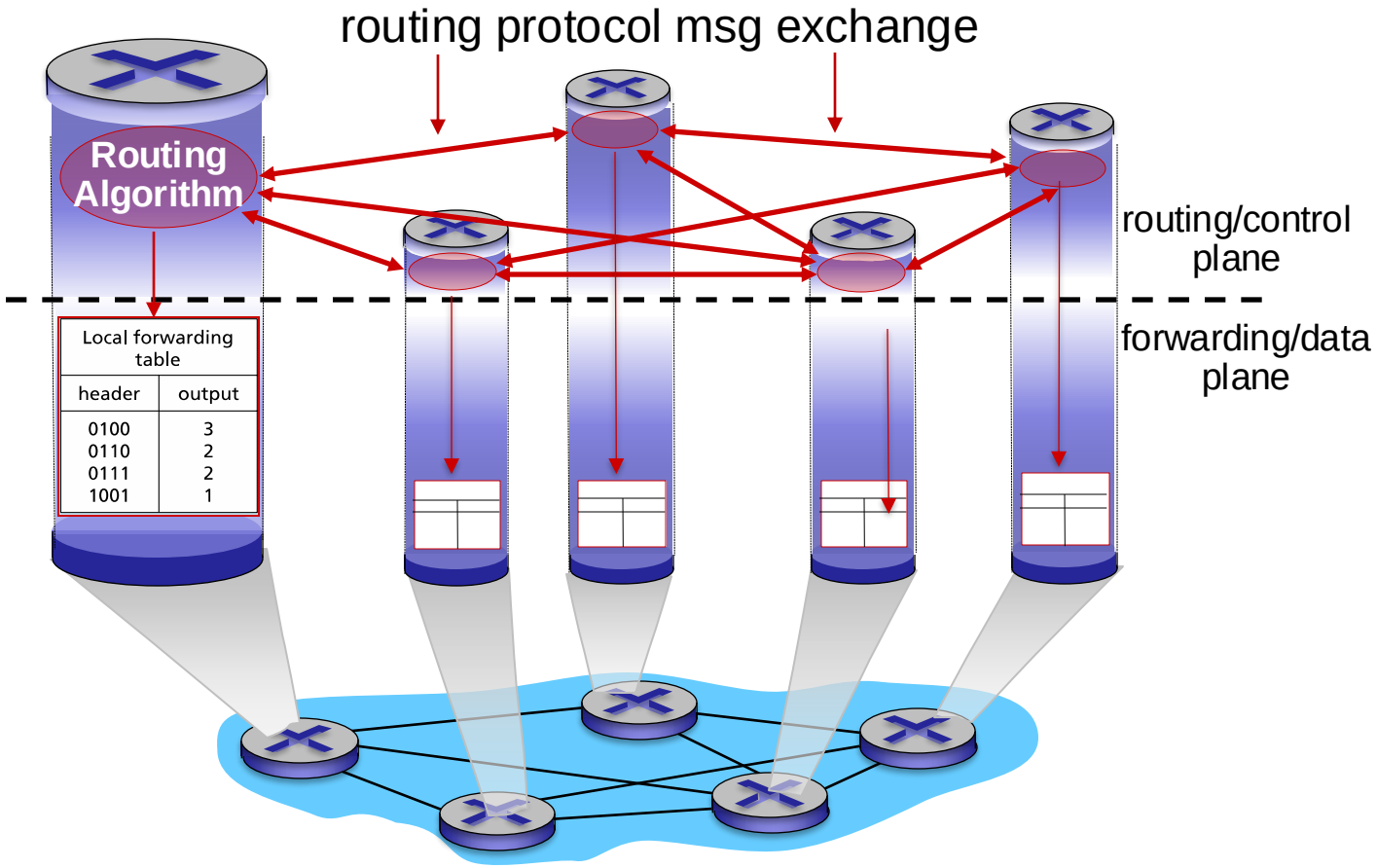
- *Forwarding*: move packets from a router's input to appropriate output interface *data plane*
- *Routing*: determine the path taken by packets to reach destinations *routing plane*, or *control plane*

Two ways to structuring network routing/control:

- per-router control (traditional)
  - Routers run routing protocol to set up forwarding table
- (logically) centralized control (more recent)
  - Software defined networking (SDN)

Per-router control plane

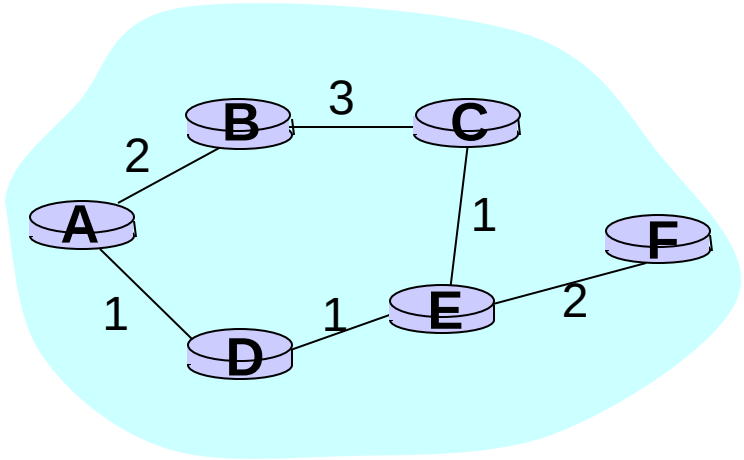
Using the info learned from other routers through *routing protocols,* each router runs the *routing algorithm* to compute forwarding table

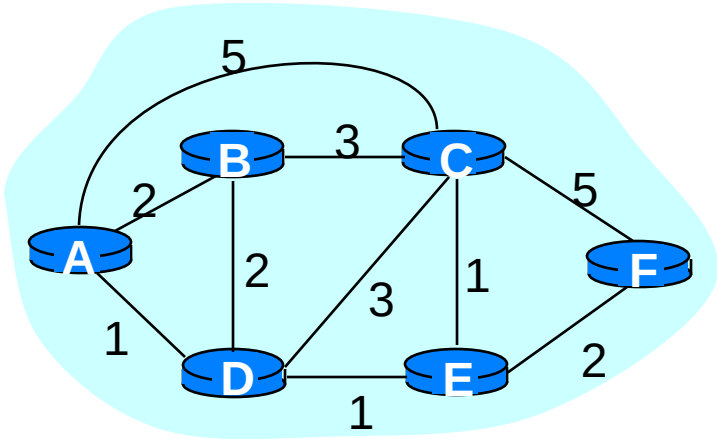


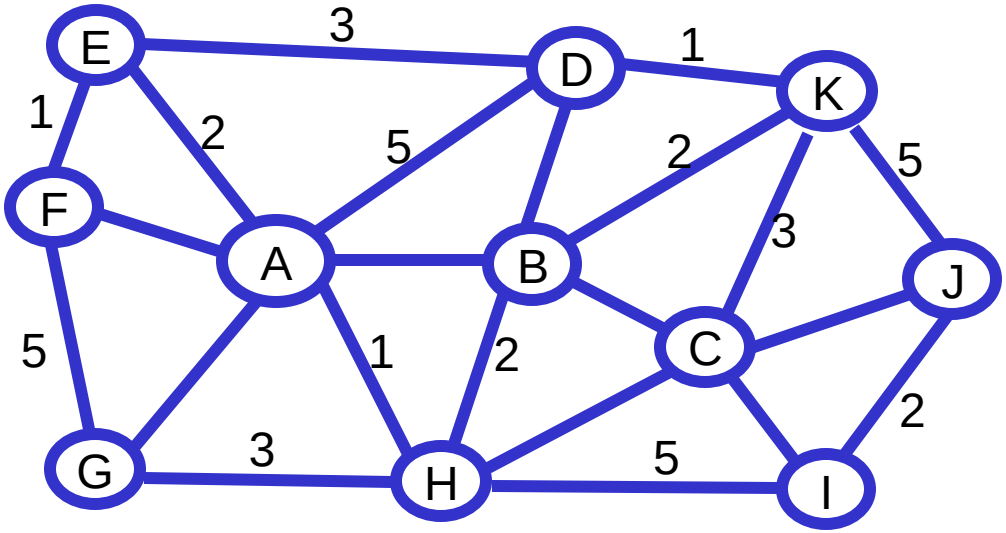
If we abstract a network as a graph,

How to find the best path to a destination?

from each node (router) to a given destination

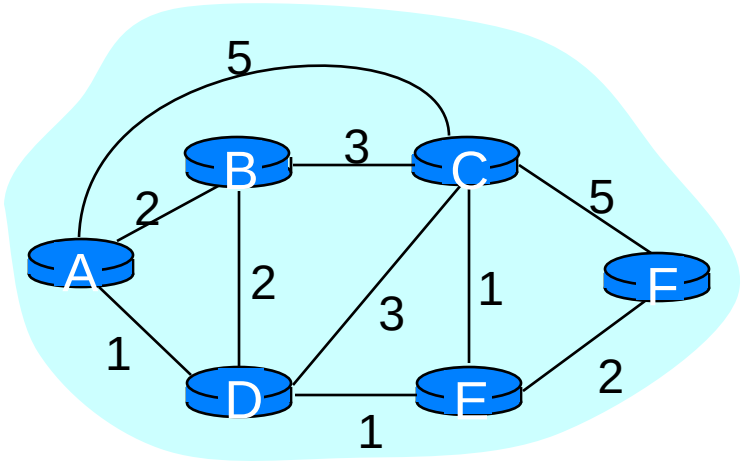






Need computation algorithm to find the best path from one point to any other point

Network Graph Abstraction



Graph: G: a set of nodes & edge

N = set of routers = { A, B, C, D, E, F }

E = set of links ={ (A,B), (A,C), (A,D), (B,C), (B,D), (C,D), (C,E), (C,F), (E,F) }

Cost of link (a, b) = C(a, b)
- e.g. C(A, B) = 2, C(B, C) = 3

Cost of path (x_1, x_2, x_3,…, x_p) = $C(x_1,x_2)+C(x_2,x_3)+ \ldots + C(x_{p-1},x_p)$

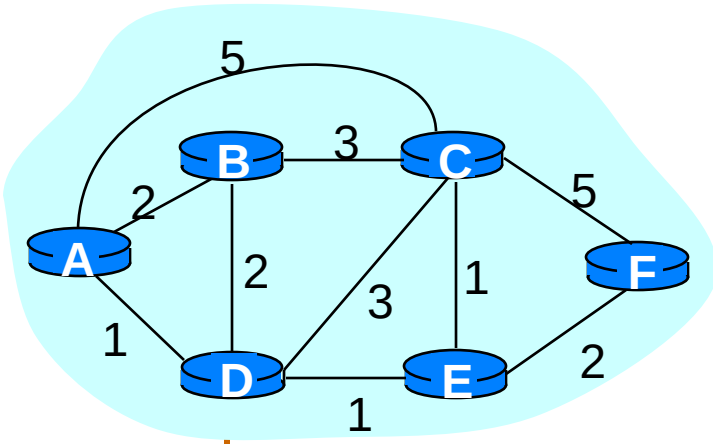
Route computation algorithm: given a graph, find least-cost path from a *given node* to *all* the other nodes in the graph

Network Routing: algorithms vs. protocols

Route computation algorithms:

link-state (Dijkstra**)**: given a complete topology graph with all nodes and link costs, each node performs its own computation of shortest paths to all destinations

distance-vector (Bellman-Ford**)**: each node knows its link cost to neighbors, and computes its shortest paths to all destinations based on the shortest paths of its neighbors

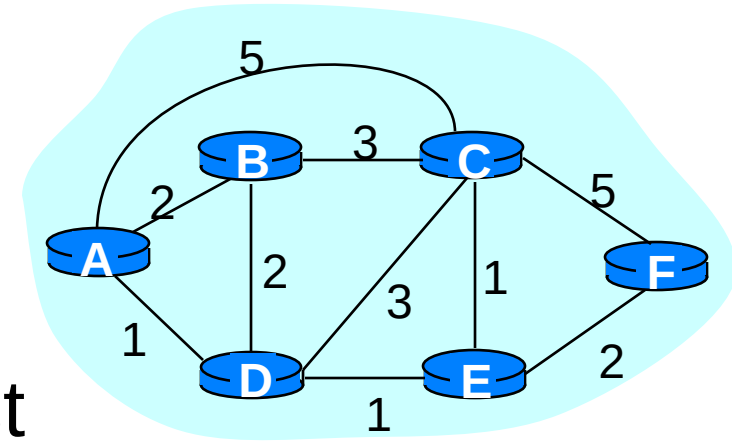


Routing protocols

- Define the format of routing information exchanges
- Define the computation upon receiving routing updates
- Network topology changes over time → continuously updates all routers with latest changes

Link-State algorithm

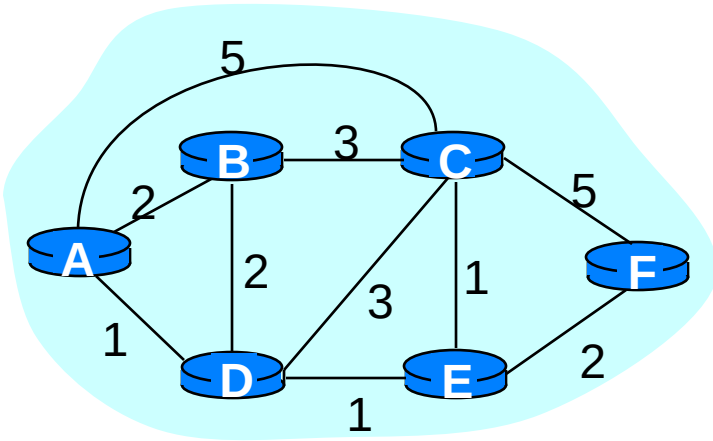
Assuming every node knows the network topology graph with link cost



- Each node computes least cost *paths* from itself to all the other nodes
  - determine *the next hop* of the best path to each destination
- Iterative operation: after k iterations, a node knows the best paths to k destinations

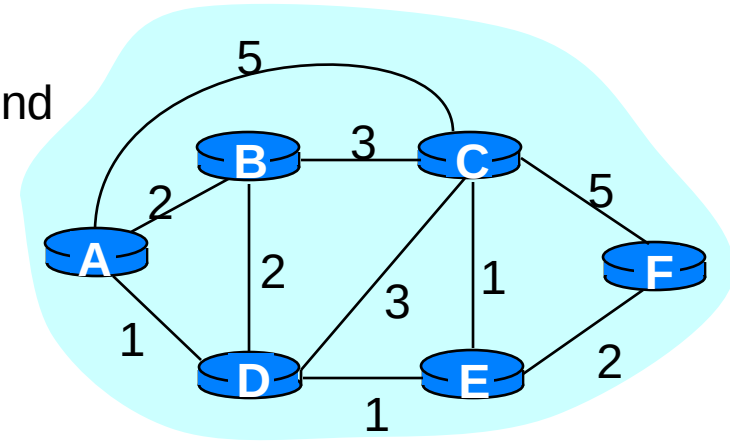
Link-State algorithm: basic notations

- c(x,y): link cost from neighbor node x to node y
- D(v): current value of path cost from source to destination v
  - Initializes to ∞ if nodes (i, j) are not direct neighbors
    - e.g. A's init. view: D(e) = D(f) = ∞
- p(v): the node right before v along best path from source to v
  - from A's view, best path from A to C is A-D-E-C, p(c)=E



Link-State algorithm

N': set of nodes whose least-cost paths has been found

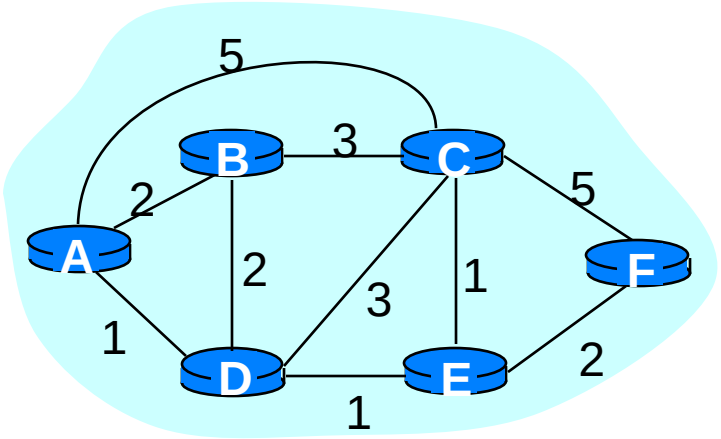


consider the computation done at node **A**

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

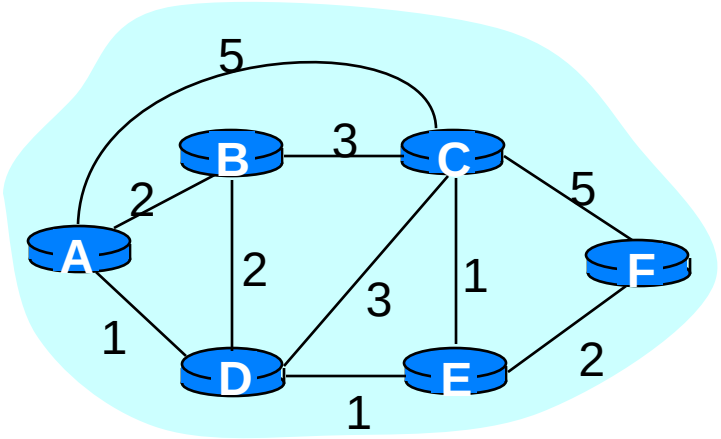
Link-State algorithm: example

Step	start N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
→ 0	A	2, A	5, A	1, A	∞	∞



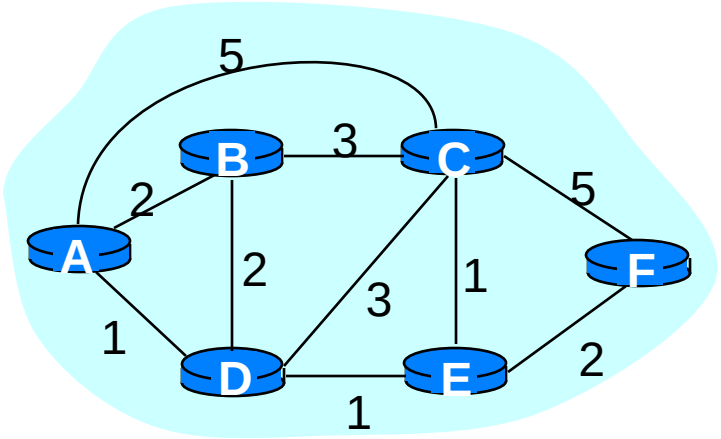
Link-State algorithm: example

Step	start N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2, A	5, A	1, A	∞	∞
1	AD	2, A	4, D		2, D	



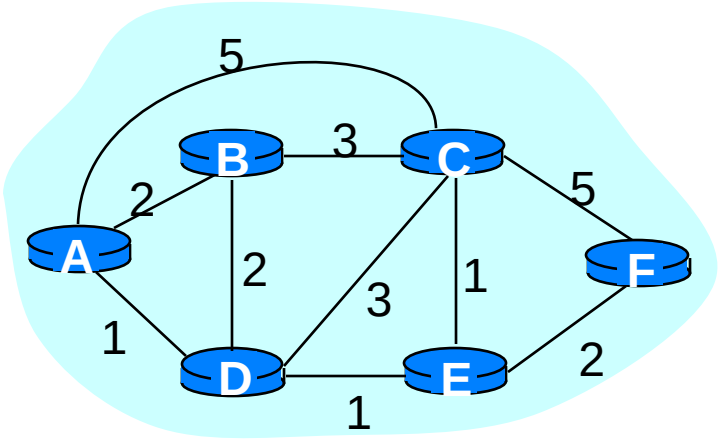
Link-State algorithm: example

Step	start N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2, A	5, A	1, A	∞	∞
1	AD	2, A	4, D		2, D	
2	ADE		3, E			4, E



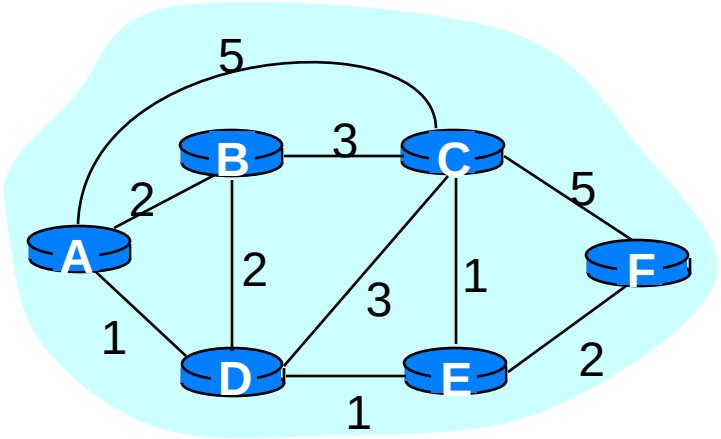
Link-State algorithm: example

Step	start N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2, A	5, A	1, A	∞	∞
1	AD	2, A	4, D		2, D	
2	ADE		3, E			4, E
3	ADEB		3, E			



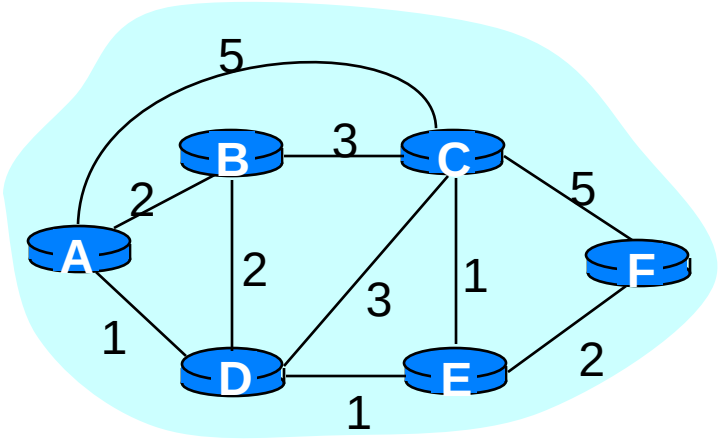
Link-State algorithm: example

Step	start N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2, A	5, A	1, A	∞	∞
1	AD	2, A	4, D		2, D	
2	ADE		3, E			4, E
3	ADEB		3, E			
→ 4	ADEBC					4, E



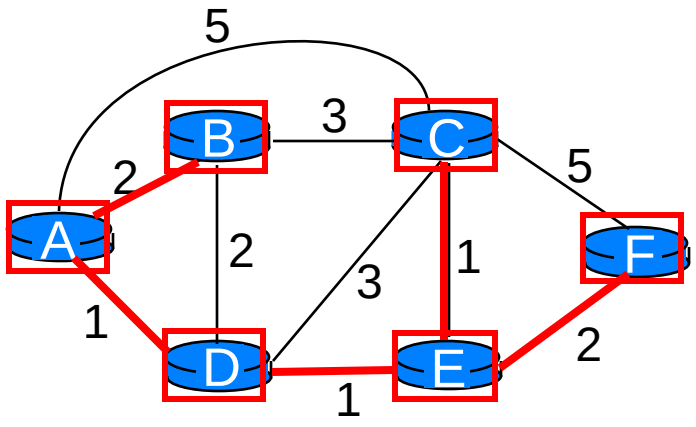
Link-State algorithm: example

Step	start N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2, A	5, A	1, A	∞	∞
1	AD	2, A	4, D		2, D	
2	ADE		3, E			4, E
3	ADEB		3, E			
4	ADEBC					4, E
5	ADEBCF					



Link-State algorithm: example

Step	start N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2, A	5, A	1, A	∞	∞
1	AD	2, A	4, D		2, D	
2	ADE		3, E			4, E
3	ADEB		3, E			
4	ADEBC					4, E
5	ADEBCF					



Destination	Next Hop
B	(A, **B**)
D	(A, **D**)
E	(A, **D**)
C	(A, **D**)
F	(A, **D**)

Resulting shortest-path tree for A:

Resulting forwarding table at A:

Link-State algorithm: discussion

Algorithm complexity: for a graph with n nodes

- each iteration: need to check all nodes, w, not in N'
- n(n+1)/2 comparisons: O(n^2)
  - more efficient implementations possible: O(nlogn)

Oscillations possible:

- e.g., link cost = amount of carried traffic

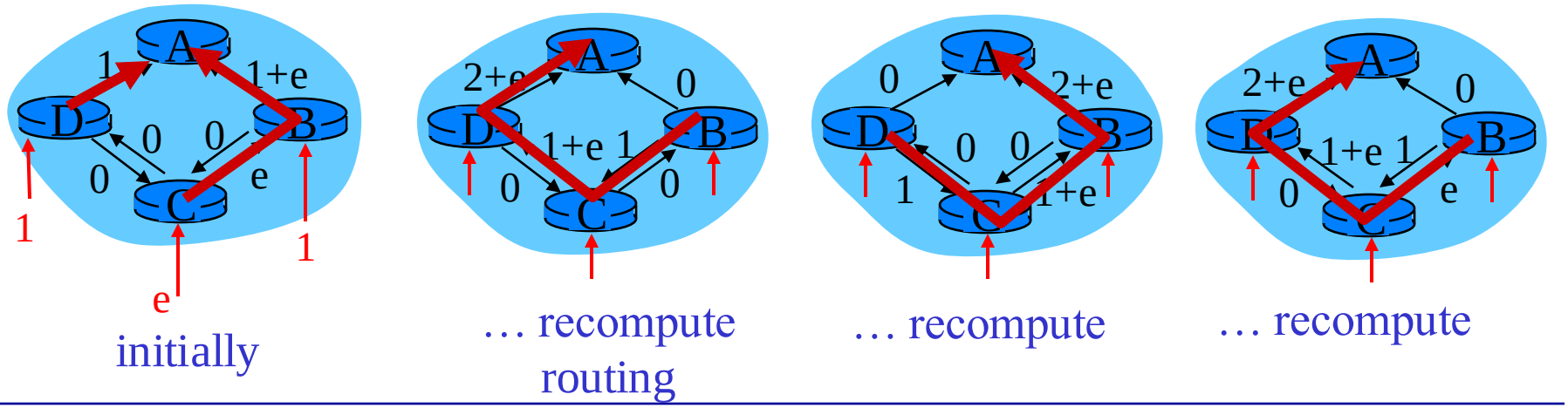
Link-State algorithm: discussion

Algorithm complexity: for a graph with n nodes

- each iteration: need to check all nodes, w, not in N'
- n(n+1)/2 comparisons: O(n^2)
  - more efficient implementations possible: O(nlogn)

Oscillations possible:

- e.g., link cost = amount of carried traffic



Distance Vector Algorithm

Summery of Link-State algorithm:

- Each node knows the complete topology graph with link costs
- Each node calculate the shortest path to all other nodes

For Distance-Vector algorithm:

- Each node know *only* needs from each direct neighbor its list of distances to all destinations
- Each node computes shortest path based on the input from all its neighbors

Distance Vector Algorithm

Bellman-Ford equation (dynamic programming)

let

dx(y) := cost of least-cost path from x to y

then

$d_x(y) = \min \{c(x,v) + d_v(y)\}$

v

cost from neighbor v to destination y

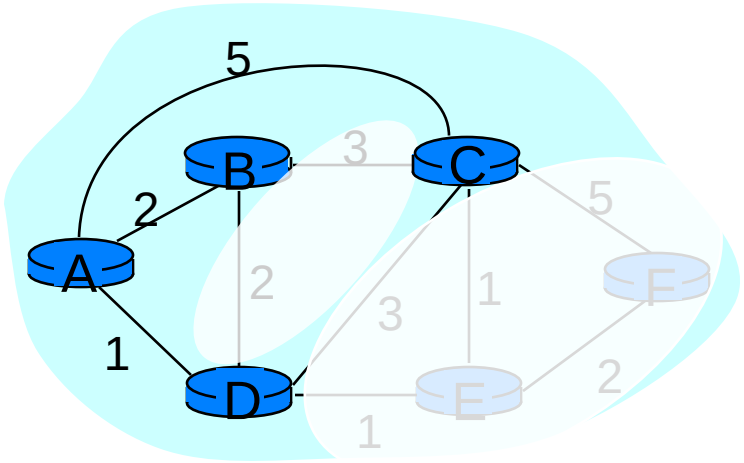
cost to neighbor v

min taken over all neighbors v of x

Distance Vector Equation

$D_x(y) = \min \{c(x,v) + D_v(y)\}$

- where min is taken over *all* neighbors v of x



$D_A(F) = \min \{c(A,B) + D_B(F),$
$c(A,D) + D_D(F),$
$c(A,C) + D_C(F)\}$
$= \min \{2 + 5,$
$1 + 3,$
$5 + 3\} = 4$

($D_B(F)=5$, $D_D(F) = 3$, $D_C(F) = 3$)

Node leading to the shortest path is next hop to be saved in the forwarding table

Distance Vector: what a node does

1. Node *x* initialization: link cost to neighbor v: c(x,v)
2. *x* maintains $\mathbf{D}_x = [D_x(y): y \in N]$
   - $D_x(y)$ = estimate of least cost from x to y
3. *x* sends its distance vector, $\mathbf{D}_x$, to all its neighbors
4. x receives $\mathbf{D}_v$ from each neighbor v, calculates
   $D'_x(y) = \min \{c(x,v) + D_v(y)\}$
   If $D'_x(y) < D_x(y)$: (going through v to reach y is a shorter path)
   - next hop to y = v (*x* picks v as next hop to reach y)
   - $D_x(y) = D'_x(y)$
   - Send out the updated $\mathbf{D}_x$ — *x* informs all its neighbors of its newly changed distance vector

Distance Vector Protocol

Iterative, asynchronous:

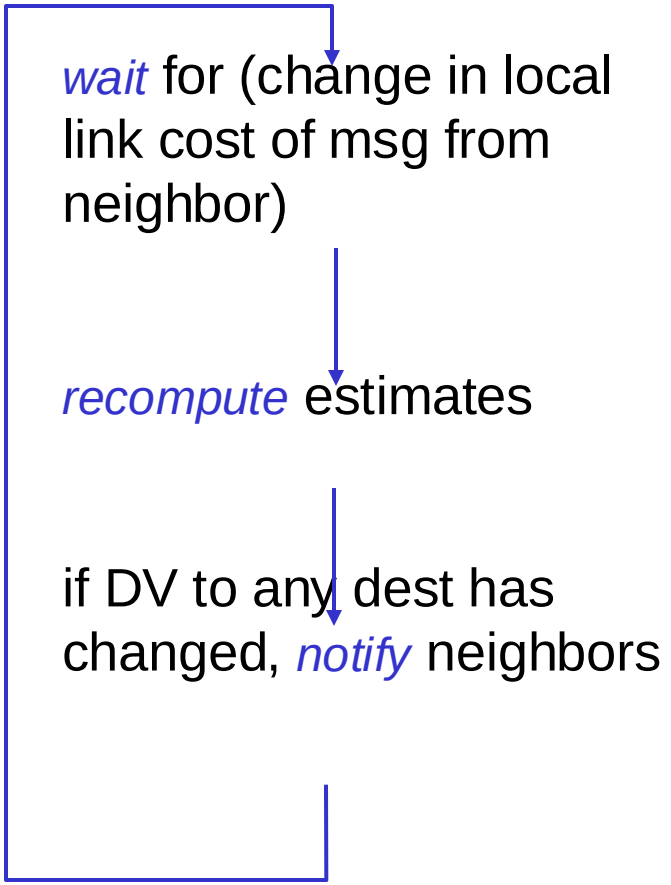
- Each local iteration caused by:
  - local link cost change
  - DV update message from neighbor
- Continues until no nodes exchange info.

Distributed:

- Each node notifies neighbors *only* when its DV changes
  - neighbors then notify their neighbors if necessary

Asynchronous: nodes need *not* Exchange info/iterate in lock step

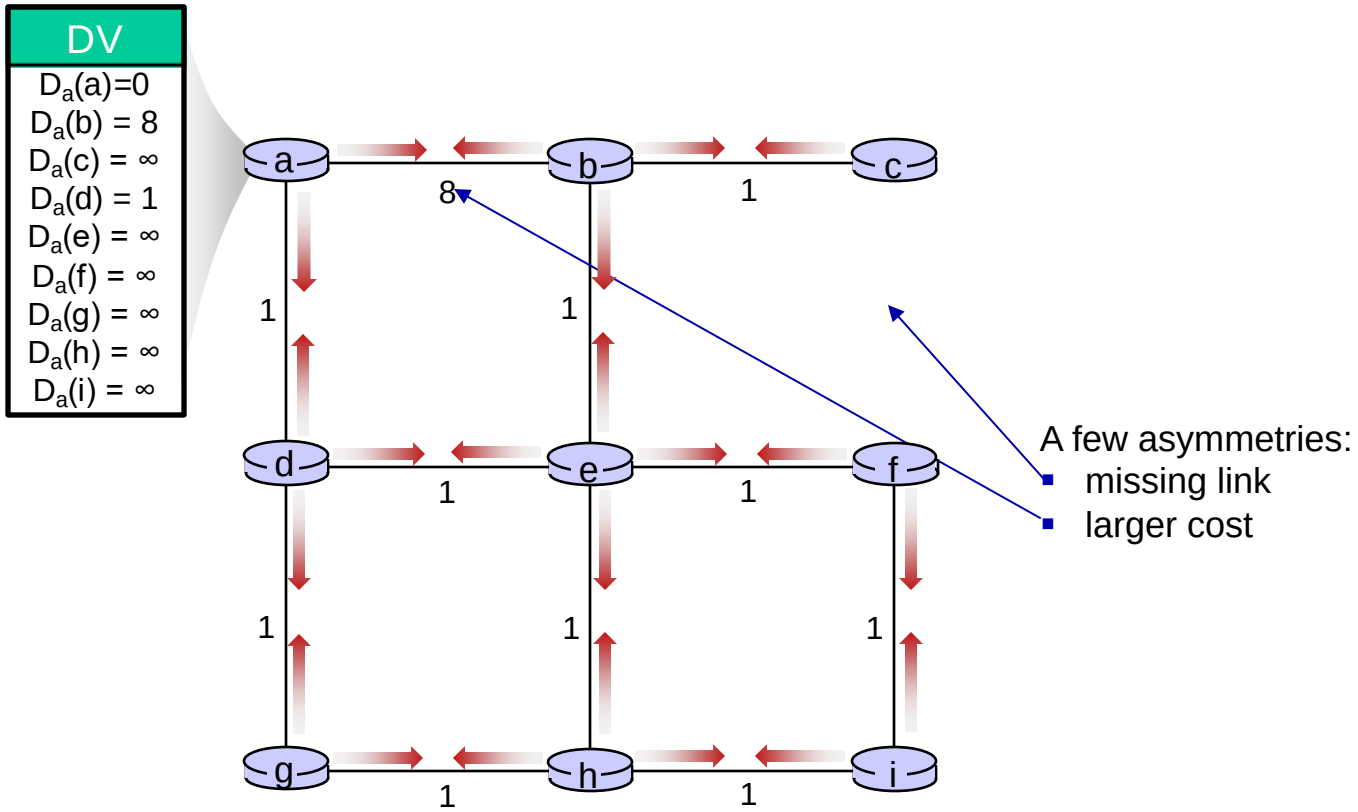
Each node:



Distance vector: example

t=0

- All nodes have distance estimates to nearest neighbors (only)
- All nodes send their local distance vector to their neighbors

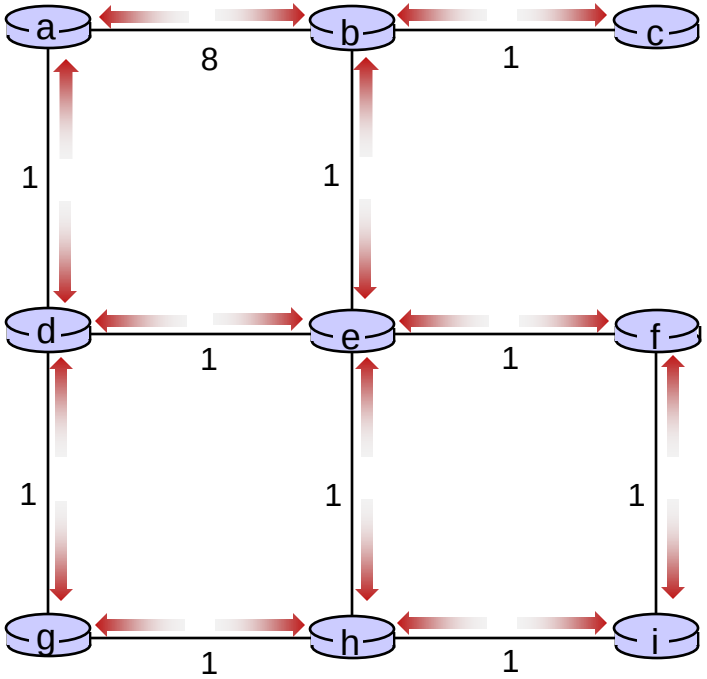


Distance vector example: iteration

t=1

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors

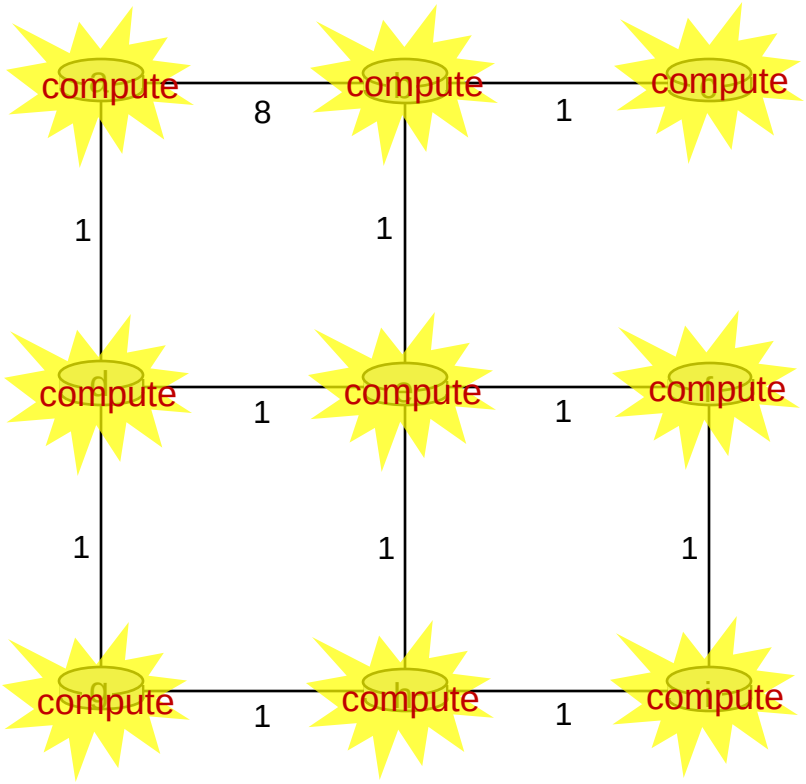


Distance vector example: iteration

t=1

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors

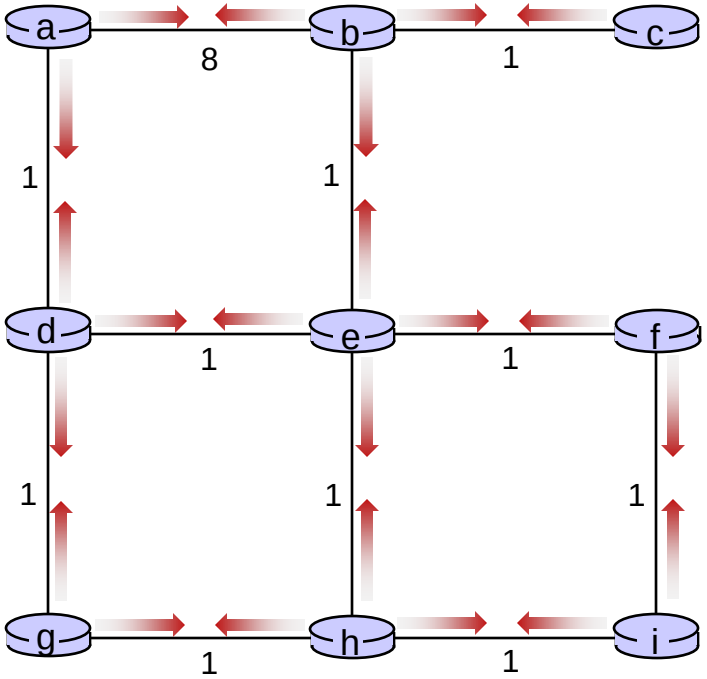


Distance vector example: iteration

t=1

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors

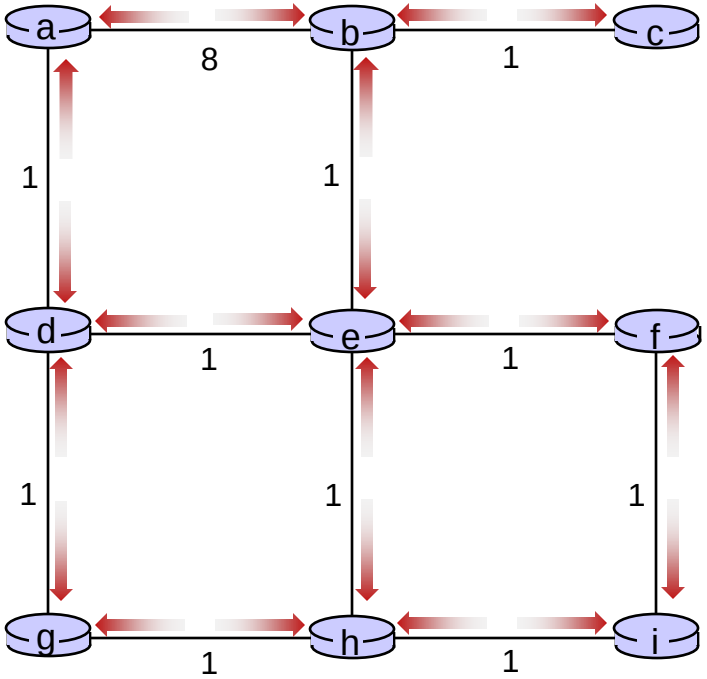


Distance vector example: iteration

t=2

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors

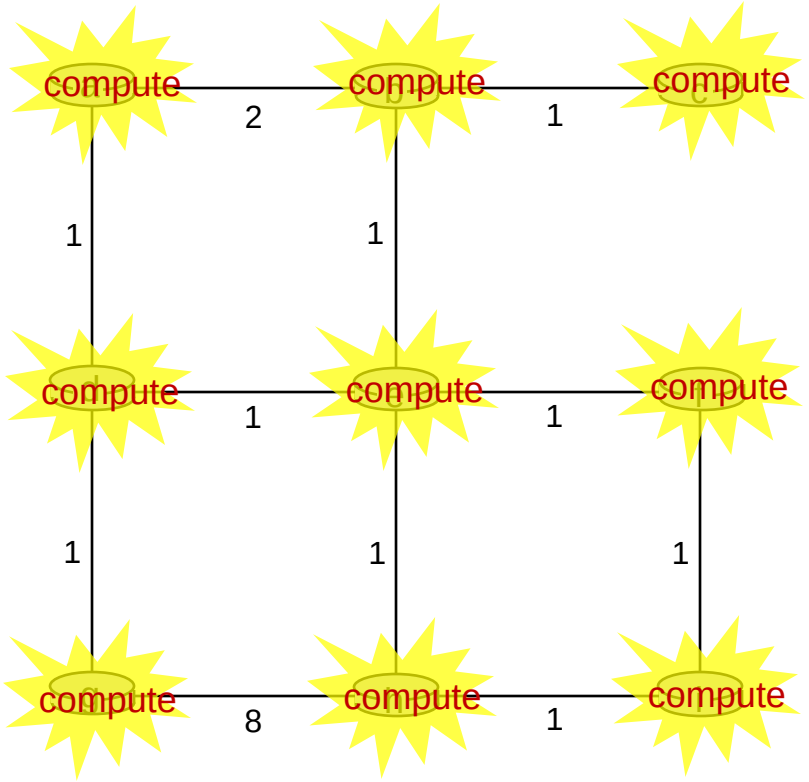


Distance vector example: iteration

t=2

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors

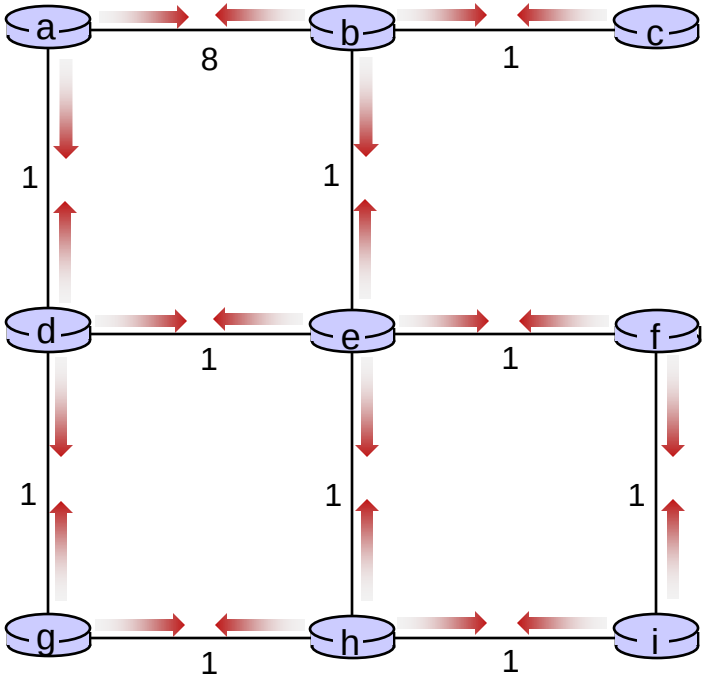


Distance vector example: iteration

t=2

All nodes:

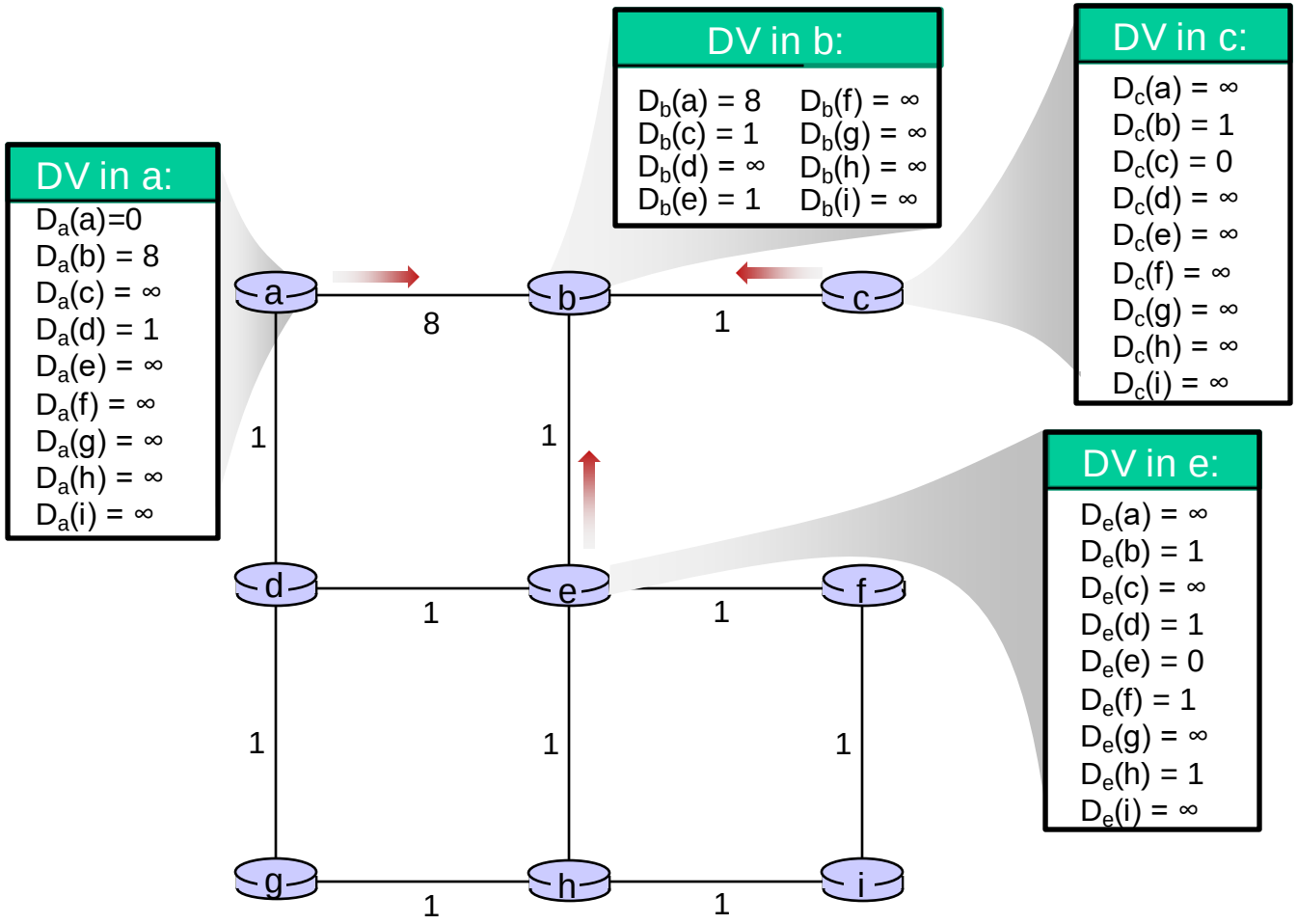
- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



Distance vector example: computation

t=1

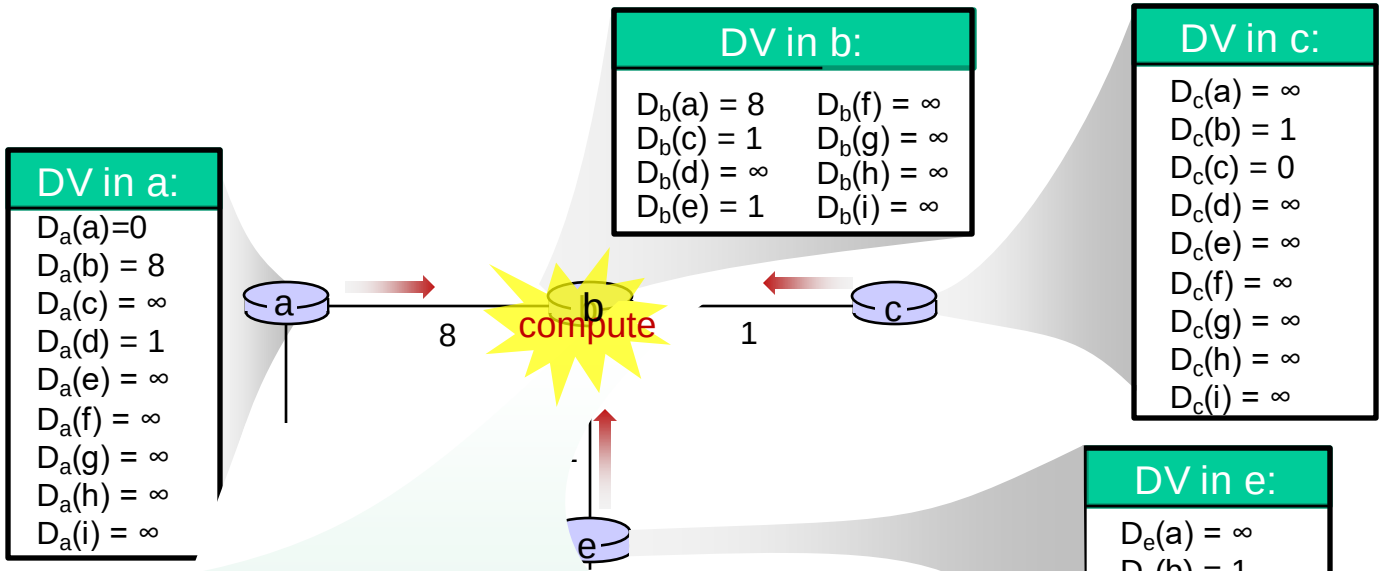
- b receives DVs from a, c, e



Distance vector example: computation

t=1

- b receives DVs from a, c, e, computes:



$D_b(a) = \min\{c_{b,a}+D_a(a), c_{b,c}+D_c(a), c_{b,e}+D_e(a)\} = \min\{8,\infty,\infty\} = 8$

$D_b(c) = \min\{c_{b,a}+D_a(c), c_{b,c}+D_c(c), c_{b,e}+D_e(c)\} = \min\{\infty,1,\infty\} = 1$

$D_b(d) = \min\{c_{b,a}+D_a(d), c_{b,c}+D_c(d), c_{b,e}+D_e(d)\} = \min\{9,2,\infty\} = 2$

$D_b(e) = \min\{c_{b,a}+D_a(e), c_{b,c}+D_c(e), c_{b,e}+D_e(e)\} = \min\{\infty,\infty,1\} = 1$

$D_b(f) = \min\{c_{b,a}+D_a(f), c_{b,c}+D_c(f), c_{b,e}+D_e(f)\} = \min\{\infty,\infty,2\} = 2$

$D_b(g) = \min\{c_{b,a}+D_a(g), c_{b,c}+D_c(g), c_{b,e}+D_e(g)\} = \min\{\infty, \infty, \infty\} = \infty$

$D_b(h) = \min\{c_{b,a}+D_a(h), c_{b,c}+D_c(h), c_{b,e}+D_e(h)\} = \min\{\infty, \infty, 2\} = 2$

$D_b(i) = \min\{c_{b,a}+D_a(i), c_{b,c}+D_c(i), c_{b,e}+D_e(i)\} = \min\{\infty, \infty, \infty\} = \infty$

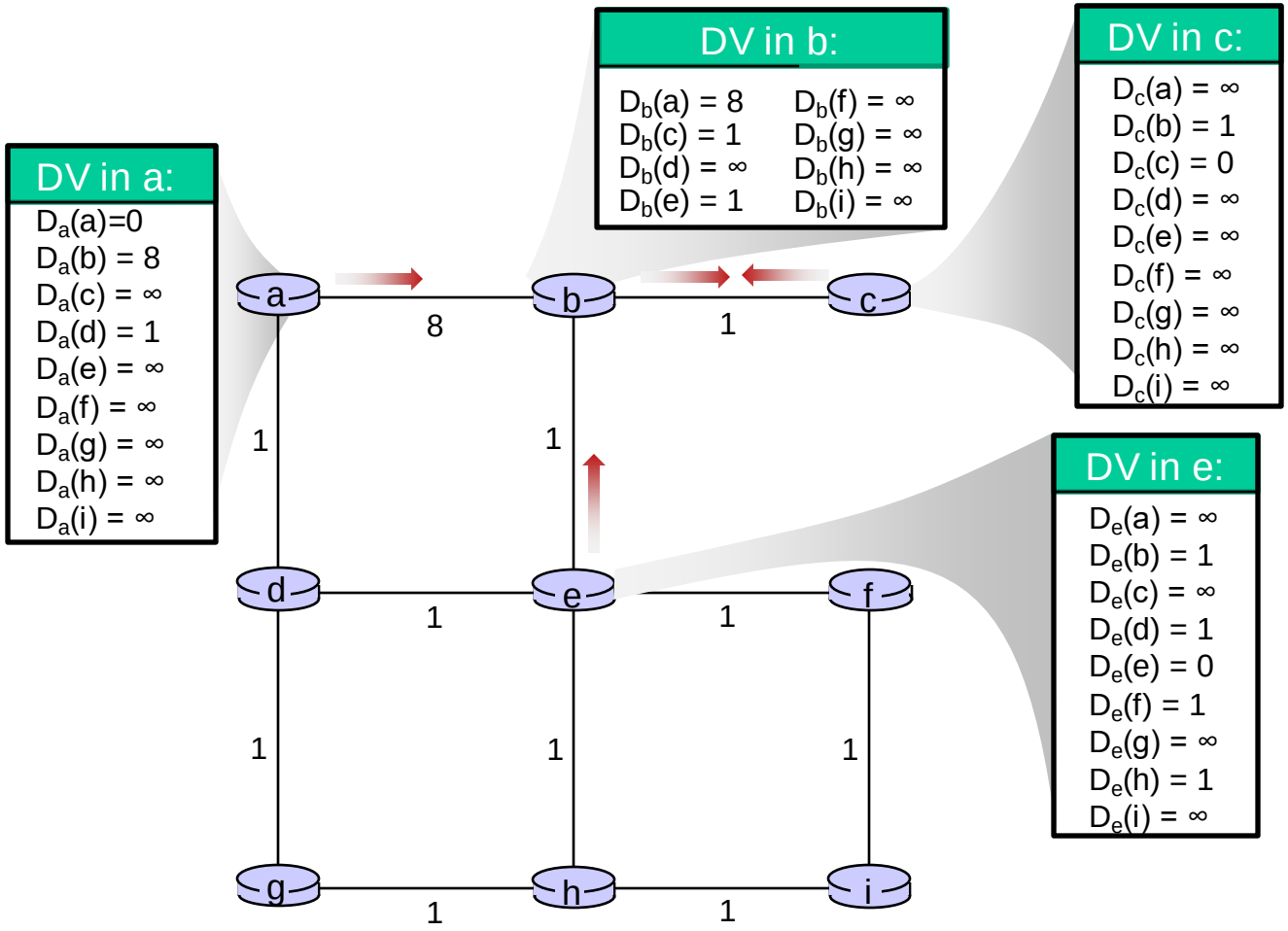
DV in b:

$D_b(a) = 8$	$D_b(f) = 2$
$D_b(c) = 1$	$D_b(g) = \infty$
$D_b(d) = 2$	$D_b(h) = 2$
$D_b(e) = 1$	$D_b(i) = \infty$

Distance vector example: computation

t=1

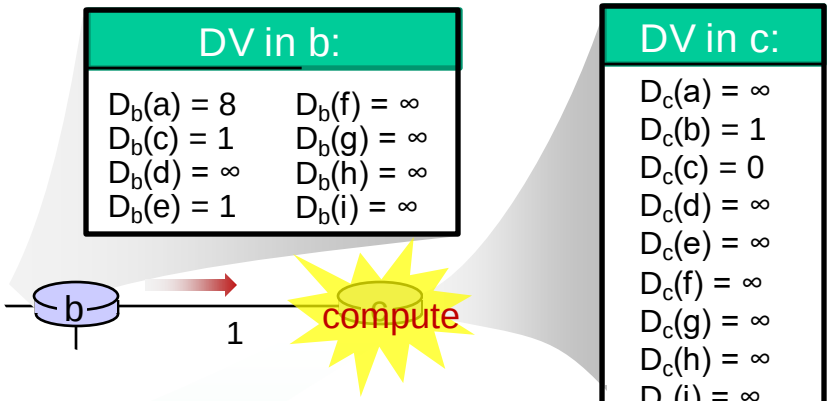
- c receives DVs from b



Distance vector example: computation

t=1

- c receives DVs from b computes:



$D_c(a) = \min\{c_{c,b}+D_b(a)\} = 1 + 8 = 9$

$D_c(b) = \min\{c_{c,b}+D_b(b)\} = 1 + 0 = 1$

$D_c(d) = \min\{c_{c,b}+D_b(d)\} = 1+ \infty = \infty$

$D_c(e) = \min\{c_{c,b}+D_b(e)\} = 1 + 1 = 2$

$D_c(f) = \min\{c_{c,b}+D_b(f)\} = 1+ \infty = \infty$

$D_c(g) = \min\{c_{c,b}+D_b(g)\} = 1+ \infty = \infty$

$D_c(h) = \min\{c_{bc,b}+D_b(h)\} = 1+ \infty = \infty$

$D_c(i) = \min\{c_{c,b}+D_b(i)\} = 1+ \infty = \infty$

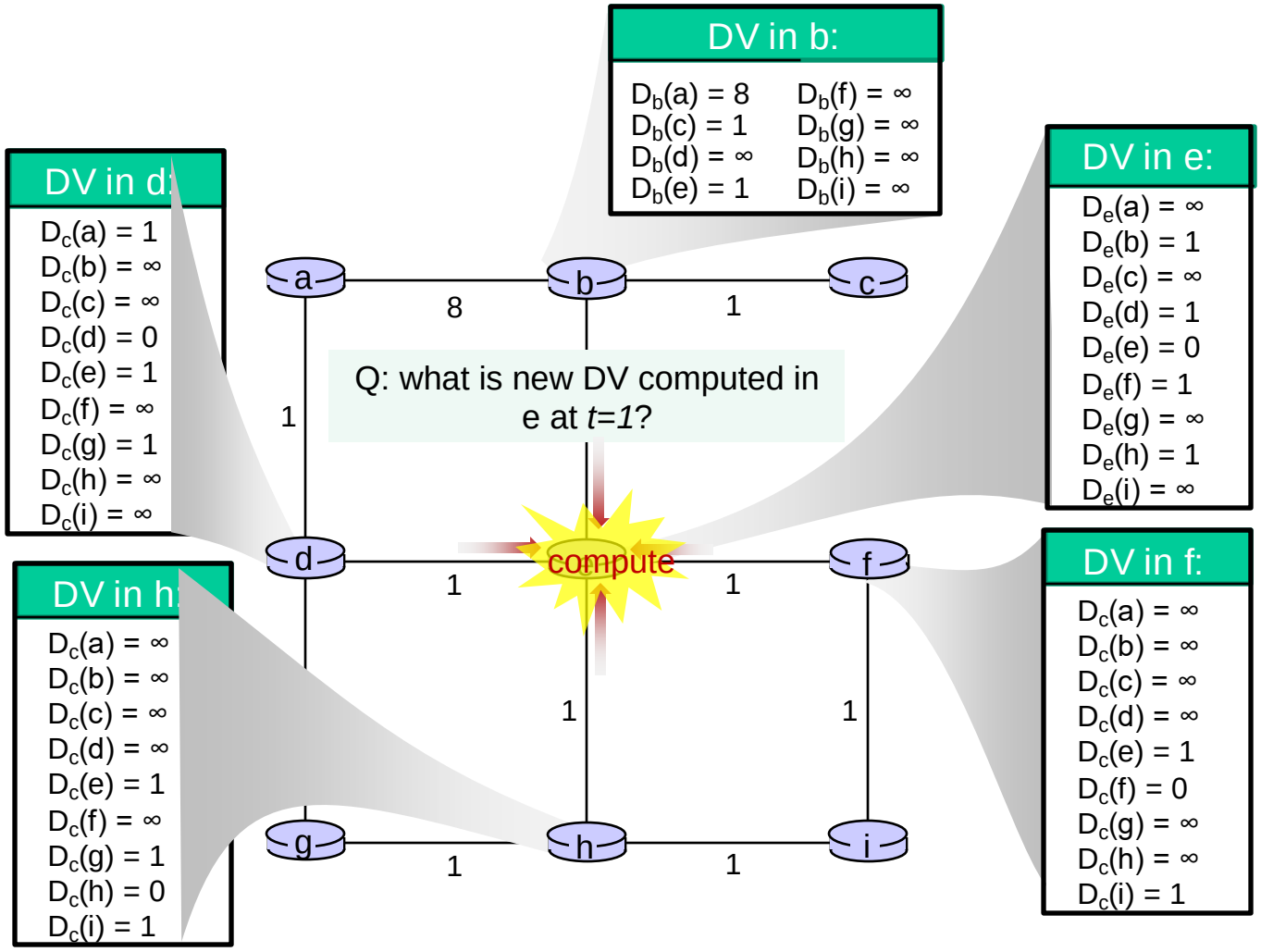
DV in c:

$D_c(a) = 9$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = 2$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

Distance vector example: computation

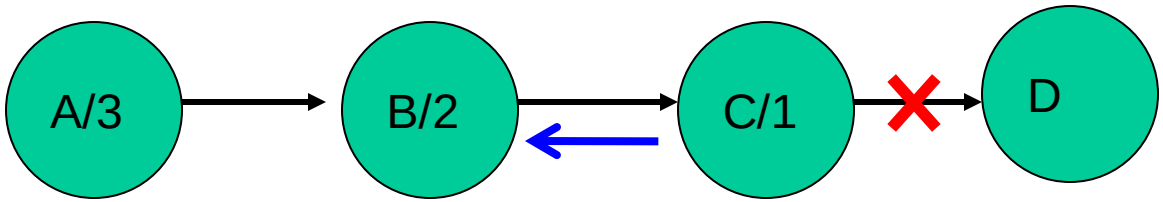
t=1

- e receives DVs from b, d, f, h



Count-To-Infinity Problem

- Assume we use hop count as metric
  - A uses B to reach D with cost 3
  - B uses C to reach D with cost 2
  - C reaches D with cost 1



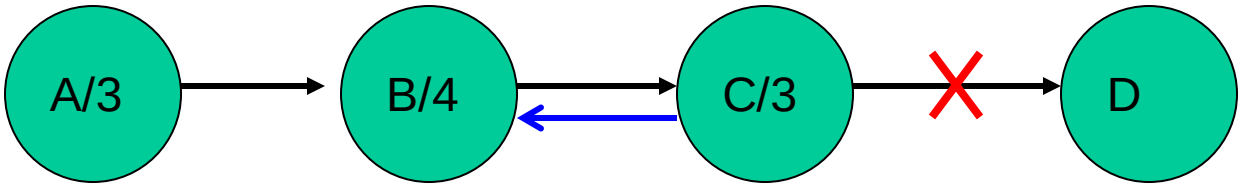
C sends to B the
packets destined to D

- Suppose link between C and D breaks
  - Since B informs its neighbors its distance to D is 2, C switches to B, sets its cost to (2+1= 3)

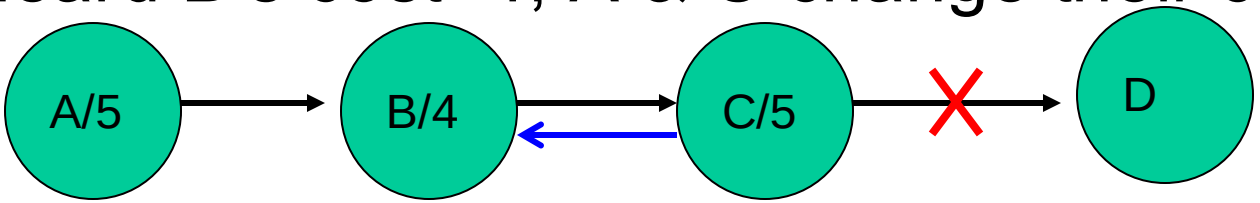
B's cost to reach D

Count-To-Infinity Problem (cont.)

- B's path cost is now 4
  - A has not realized what has happened yet



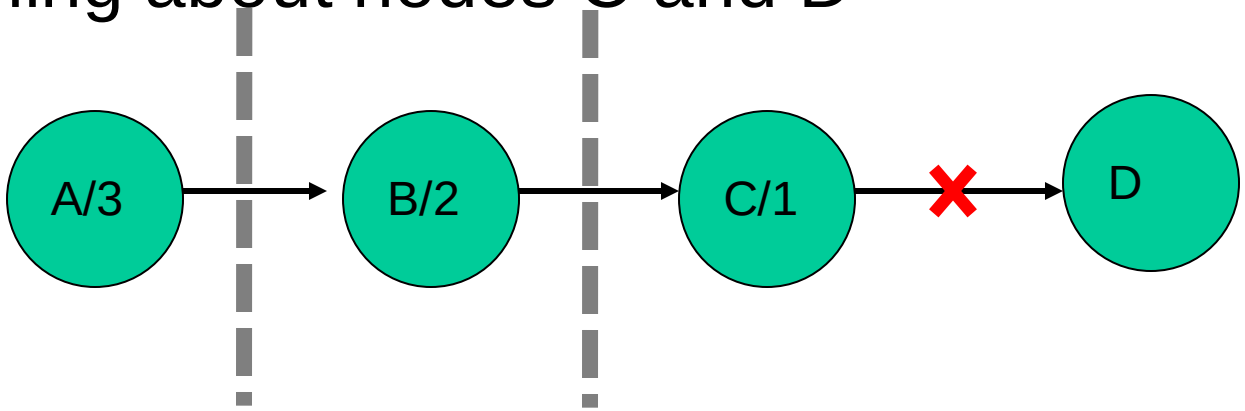
- Once heard B's cost=4, A & C change their cost to 5



- B hears C's cost=5, B changes its cost to D to 6
  - Cycle repeats, the distance "counting to infinity"
  - Meanwhile data packets from A to D loop between B and C

Split Horizon

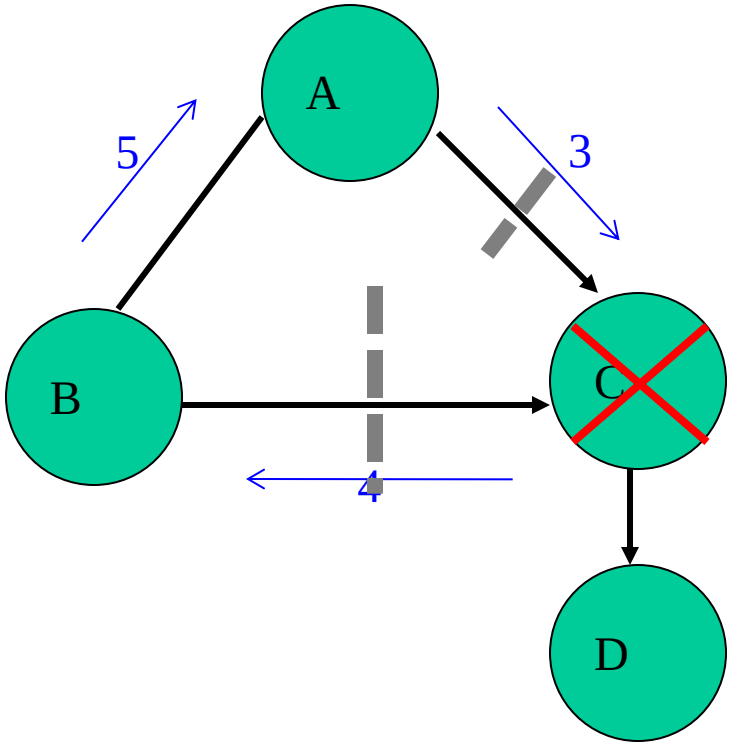
- Because B reaches D via C, B tells C nothing about node D
  - A tells B nothing about nodes C and D



A router should **not** advertise a route back to the same interface from which it learned it

Split Horizon: not effective in many cases

- Suppose the link between C and D breaks



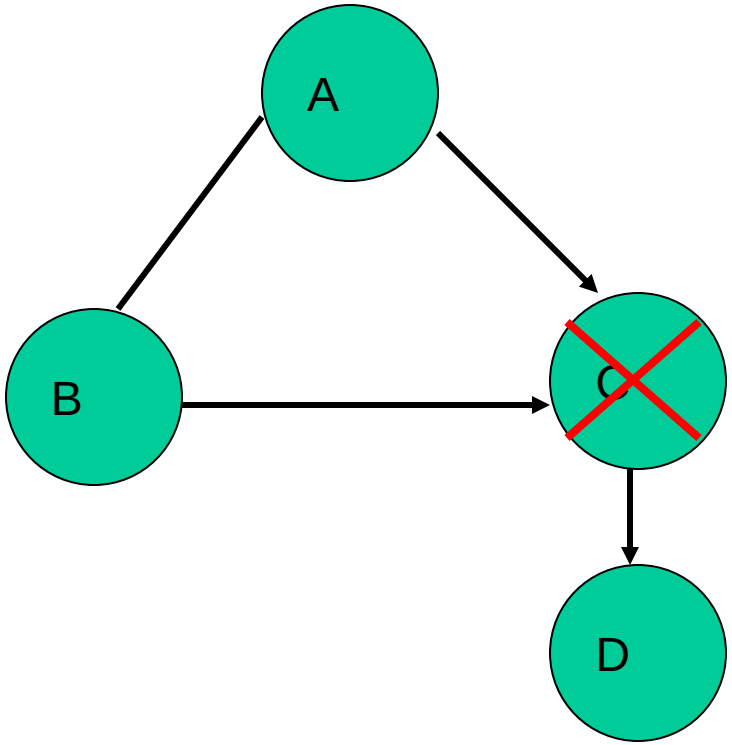
1. A and B do not tell C they can reach D
   - They do tell each other
     - A→B: my distance to D is 2
     - B→A: my distance to D is 2
2. When C fails
   - A sends to B the packets with destination=D
   - B sends to A the packets with destination=D

Packets can still loop

Split Horizon with poison reverse

important

- A & B go through C to reach D: both tell C that their distance to D is *infinite (poison reverse)*
  - C never attempts to reach D via A or B



When node C fails,

1. A tries to reach D via B: A tells B $D_D = \infty$
2. B does the same thing
3. Both A and B realize that D is no longer reachable, prevent packet looping.

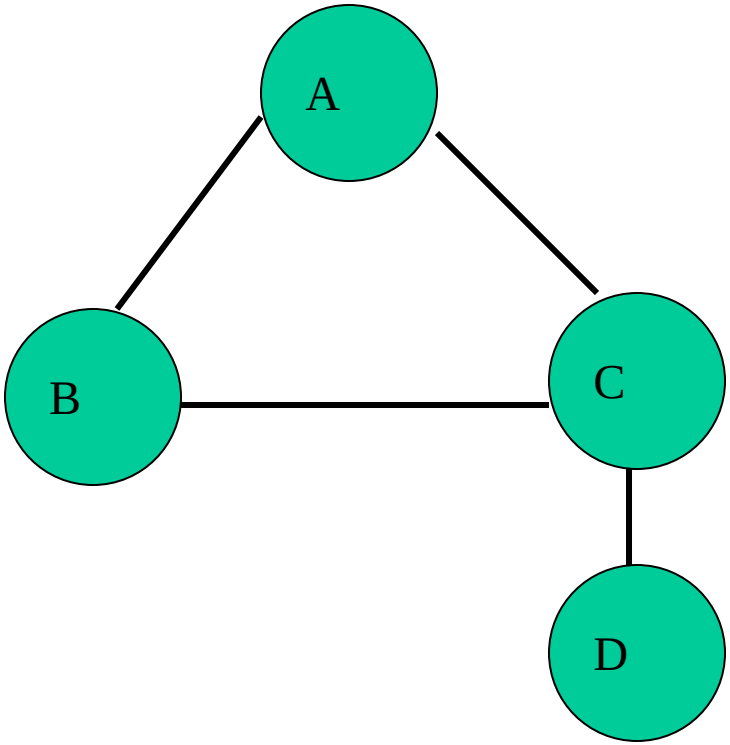
Routing Information Protocol (RIP): a distance vector protocol. From specification https://datatracker.ietf.org/doc/html/rfc2453

"... Split horizon with poisoned reverse will prevent any routing loops that involve only two routers. However, it is still possible to end up with patterns in which three routers are engaged in mutual deception..."

Another way to mitigate routing loops

Without using any split-horizon or poison-reverse stuff:

path-vector routing



- A's announcement: my path to D: A→C→D
- B's announcement: my path to D: B→C→D
- When C fails: both A and B realize there is no path to D

path-vector is used by BGP (border gateway protocol)

Comparison of LS and DV algorithms

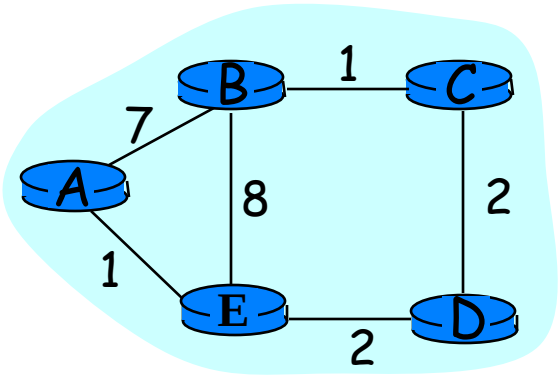
important

- **Performance measure**: i) message overhead, ii) time to convergence
- Distance vector:
  - Each node sends to neighbors its distances to all destinations
    - Each update msg can be large in size (linear with the #destinations), but travels over one link only
  - Each node only knows *distances* to other destinations
- Link state:
  - One's distance to all neighbors is broadcasted to the entire network
    - Each update msg is small in size, but travels through all the links in the network
  - Each node learns the entire topology map

What happens if a router malfunctions?

important

- Link-state
  - A node can advertise incorrect link cost
  - each node computes its own table
- Distance vector
  - A node can advertise incorrect path cost
  - one node's distance-list is used by its neighbors for their own routing selection



Node-D: "I have 0 cost to all other nodes"

Link-State:

- updates from A & B: not connected to D
- Updates from C & E: cost not 0

Distance-Vector:

- other nodes do not have info to verify