

# COMP 3331/9331: Computer Networks and Applications

Week 8

Network Layer: Control Plane (Routing)

**Chapter 5: Section 5.1 – 5.2, 5.6**

# Network layer, control plane: outline

## 5.1 introduction

## 5.2 routing protocols

- ❖ link state
- ❖ distance vector
- ❖ hierarchical routing

## 5.6 ICMP: The Internet Control Message Protocol

Self study

# Network-layer functions

*Recall: two network-layer functions:*

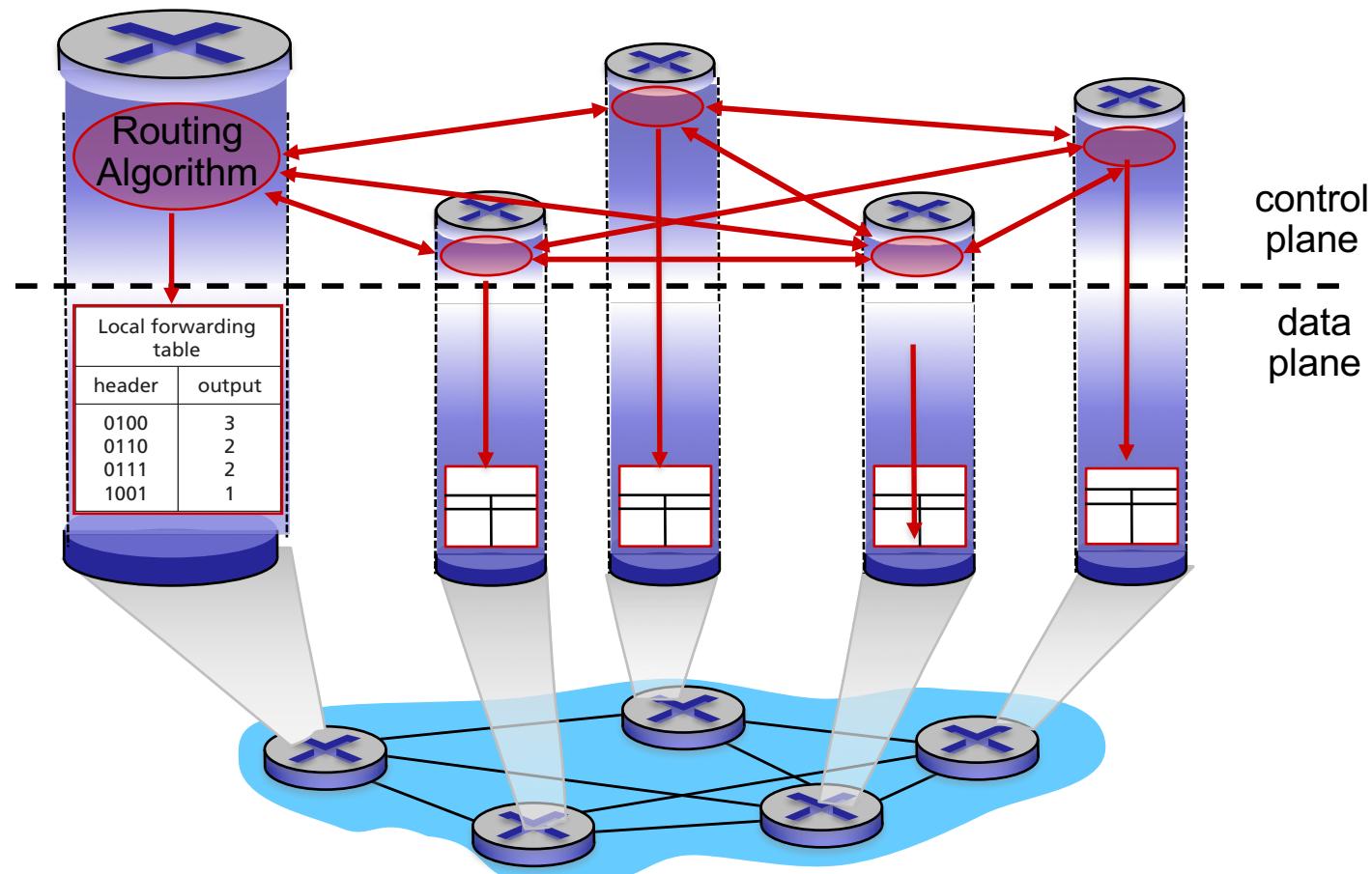
- ❖ *forwarding*: move packets from router's input to appropriate router output      ***data plane***
- *routing*: determine route taken by packets from source to destination      ***control plane***

*Two approaches to structuring network control plane:*

- per-router control (traditional)
- logically centralized control (software defined networking)

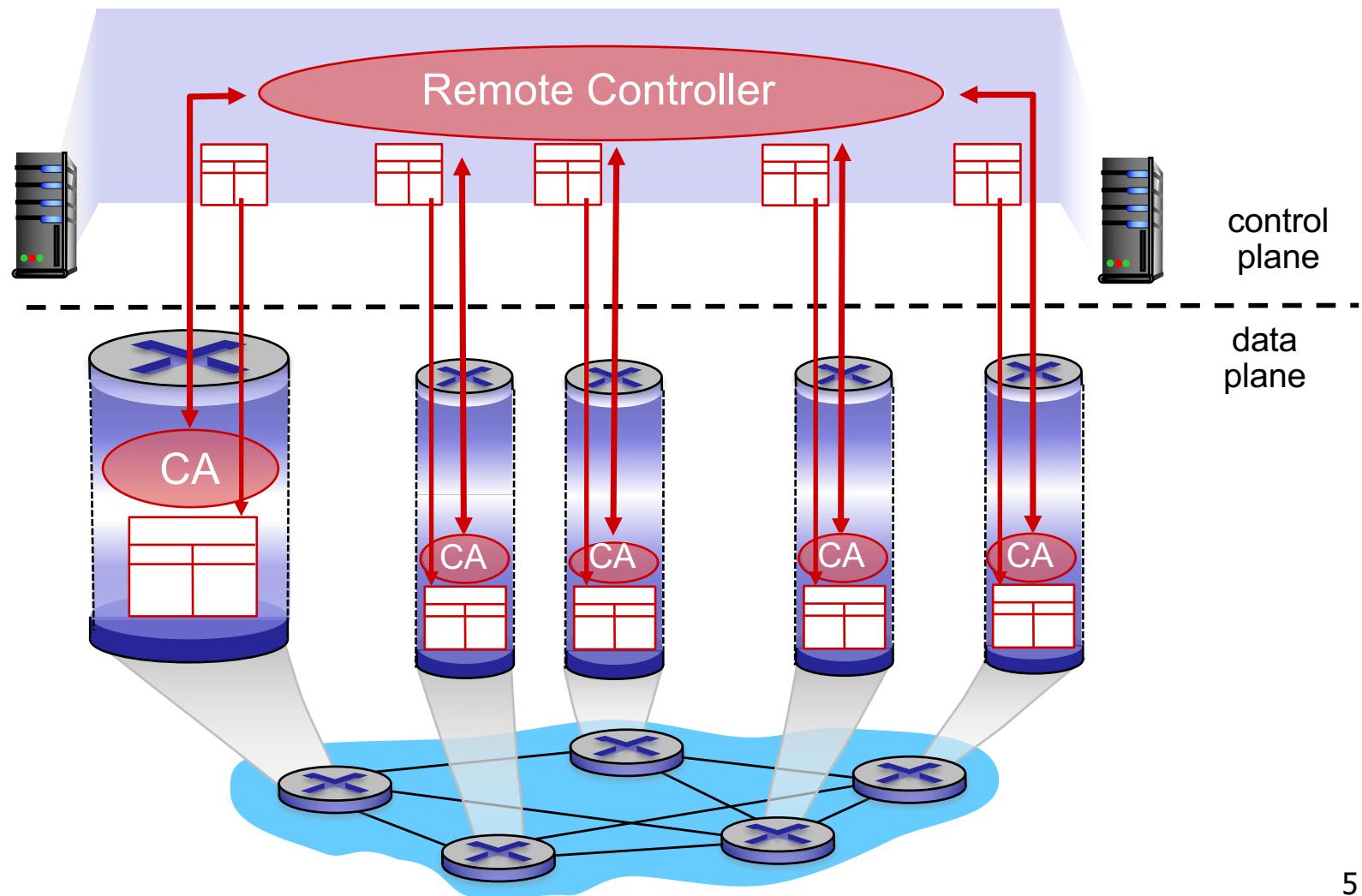
# Per-router control plane

Individual routing algorithm components *in each and every router* interact with each other in control plane to compute forwarding tables



# Logically centralized control plane

A distinct (typically remote) controller interacts with local control agents (CAs) in routers to compute forwarding tables



# Network layer, control plane: outline

5.1 introduction

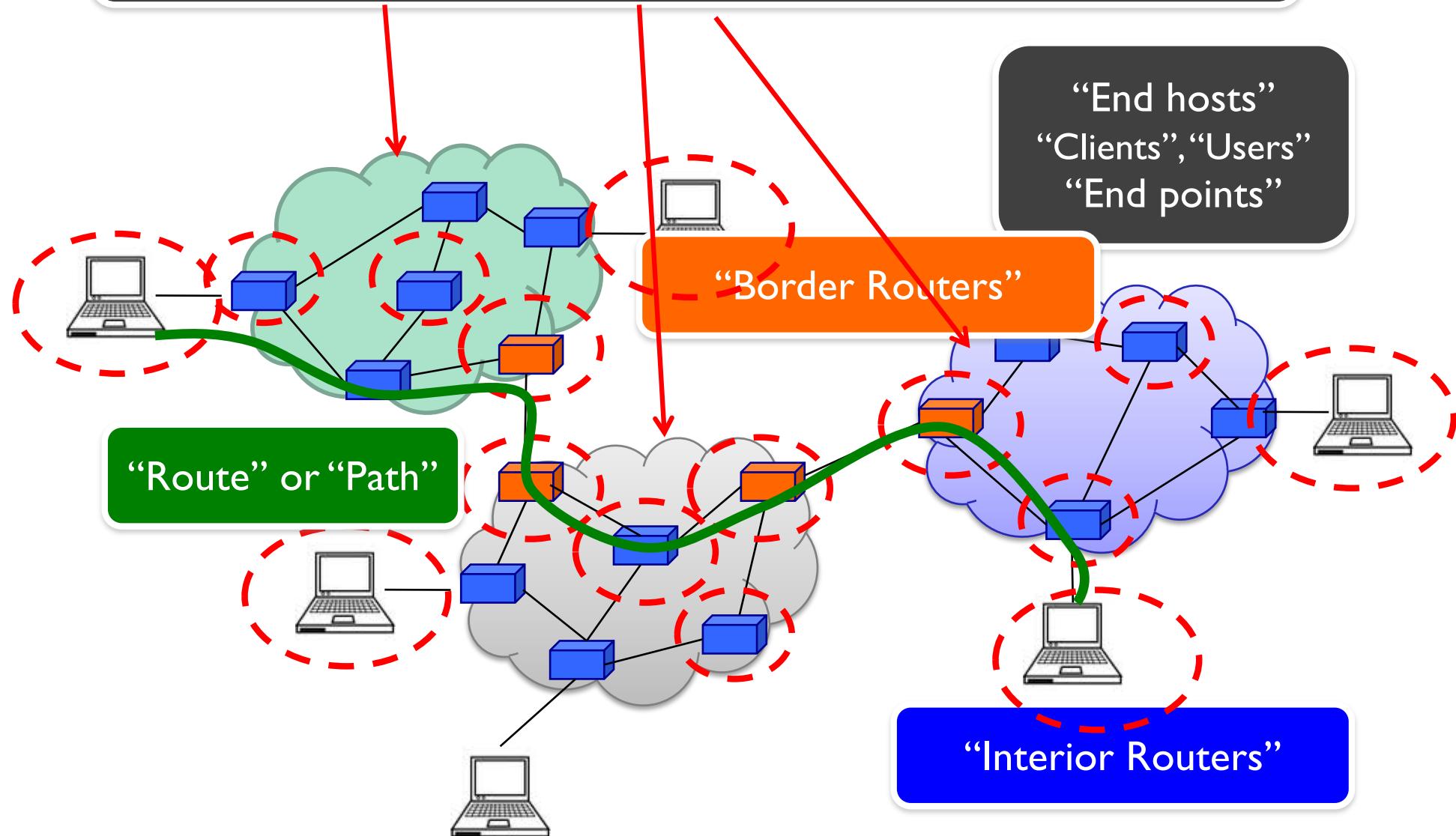
5.2 routing protocols

- ❖ link state
- ❖ distance vector
- ❖ Hierarchical routing

5.6 ICMP: The Internet  
Control Message  
Protocol

## “Autonomous System (AS)” or “Domain”

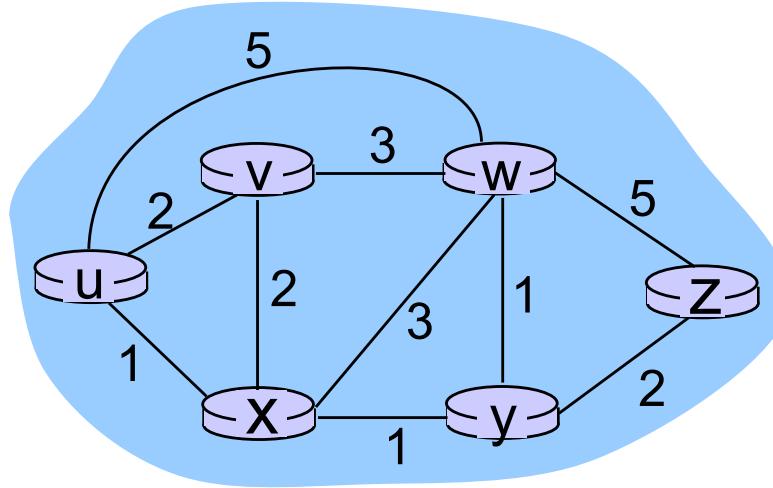
Region of a network under a single administrative entity



# Internet Routing

- ❖ Internet Routing works at two levels
- ❖ Each AS runs an **intra-domain** routing protocol that establishes routes within its domain
  - AS -- region of network under a single administrative entity
  - Link State, e.g., Open Shortest Path First (OSPF)
  - Distance Vector, e.g., Routing Information Protocol (RIP)
- ❖ ASes participate in an **inter-domain** routing protocol that establishes routes between domains
  - Path Vector, e.g., Border Gateway Protocol (BGP)

# Graph abstraction

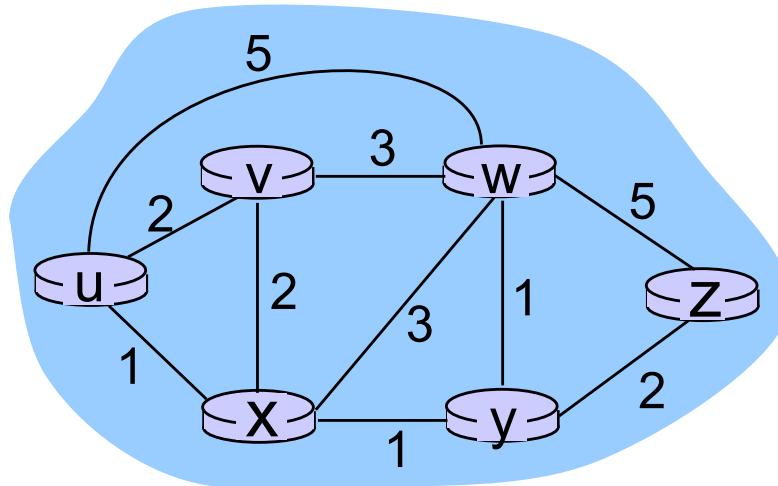


graph:  $G = (N, E)$

$N = \text{set of routers} = \{ u, v, w, x, y, z \}$

$E = \text{set of links} = \{ (u, v), (u, x), (u, w), (v, x), (v, w), (x, w), (x, y), (w, y), (w, z), (y, z) \}$

# Graph abstraction: costs



$c(x, x') = \text{cost of link } (x, x')$   
e.g.,  $c(w, z) = 5$

cost of path  $(x_1, x_2, x_3, \dots, x_p) = c(x_1, x_2) + c(x_2, x_3) + \dots + c(x_{p-1}, x_p)$

**key question:** what is the least-cost path between u and z ?  
**routing algorithm:** algorithm that finds that least cost path

## Link Cost

- ❖ Typically simple: all links are equal
- ❖ Least-cost paths => shortest paths (hop count)
- ❖ Network operators add policy exceptions
  - Lower operational costs
  - Peering agreements
  - Security concerns

# Network layer, control plane: outline

5.1 introduction

5.2 routing protocols

- ❖ link state

- ❖ distance vector

- ❖ hierarchical routing

5.6 ICMP: The Internet  
Control Message  
Protocol

# Routing algorithm classes

## *Link State (Global)*

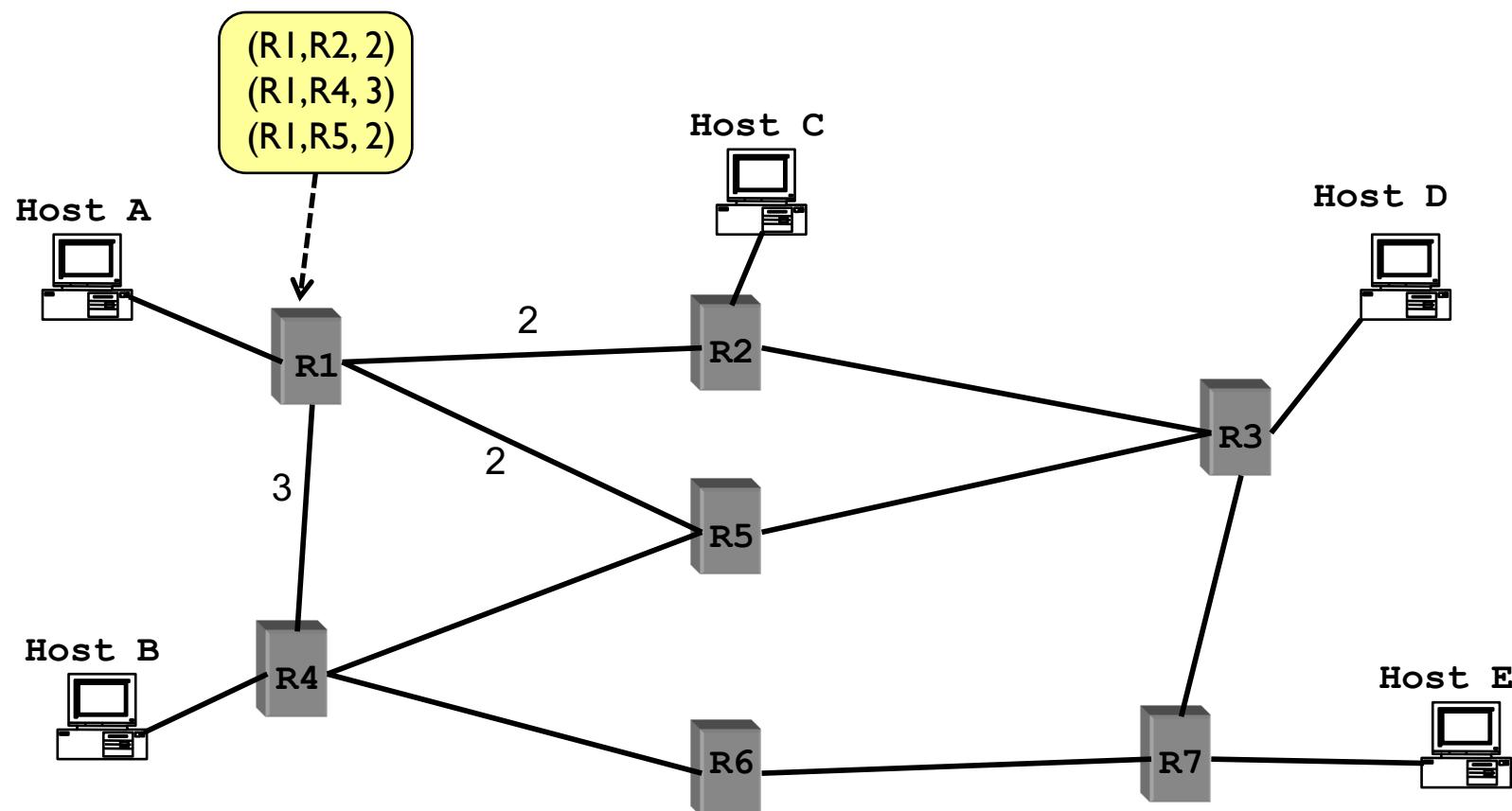
- Routers maintain cost of each link in the network
- Connectivity/cost changes flooded to all routers
- Converges quickly (less inconsistency, looping, etc.)
- Limited network sizes

## *Distance Vector (Decentralised)*

- Routers maintain next hop & cost of each destination.
- Connectivity/cost changes iteratively propagate from neighbour to neighbour
- Requires multiple rounds to converge
- Scales to large networks

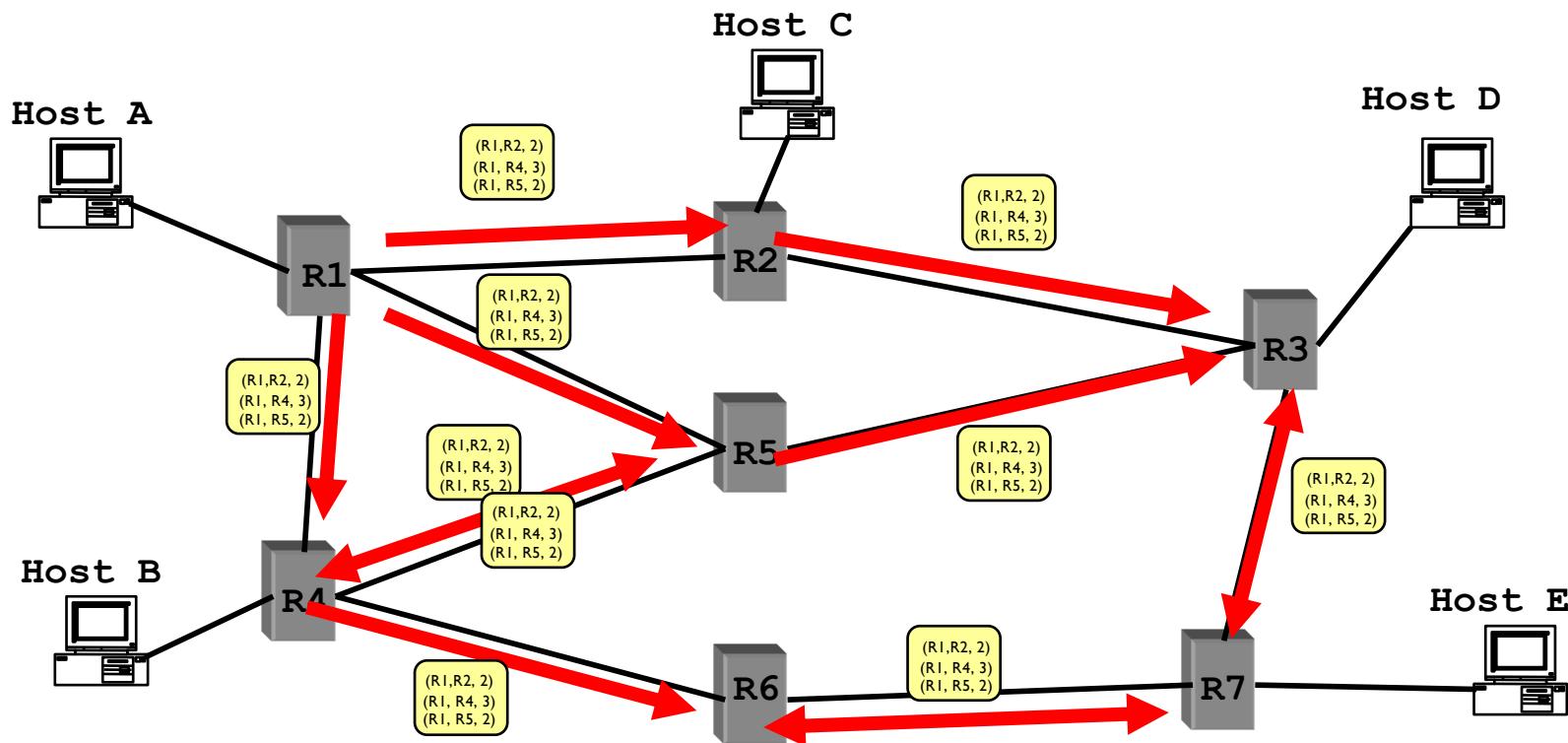
# Link State Routing

- ❖ Each node maintains its **local** “link state” (LS)
  - i.e., a list of its directly attached links and their costs



# Link State Routing

- ❖ Each node maintains its local “link state” (LS)
- ❖ Each node floods its local link state
  - on receiving a **new LS** message, a router forwards the message to all its neighbors other than the one it received the message from

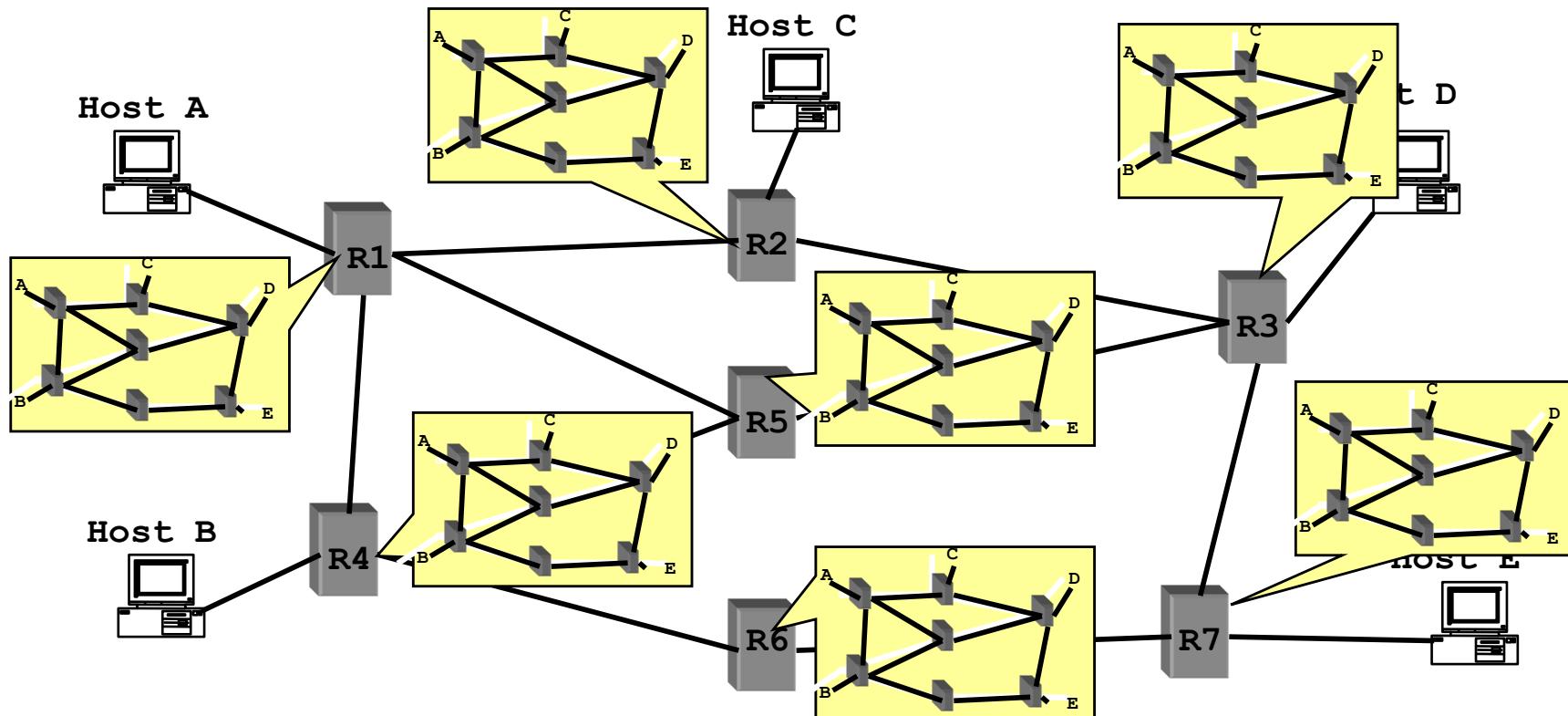


# Flooding LSAs

- ❖ Routers transmit **Link State Advertisement (LSA)** on links
  - A neighbouring router forwards out on all links except incoming
  - Keep a copy locally; don't forward previously-seen LSAs
- ❖ Challenges
  - Packet loss
  - Out of order arrival
- ❖ Solutions
  - Acknowledgements and retransmissions
  - Sequence numbers
  - Time-to-live for each packet

# Link State Routing

- ❖ Each node maintains its local “link state” (LS)
- ❖ Each node floods its local link state
- ❖ Eventually, each node learns the entire network topology
  - Can use Dijkstra’s to compute the shortest paths between nodes



# A Link-State Routing Algorithm

## *Dijkstra's algorithm*

- ❖ net topology, link costs known to all nodes
  - accomplished via “link state broadcast”
  - all nodes have same info
- ❖ computes least cost paths from one node (‘source’) to all other nodes
  - gives *forwarding table* for that node
- ❖ iterative: after k iterations, know least cost path to k dest.’s

## *notation:*

- ❖  $c(x,y)$ : link cost from node  $x$  to  $y$ ;  $= \infty$  if not direct neighbors
- ❖  $D(v)$ : current value of cost of path from source to dest.  $v$
- ❖  $p(v)$ : predecessor node along path from source to  $v$
- ❖  $N'$ : set of nodes whose least cost path definitively known

# Dijkstra's Algorithm

1 ***Initialization:***

2    $N' = \{u\}$

3   for all nodes  $v$

4     if  $v$  adjacent to  $u$

5       then  $D(v) = c(u,v)$

6     else  $D(v) = \infty$

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 all  $v$  adjacent to  $w$  and not in  $N'$  :

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

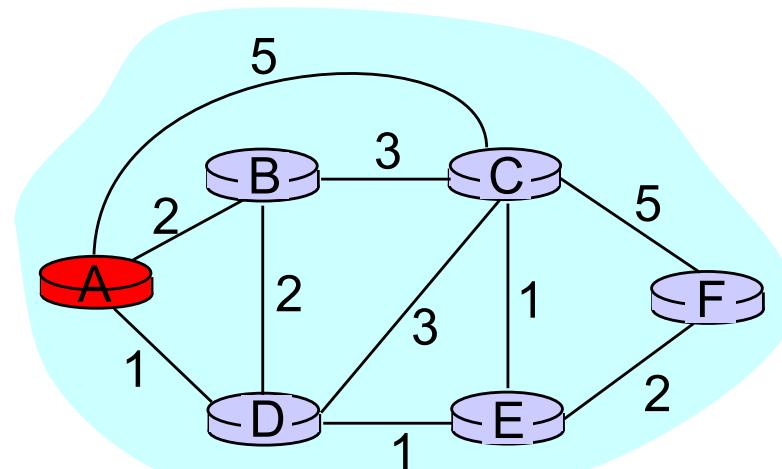
13   /\* new cost to  $v$  is either old cost to  $v$  or known

14   shortest path cost to  $w$  plus cost from  $w$  to  $v$  \*/

15 ***until all nodes in  $N'$***

# Example: Dijkstra's Algorithm

| Step | Set 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       | $\infty$  | $\infty$  |
| 1    |        |           |           |           |           |           |
| 2    |        |           |           |           |           |           |
| 3    |        |           |           |           |           |           |
| 4    |        |           |           |           |           |           |
| 5    |        |           |           |           |           |           |

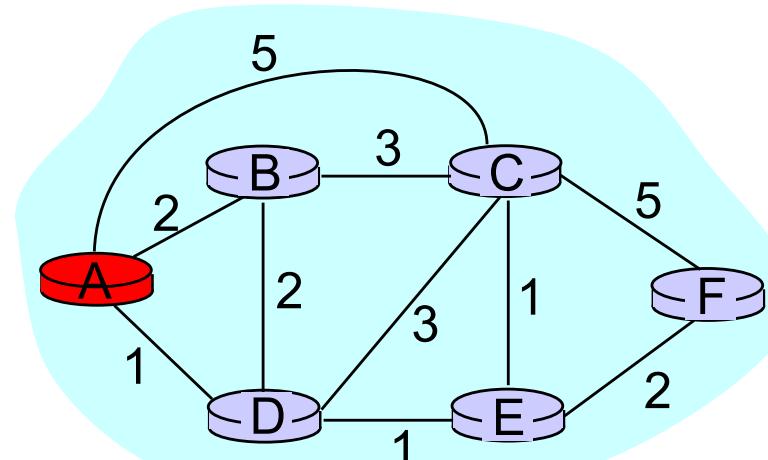


```
1 Initialization:
2    $N' = \{A\};$ 
3   for all nodes  $v$ 
4     if  $v$  adjacent to  $A$ 
5       then  $D(v) = c(A,v);$ 
6     else  $D(v) = \infty;$ 
...

```

# Example: Dijkstra's Algorithm

| Step | Set $N'$ | $D(B), p(B)$ | $D(C), p(C)$ | $D(D), p(D)$ | $D(E), p(E)$ | $D(F), p(F)$ |
|------|----------|--------------|--------------|--------------|--------------|--------------|
| 0    | A        | 2,A          | 5,A          | 1,A          | $\infty$     | $\infty$     |
| 1    |          |              |              |              |              |              |
| 2    |          |              |              |              |              |              |
| 3    |          |              |              |              |              |              |
| 4    |          |              |              |              |              |              |
| 5    |          |              |              |              |              |              |



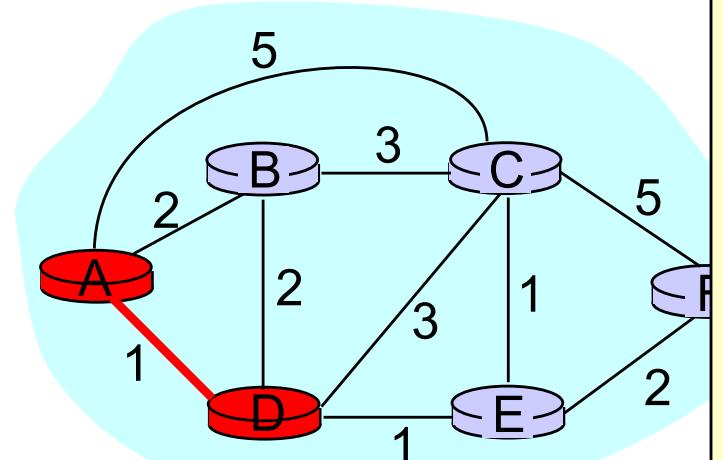
```

...
8 Loop
9   find w not in  $N'$  s.t.  $D(w)$  is a minimum;
10  add w to  $N'$ ;
11  update  $D(v)$  for all v adjacent
      to w and not in  $N'$ :
12  If  $D(w) + c(w,v) < D(v)$  then
13     $D(v) = D(w) + c(w,v)$ ;  $p(v) = w$ ;
14  until all nodes in  $N'$ ;

```

# Example: Dijkstra's Algorithm

| Step | Set $N'$ | $D(B), p(B)$ | $D(C), p(C)$ | $D(D), p(D)$ | $D(E), p(E)$ | $D(F), p(F)$ |
|------|----------|--------------|--------------|--------------|--------------|--------------|
| 0    | A        | 2,A          | 5,A          | 1,A          | $\infty$     | $\infty$     |
| 1    | AD       |              |              |              |              |              |
| 2    |          |              |              |              |              |              |
| 3    |          |              |              |              |              |              |
| 4    |          |              |              |              |              |              |
| 5    |          |              |              |              |              |              |



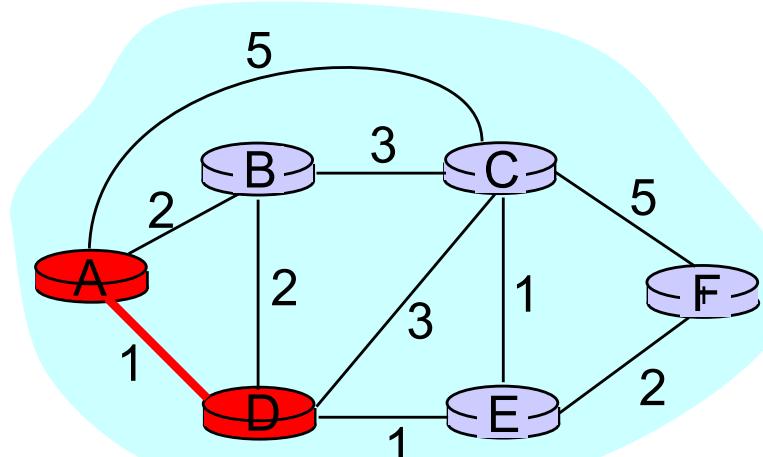
```

...
8  Loop
9    find w not in N' s.t. D(w) is a minimum;
10   add w to N';
11   update D(v) for all v adjacent
      to w and not in N':
12   If D(w) + c(w,v) < D(v) then
13     D(v) = D(w) + c(w,v); p(v) = w;
14   until all nodes in N';

```

# Example: Dijkstra's Algorithm

| Step | Set $N'$ | $D(B), p(B)$ | $D(C), p(C)$ | $D(D), p(D)$ | $D(E), p(E)$ | $D(F), p(F)$ |
|------|----------|--------------|--------------|--------------|--------------|--------------|
| 0    | A        | 2, A         | 5, A         | 1, A         | $\infty$     | $\infty$     |
| 1    | AD       | 2, A         | 4, D         |              | 2, D         |              |
| 2    |          |              |              |              |              |              |
| 3    |          |              |              |              |              |              |
| 4    |          |              |              |              |              |              |
| 5    |          |              |              |              |              |              |



```

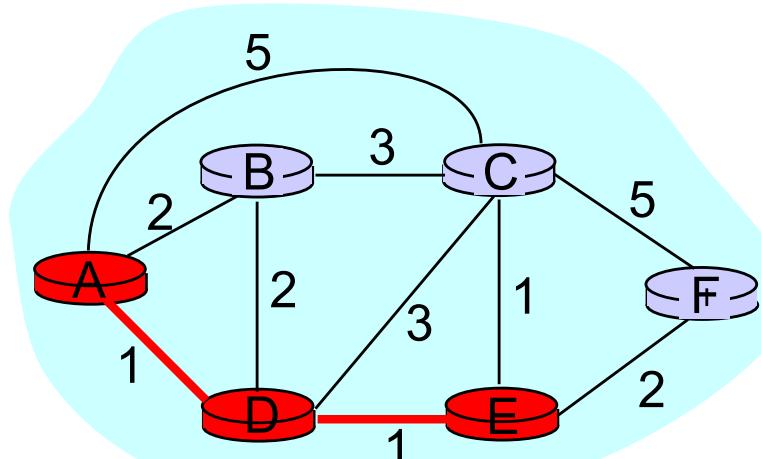
...
8  Loop
9  find w not in N' s.t. D(w) is a minimum;
10 add w to N';
11 update D(v) for all v adjacent
     to w and not in N';
12 If D(w) + c(w,v) < D(v) then
13   D(v) = D(w) + c(w,v); p(v) = w;
14 until all nodes in N';

```

# Example: Dijkstra's Algorithm

| Step | Set $N'$ | $D(B), p(B)$ | $D(C), p(C)$ | $D(D), p(D)$ | $D(E), p(E)$ | $D(F), p(F)$ |
|------|----------|--------------|--------------|--------------|--------------|--------------|
| 0    | A        | 2,A          | 5,A          | 1,A          | $\infty$     | $\infty$     |
| 1    | AD       | 2, A         | 4,D          |              | 2,D          |              |
| 2    | ADE      | 2, A         | 3,E          |              |              | 4,E          |
| 3    |          |              |              |              |              |              |
| 4    |          |              |              |              |              |              |
| 5    |          |              |              |              |              |              |

→ Step 2



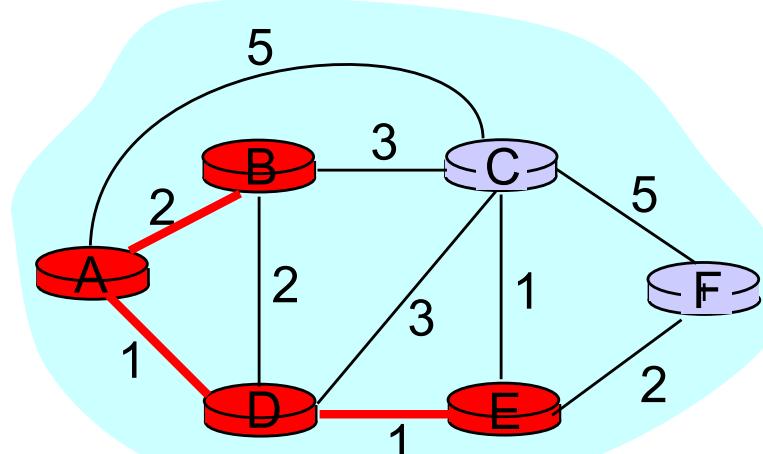
```

...
8 Loop
9   find w not in N' s.t. D(w) is a minimum;
10  add w to N';
11  update D(v) for all v adjacent
      to w and not in N':
12  If D(w) + c(w,v) < D(v) then
13    D(v) = D(w) + c(w,v); p(v) = w;
14 until all nodes in N';

```

# Example: Dijkstra's Algorithm

| Step | Set $N'$ | $D(B), p(B)$ | $D(C), p(C)$ | $D(D), p(D)$ | $D(E), p(E)$ | $D(F), p(F)$ |
|------|----------|--------------|--------------|--------------|--------------|--------------|
| 0    | A        | 2,A          | 5,A          | 1,A          | $\infty$     | $\infty$     |
| 1    | AD       | 2,A          | 4,D          |              | 2,D          |              |
| 2    | ADE      | 2,A          | 3,E          |              |              | 4,E          |
| 3    | ADEB     |              | 3,E          |              |              | 4,E          |
| 4    |          |              |              |              |              |              |
| 5    |          |              |              |              |              |              |



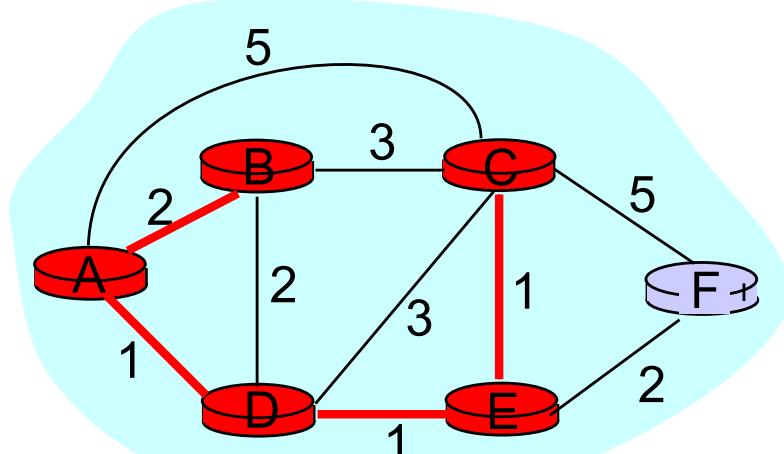
```

...
8 Loop
9   find w not in N' s.t. D(w) is a minimum;
10  add w to N';
11  update D(v) for all v adjacent
      to w and not in N':
12  If D(w) + c(w,v) < D(v) then
13    D(v) = D(w) + c(w,v); p(v) = w;
14 until all nodes in N';

```

# Example: Dijkstra's Algorithm

| Step | Set $N'$ | $D(B), p(B)$ | $D(C), p(C)$ | $D(D), p(D)$ | $D(E), p(E)$ | $D(F), p(F)$ |
|------|----------|--------------|--------------|--------------|--------------|--------------|
| 0    | A        | 2,A          | 5,A          | 1,A          | $\infty$     | $\infty$     |
| 1    | AD       | 2,A          | 4,D          |              | 2,D          |              |
| 2    | ADE      | 2,A          | 3,E          |              |              | 4,E          |
| 3    | ADEB     |              | 3,E          |              |              | 4,E          |
| →4   | ADEBC    |              |              |              |              | 4,E          |
| 5    |          |              |              |              |              |              |



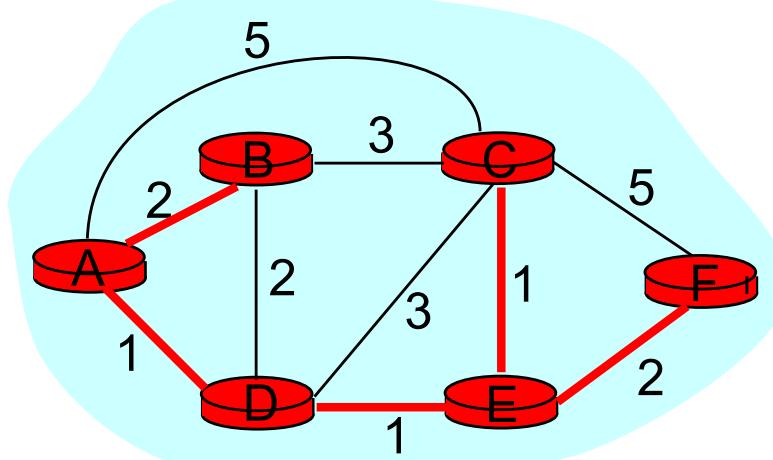
```

...
8 Loop
9   find w not in N' s.t. D(w) is a minimum;
10  add w to N';
11  update D(v) for all v adjacent
      to w and not in N':
12  If D(w) + c(w,v) < D(v) then
13    D(v) = D(w) + c(w,v); p(v) = w;
14 until all nodes in N';

```

# Example: Dijkstra's Algorithm

| Step | Set 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       | $\infty$  | $\infty$  |
| 1    | AD     | 2,A       | 4,D       |           | 2,D       |           |
| 2    | ADE    | 2,A       | 3,E       |           | 4,E       |           |
| 3    | ADEB   |           | 3,E       |           | 4,E       |           |
| 4    | ADEBC  |           |           |           | 4,E       |           |
| →5   | ADEBCF |           |           |           |           |           |



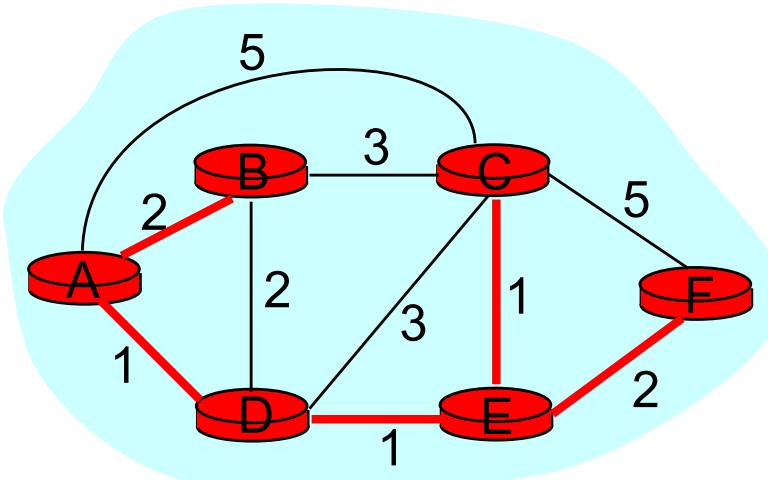
```

...
8 Loop
9   find w not in N' s.t. D(w) is a minimum;
10  add w to N';
11  update D(v) for all v adjacent
      to w and not in N':
12  If D(w) + c(w,v) < D(v) then
13    D(v) = D(w) + c(w,v); p(v) = w;
14 until all nodes in N';

```

# Example: Dijkstra's Algorithm

| Step | Set 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       | $\infty$  | $\infty$  |
| 1    | AD     |           | 4,D       |           | 2,D       |           |
| 2    | ADE    |           | 3,E       |           |           | 4,E       |
| 3    | ADEB   |           |           |           |           |           |
| 4    | ADEBC  |           |           |           |           |           |
| 5    | ADEBCF |           |           |           |           |           |

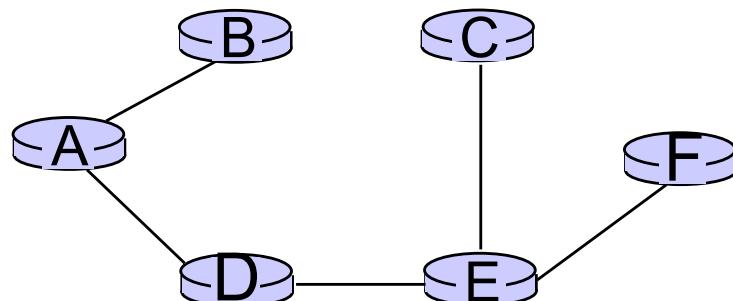


To determine path  $A \rightarrow C$  (say), work backward from C via  $p(v)$

# The Forwarding Table

- Running Dijkstra at node A gives the shortest path from A to all destinations
- We then construct the *forwarding table*

resulting shortest-path tree from A:



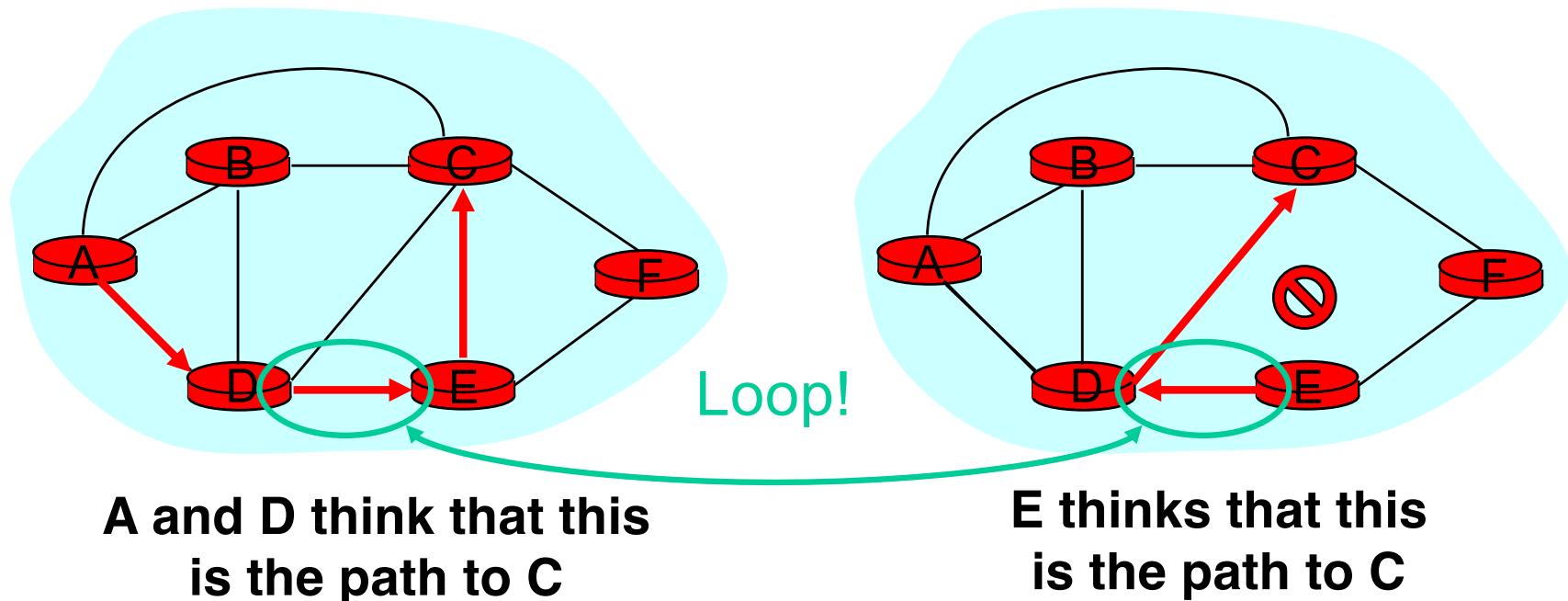
| Destination | Link  |
|-------------|-------|
| B           | (A,B) |
| C           | (A,D) |
| D           | (A,D) |
| E           | (A,D) |
| F           | (A,D) |

# Issue #1: Scalability

- ❖ How many messages needed to flood link state messages?
  - $O(N \times E)$ , where  $N$  is #nodes;  $E$  is #edges in graph
- ❖ Processing complexity for Dijkstra's algorithm?
  - $O(N^2)$ , because we check all nodes  $w$  not in  $N'$  at each iteration and we have  $O(N)$  iterations
- ❖ How many entries in the LS topology database?  $O(E)$
- ❖ How many entries in the forwarding table?  $O(N)$

# Issue#2: Transient Disruptions

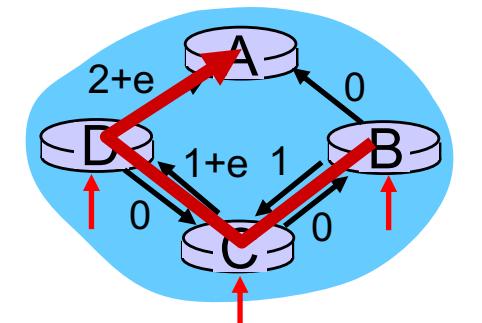
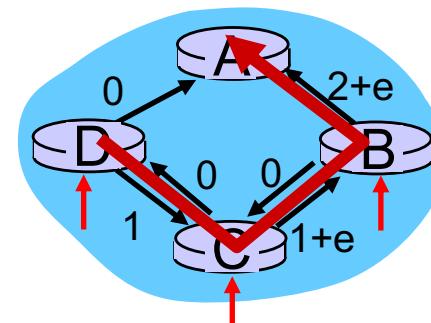
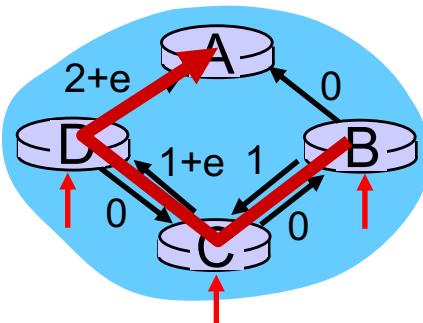
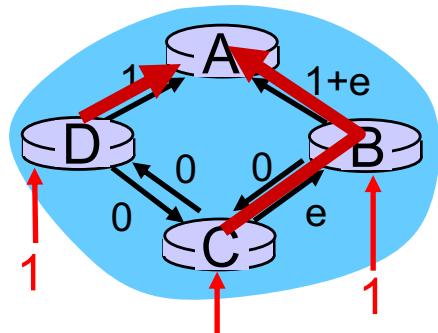
- ❖ Inconsistent link-state database
    - Some routers know about failure before others
    - The shortest paths are no longer consistent
    - Can cause **transient forwarding loops**



# Oscillations

*oscillations possible:*

- ❖ e.g., suppose link cost equals amount of carried traffic:



# Network layer, control plane: outline

5.1 introduction

5.2 routing protocols

- ❖ link state

- ❖ distance vector

- ❖ hierarchical routing

5.6 ICMP: The Internet  
Control Message  
Protocol

# Distance vector algorithm

*Bellman-Ford equation*

let

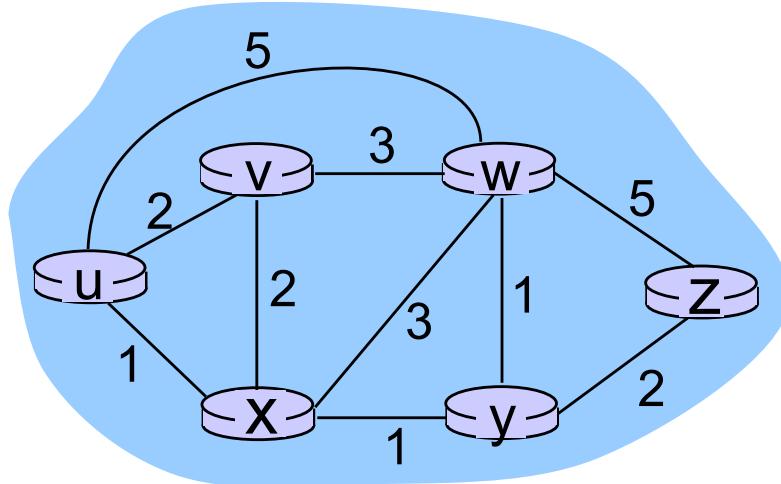
$d_x(y) := \text{cost of least-cost path from } x \text{ to } y$

then

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

|            |            |  
|            |            |  
|            |            |  
 $v$             cost from neighbor  $v$  to destination  $y$   
cost to neighbor  $v$   
 $\min$  taken over all neighbors  $v$  of  $x$

# Bellman-Ford example



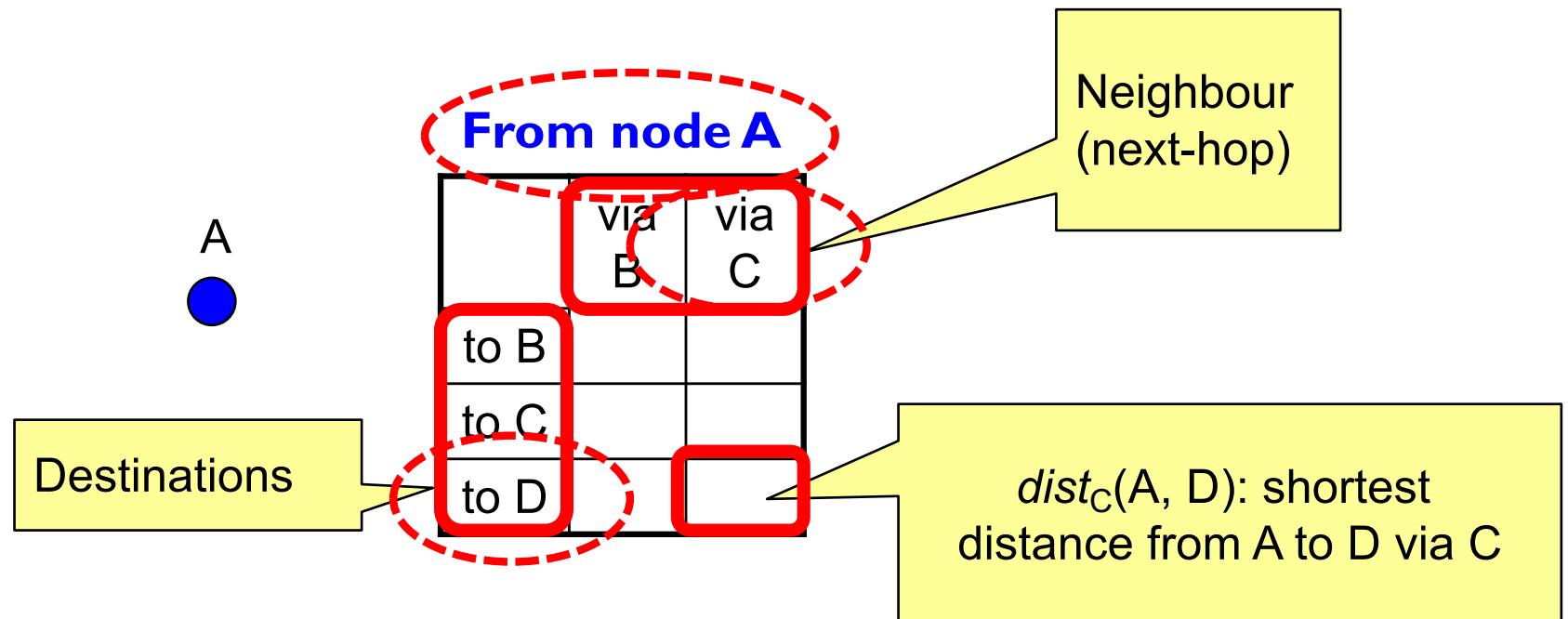
clearly,  $d_v(z) = 5$ ,  $d_x(z) = 3$ ,  $d_w(z) = 3$

B-F equation says:

$$\begin{aligned}d_u(z) &= \min \{ c(u,v) + d_v(z), \\&\quad c(u,x) + d_x(z), \\&\quad c(u,w) + d_w(z) \} \\&= \min \{ 2 + 5, \\&\quad 1 + 3, \\&\quad 5 + 3 \} = 4\end{aligned}$$

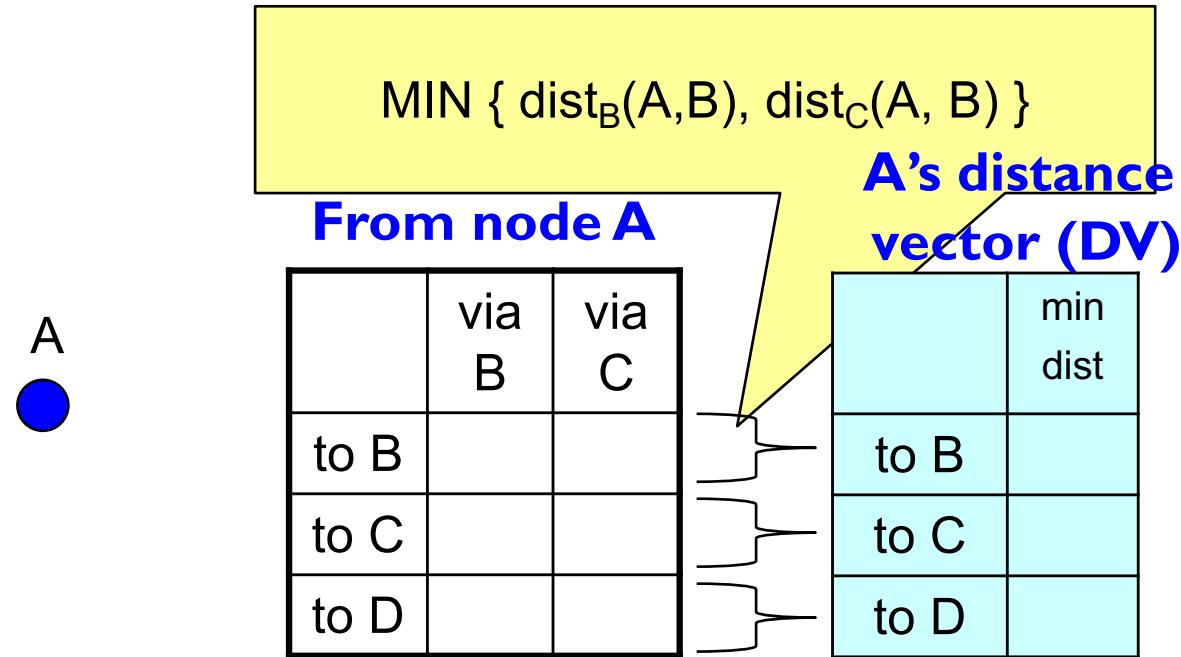
node achieving minimum is next  
hop in shortest path, used in forwarding table

# How Distance-Vector (DV) works



Each router maintains its shortest distance to every destination via each of its neighbours

# How Distance-Vector (DV) works



Each router computes its shortest distance to every destination via any of its neighbors

# How Distance-Vector (DV) works

A  
●

**From node A**

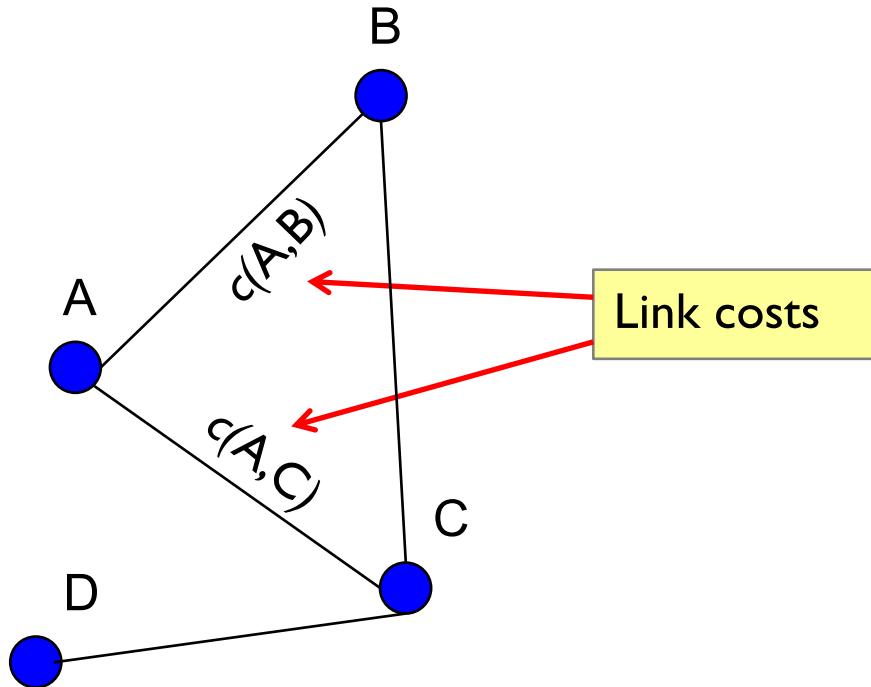
|      | via<br>B | via<br>C |
|------|----------|----------|
| to B | ?        | ?        |
| to C | ?        | ?        |
| to D | ?        | ?        |

**A's DV**

|      | min<br>dist |
|------|-------------|
| to B | ?           |
| to C | ?           |
| to D | ?           |

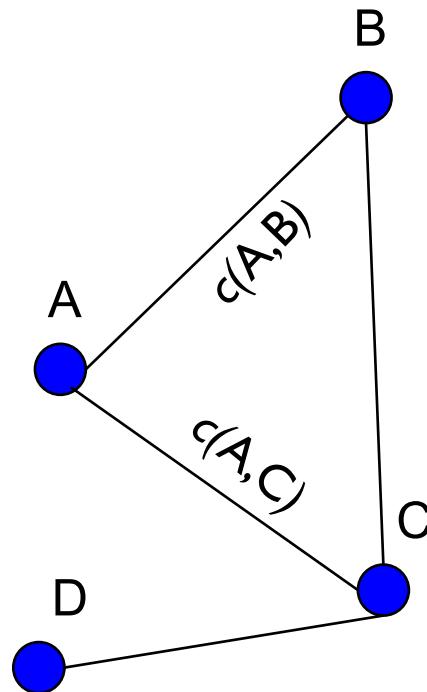
How does A initialize its dist() table and DV?

# How Distance-Vector (DV) works



How does A initialize its `dist()` table and DV?

# How Distance-Vector (DV) works



From node A

|      | via B    | via C    |
|------|----------|----------|
| to B | $c(A,B)$ | $\infty$ |
| to C | $\infty$ | $c(A,C)$ |

A's DV

|      | mindist  |
|------|----------|
| to B | $c(A,B)$ |
| to C | $c(A,C)$ |

Each router initializes its  $dist()$  table based on its immediate neighbors and link costs

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

$$= \min\{2+0, 7+1\} = 2$$

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

$$= \min\{2+1, 7+0\} = 3$$

**node x  
table**

|    | via      |          |
|----|----------|----------|
|    | y        | z        |
| to | 2        | $\infty$ |
| z  | $\infty$ | 7        |

**node y  
table**

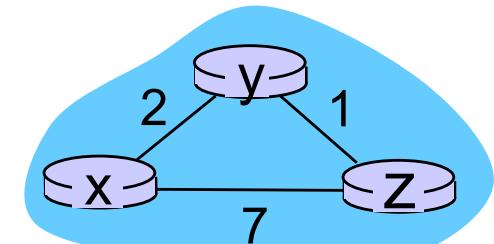
|    | via      |          |
|----|----------|----------|
|    | x        | z        |
| to | 2        | $\infty$ |
| z  | $\infty$ | 1        |

**node z  
table**

|    | via      |          |
|----|----------|----------|
|    | x        | y        |
| to | 7        | $\infty$ |
| y  | $\infty$ | 1        |

|    | via |   |
|----|-----|---|
|    | y   | z |
| to | 2   | 8 |
| z  | 3   | 7 |

→ time



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

$$= \min\{2+0, 7+1\} = 2$$

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

$$= \min\{2+1, 7+0\} = 3$$

**node x  
table**

|    | via      |          |
|----|----------|----------|
|    | y        | z        |
| to | 2        | $\infty$ |
| z  | $\infty$ | 7        |

**node y  
table**

|    | via      |          |
|----|----------|----------|
|    | x        | z        |
| to | 2        | $\infty$ |
| z  | $\infty$ | 1        |

**node z  
table**

|    | via      |          |
|----|----------|----------|
|    | x        | y        |
| to | 7        | $\infty$ |
| y  | $\infty$ | 1        |

|    | via |   |
|----|-----|---|
|    | y   | z |
| to | 2   | 8 |
| z  | 3   | 7 |

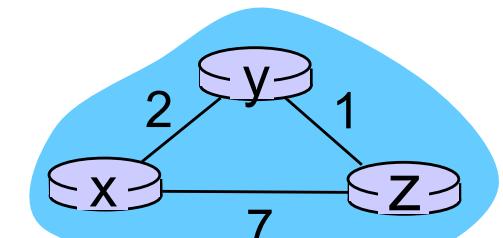
|    | via |   |
|----|-----|---|
|    | y   | z |
| to | 2   | 8 |
| z  | 3   | 7 |

|    | via |   |
|----|-----|---|
|    | x   | z |
| to | 2   | 8 |
| z  | 9   | 1 |

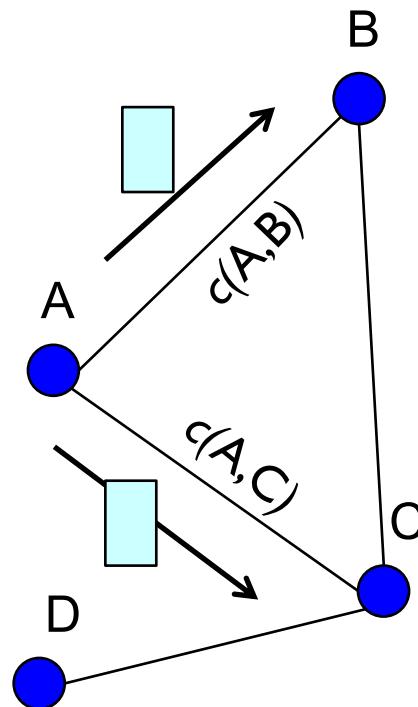
|    | via |   |
|----|-----|---|
|    | x   | z |
| to | 2   | 4 |
| z  | 5   | 1 |

|    | via |   |
|----|-----|---|
|    | x   | y |
| to | 7   | 3 |
| y  | 9   | 1 |

|    | via |   |
|----|-----|---|
|    | x   | y |
| to | 7   | 3 |
| y  | 9   | 1 |



# How Distance-Vector (DV) works



From node A

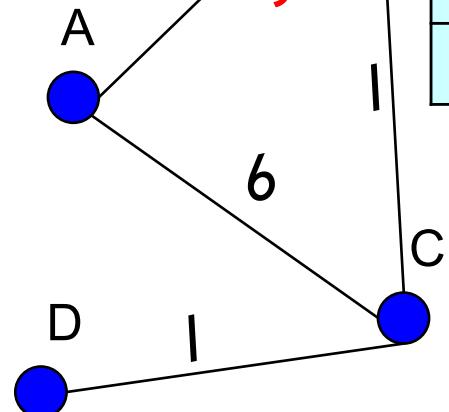
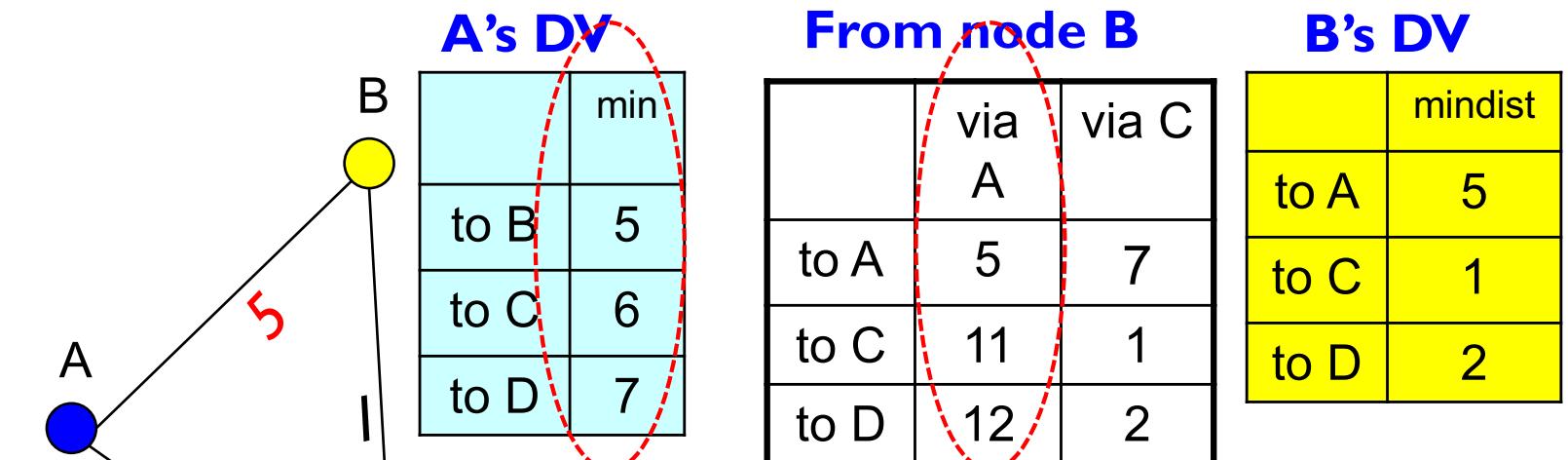
|      | via B    | via C    |
|------|----------|----------|
| to B | $c(A,B)$ | $\infty$ |
| to C | $\infty$ | $c(A,C)$ |

A's DV

|      | mindist |
|------|---------|
| to B | 5       |
| to C | 6       |
| to D | 7       |

Each router sends its DV to its immediate neighbors

# How Distance-Vector (DV) works



Routers process received DVs

# Distance Vector Routing

- ❖ Each router knows the links to its neighbors
- ❖ Each router has provisional “shortest path” to **every** other router -- its **distance vector (DV)**
- ❖ Routers exchange this DV with their neighbors
- ❖ Routers look over the set of options offered by their neighbors and select the best one
- ❖ Iterative process converges to set of shortest paths

# Distance Vector Routing

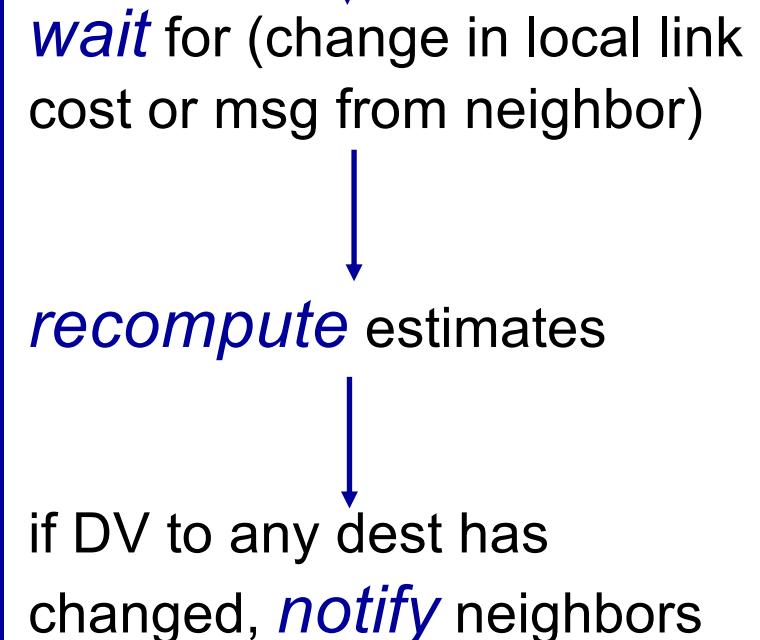
*iterative, asynchronous:*

- each local iteration caused by:
  - ❖ local link cost change
  - ❖ DV update message from neighbor

*distributed:*

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

*each node:*



# Distance Vector

- ❖  $c(i,j)$ : link cost from node  $i$  to  $j$
- ❖  $\text{dist}_Z(A,V)$ : shortest dist. from  $A$  to  $V$  via  $Z$
- ❖  $\text{mindist}(A,V)$ : shortest dist. from  $A$  to  $V$

## 0 At node A

1 **Initialization:**

```
2 for all destinations  $V$  do
3     if  $V$  is neighbor of  $A$ 
4          $\text{dist}_V(A, V) = \text{mindist}(A, V) = c(A, V);$ 
5     else
6          $\text{dist}_V(A, V) = \text{mindist}(A, V) = \infty;$ 
7     send  $\text{mindist}(A, *)$  to all neighbors
```

*loop:*

```
8 wait (until  $A$  sees a link cost change to neighbor  $V$  /* case 1 */
9     or until  $A$  receives  $\text{mindist}(V, *)$  from neighbor  $V$ ) /* case 2 */
10 if ( $c(A, V)$  changes by  $\pm d$ ) /*  $\Leftarrow$  case 1 */
11     for all destinations  $Y$  do
12          $\text{dist}_V(A, Y) = \text{dist}_V(A, Y) \pm d$ 
13 else /*  $\Leftarrow$  case 2: */
14     for all destinations  $Y$  do
15          $\text{dist}_V(A, Y) = c(A, V) + \text{mindist}(V, Y);$ 
16 update  $\text{mindist}(A, *)$ 
15 if (there is a change in  $\text{mindist}(A, *)$ )
16     send  $\text{mindist}(A, *)$  to all neighbors
17 forever
```

# Distance Vector: How Does it Work

---

- Periodically, each router sends its DV (destination, distance columns) to *directly connected* routers
- When router  $K$  receives DV from router  $J$ ,  $K$  updates its table if:
  - $J$  knows a shorter route for a given destination
  - $J$  knows a destination  $K$  didn't know about
  - $K$  currently routes to a destination through  $J$  and  $J$ 's distance to that destination has changed

# Router Message Exchange

Routing Table for Router K

| To    | Distance | Route  |
|-------|----------|--------|
| Net1  | 0        | Direct |
| Net2  | 0        | Direct |
| Net4  | 8        | L      |
| Net17 | 5        | M      |
| Net24 | 6        | J      |
| Net30 | 2        | Q      |
| Net42 | 2        | J      |

Update from Router J

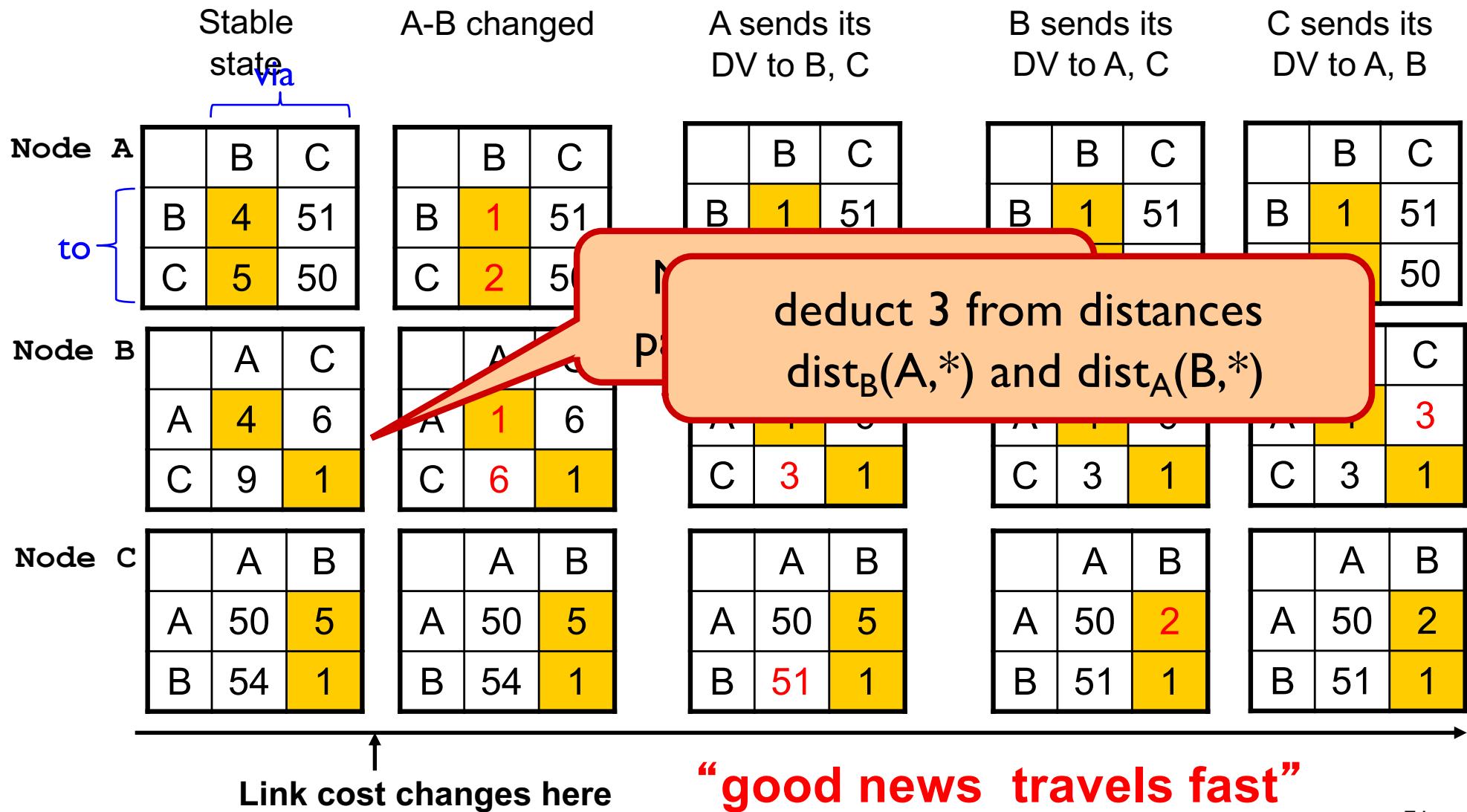
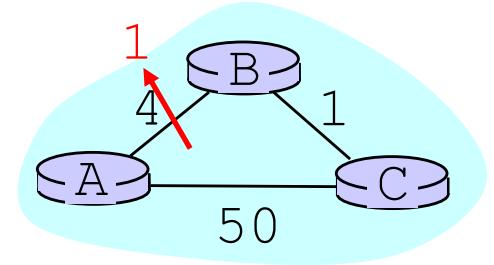
| To    | Distance |
|-------|----------|
| Net1  | 1        |
| Net4  | 3        |
| Net17 | 6        |
| Net21 | 4        |
| Net24 | 5        |
| Net30 | 10       |
| Net42 | 3        |

- If distance is  $N$  for  $J$ , it is  $N+C_j$  for  $K$ 
  - packet has to go through router  $J$
- If  $K$  updates or adds an entry in response to  $J$ 's message, it assigns Route= $J$

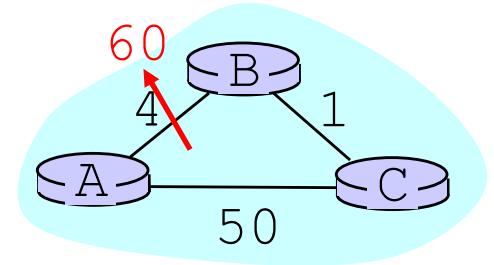
# Problems with Distance Vector

- A number of problems can occur in a network using distance vector algorithm
- Most of these problems are caused by slow convergence or routers converging on incorrect information
- *Convergence* is the time during which all routers come to an agreement about the best paths through the internetwork
  - whenever topology changes there is a period of instability in the network as the routers converge
- Reacts rapidly to good news, but leisurely to bad news

# DV: Link Cost Changes



# DV: Link Cost Changes



Stable state via

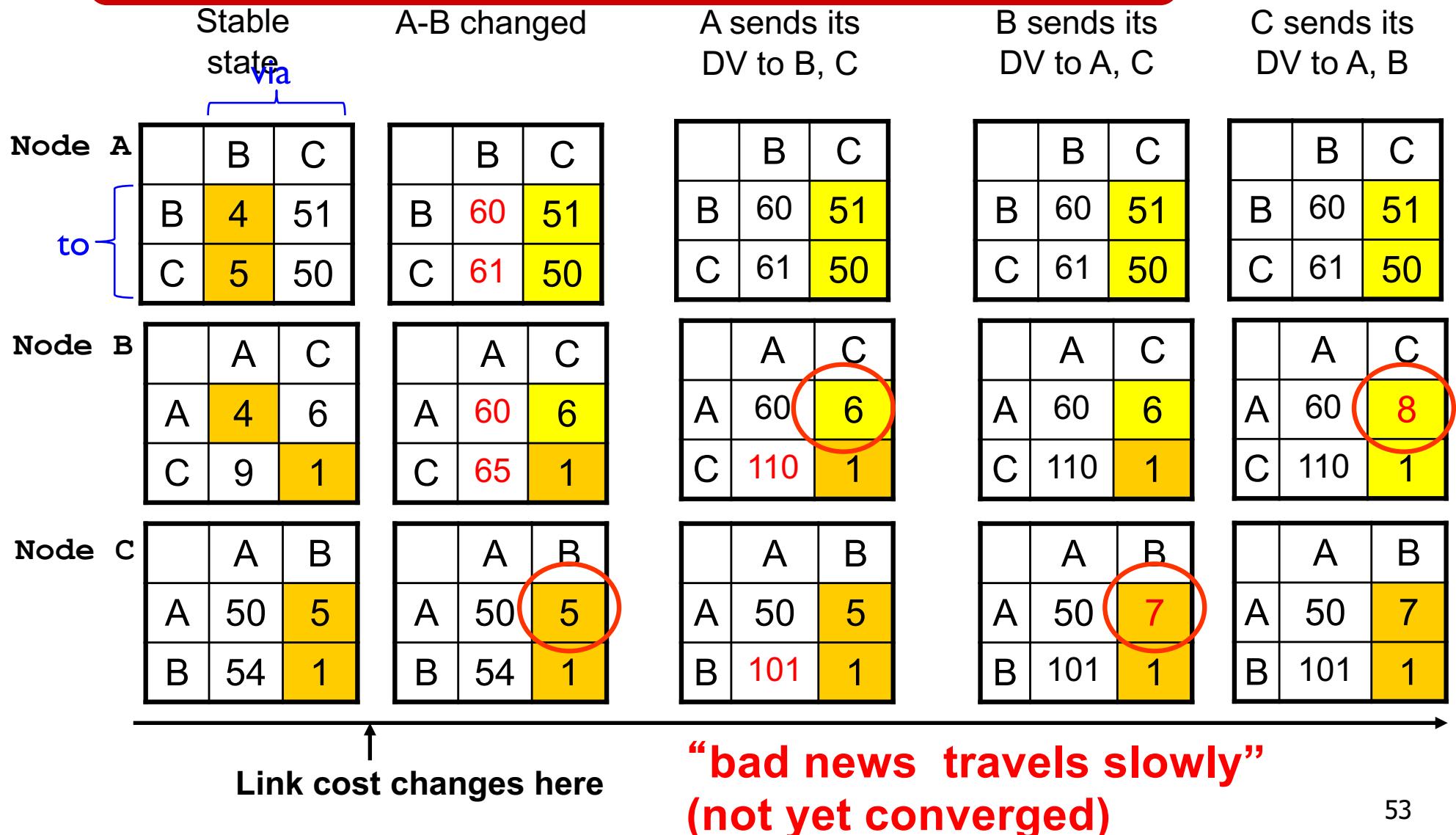
A-B changed

|        |  |                        |   |   |   |    |    |   |    |    |  |  |   |   |   |    |    |   |    |    |
|--------|--|------------------------|---|---|---|----|----|---|----|----|--|--|---|---|---|----|----|---|----|----|
| Node A | <table border="1"> <tr><td></td><td>B</td><td>C</td></tr> <tr><td>B</td><td>4</td><td>51</td></tr> <tr><td>C</td><td>5</td><td>50</td></tr> </table> |                        | B | C | B | 4  | 51 | C | 5  | 50 | <table border="1"> <tr><td></td><td>B</td><td>C</td></tr> <tr><td>B</td><td>60</td><td>51</td></tr> <tr><td>C</td><td>61</td><td>50</td></tr> </table> |  | B | C | B | 60 | 51 | C | 61 | 50 |
|        | B  | C                      |   |   |   |    |    |   |    |    |  |  |   |   |   |    |    |   |    |    |
| B      | 4  | 51                     |   |   |   |    |    |   |    |    |  |  |   |   |   |    |    |   |    |    |
| C      | 5  | 50                     |   |   |   |    |    |   |    |    |  |  |   |   |   |    |    |   |    |    |
|        | B  | C                      |   |   |   |    |    |   |    |    |  |  |   |   |   |    |    |   |    |    |
| B      | 60   | 51                     |   |   |   |    |    |   |    |    |  |  |   |   |   |    |    |   |    |    |
| C      | 61   | 50                     |   |   |   |    |    |   |    |    |  |  |   |   |   |    |    |   |    |    |
| to     |  |                        |   |   |   |    |    |   |    |    |  |  |   |   |   |    |    |   |    |    |
| Node B | <table border="1"> <tr><td></td><td>A</td><td>C</td></tr> <tr><td>A</td><td>4</td><td>6</td></tr> <tr><td>C</td><td>9</td><td>1</td></tr> </table>   |                        | A | C | A | 4  | 6  | C | 9  | 1  | <table border="1"> <tr><td></td><td>A</td><td>C</td></tr> <tr><td>A</td><td>60</td><td>6</td></tr> <tr><td>C</td><td>65</td><td>1</td></tr> </table>   |  | A | C | A | 60 | 6  | C | 65 | 1  |
|        | A  | C                      |   |   |   |    |    |   |    |    |  |  |   |   |   |    |    |   |    |    |
| A      | 4  | 6                      |   |   |   |    |    |   |    |    |  |  |   |   |   |    |    |   |    |    |
| C      | 9  | 1                      |   |   |   |    |    |   |    |    |  |  |   |   |   |    |    |   |    |    |
|        | A  | C                      |   |   |   |    |    |   |    |    |  |  |   |   |   |    |    |   |    |    |
| A      | 60   | 6                      |   |   |   |    |    |   |    |    |  |  |   |   |   |    |    |   |    |    |
| C      | 65   | 1                      |   |   |   |    |    |   |    |    |  |  |   |   |   |    |    |   |    |    |
| Node C | <table border="1"> <tr><td></td><td>A</td><td>B</td></tr> <tr><td>A</td><td>50</td><td>5</td></tr> <tr><td>B</td><td>54</td><td>1</td></tr> </table> |                        | A | B | A | 50 | 5  | B | 54 | 1  | <table border="1"> <tr><td></td><td>A</td><td>B</td></tr> <tr><td>A</td><td>50</td><td>5</td></tr> <tr><td>B</td><td>54</td><td>1</td></tr> </table>   |  | A | B | A | 50 | 5  | B | 54 | 1  |
|        | A  | B                      |   |   |   |    |    |   |    |    |  |  |   |   |   |    |    |   |    |    |
| A      | 50   | 5                      |   |   |   |    |    |   |    |    |  |  |   |   |   |    |    |   |    |    |
| B      | 54   | 1                      |   |   |   |    |    |   |    |    |  |  |   |   |   |    |    |   |    |    |
|        | A  | B                      |   |   |   |    |    |   |    |    |  |  |   |   |   |    |    |   |    |    |
| A      | 50   | 5                      |   |   |   |    |    |   |    |    |  |  |   |   |   |    |    |   |    |    |
| B      | 54   | 1                      |   |   |   |    |    |   |    |    |  |  |   |   |   |    |    |   |    |    |
|        |  | Link cost changes here |   |   |   |    |    |   |    |    |  |  |   |   |   |    |    |   |    |    |

add 56 to distances  
 $\text{dist}_B(A,*)$  and  $\text{dist}_A(B,*)$

# DV: Link Cost Changes

This is the “Counting to Infinity” Problem



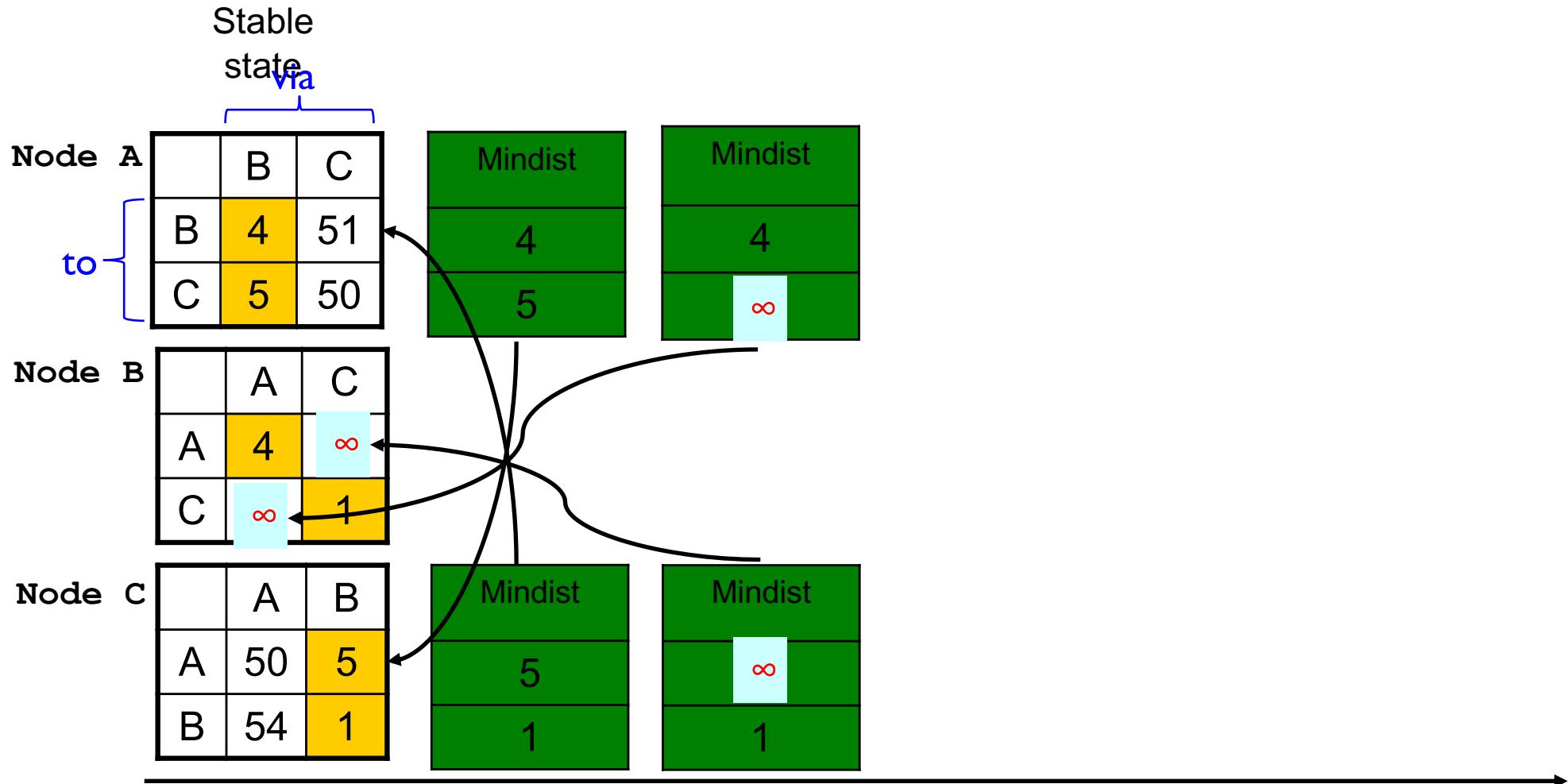
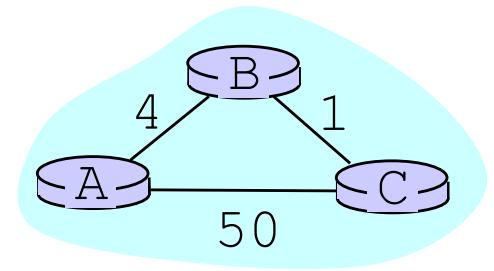
## The “Poisoned Reverse” Rule

- ❖ Heuristic to avoid count-to-infinity
- ❖ If B routes via C to get to A:
  - B tells C its (B's) distance to A is infinite  
(so C won't route to A via B)

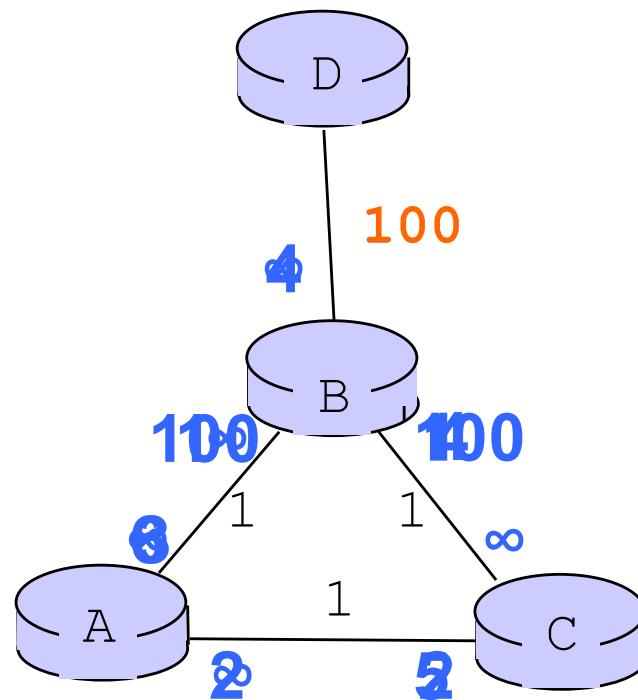
# DV: Poisoned Reverse

If B routes through C to get to A:

B tells C its (B's) distance to A is infinite



# Will Poison-Reverse Completely Solve the Count-to-Infinity Problem?



Numbers in blue denote the best cost to destination D advertised along the link

# Comparison of LS and DV algorithms

## *message complexity*

- ❖ **LS:** with  $n$  nodes,  $E$  links,  $O(nE)$  msgs sent
- ❖ **DV:** exchange between neighbors only
  - convergence time varies

## *speed of convergence*

- ❖ **LS:**  $O(n^2)$  algorithm requires  $O(nE)$  msgs
  - may have oscillations
- ❖ **DV:** convergence time varies
  - may be routing loops
  - count-to-infinity problem

## *robustness: what happens if router malfunctions?*

### *LS:*

- node can advertise incorrect *link* cost
- each node computes only its own table

### *DV:*

- DV node can advertise incorrect *path* cost
- each node's table used by others
  - error propagate thru network

# Real Protocols

*Link State*

Open Shortest Path  
First (OSPF)

Intermediate system to  
intermediate system (IS-  
IS)

*Distance Vector*

Routing Information  
Protocol (RIP)

Interior Gateway  
Routing Protocol  
(IGRP-Cisco)

Border Gateway  
Protocol (BGP)

# Network layer, control plane: outline

5.1 introduction

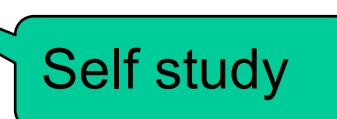
5.2 routing protocols

- ❖ link state

- ❖ distance vector

- ❖ hierarchical routing

5.6 ICMP: The Internet  
Control Message  
Protocol

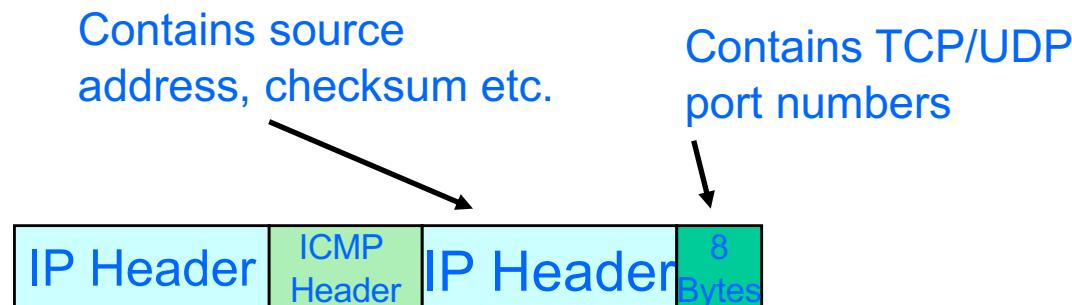


Self study

# ICMP: Internet Control Message Protocol

---

- ❖ Used by hosts & routers to communicate network level information
  - Error reporting: unreachable host, network, port
  - Echo request/reply (used by ping)
- ❖ Works above IP layer
  - ICMP messages carried in IP datagrams
- ❖ ICMP message: type, code plus IP header and first 8 bytes of IP datagram payload causing error



# ICMP: Internet Control Message Protocol

---

| Type | Code | Description                   |
|------|------|-------------------------------|
| 0    | 0    | echo reply(ping)              |
| 3    | 0    | dest. network unreachable     |
| 3    | 1    | dest host unreachable         |
| 3    | 3    | dest port unreachable         |
| 3    | 4    | frag needed; DF set           |
| 8    | 0    | echo request(ping)            |
| 11   | 0    | TTL expired                   |
| 11   | 1    | frag reassembly time exceeded |
| 12   | 0    | bad IP header                 |

# Traceroute and ICMP

- Source sends series of UDP segments to dest
    - first set has TTL = 1
    - second set has TTL=2, etc.
    - unlikely port number
  - When  $n$ th set of datagrams arrives to  $n$ th router:
    - router discards datagrams
    - and sends source ICMP messages (type 11, code 0)
    - ICMP messages includes IP address of router
  - when ICMP messages 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

