

COMP/ELEC 429/556

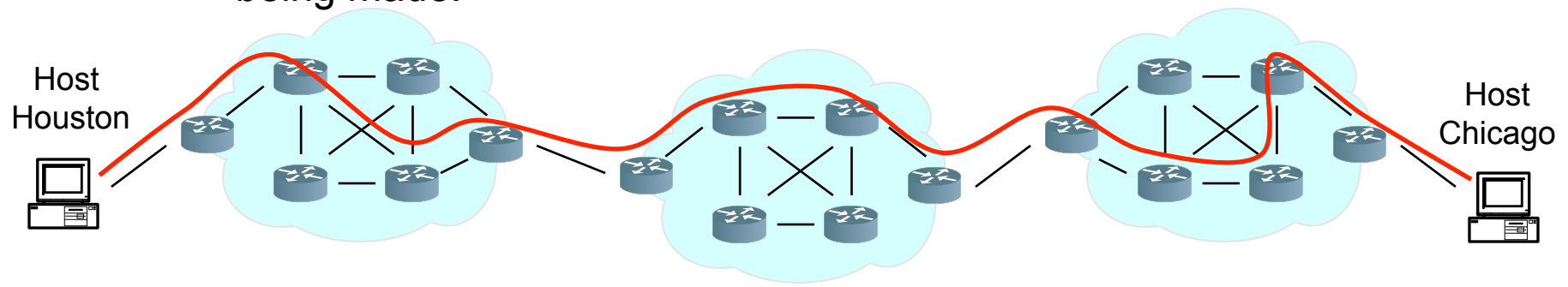
Introduction to Computer Networks

Intra-domain routing

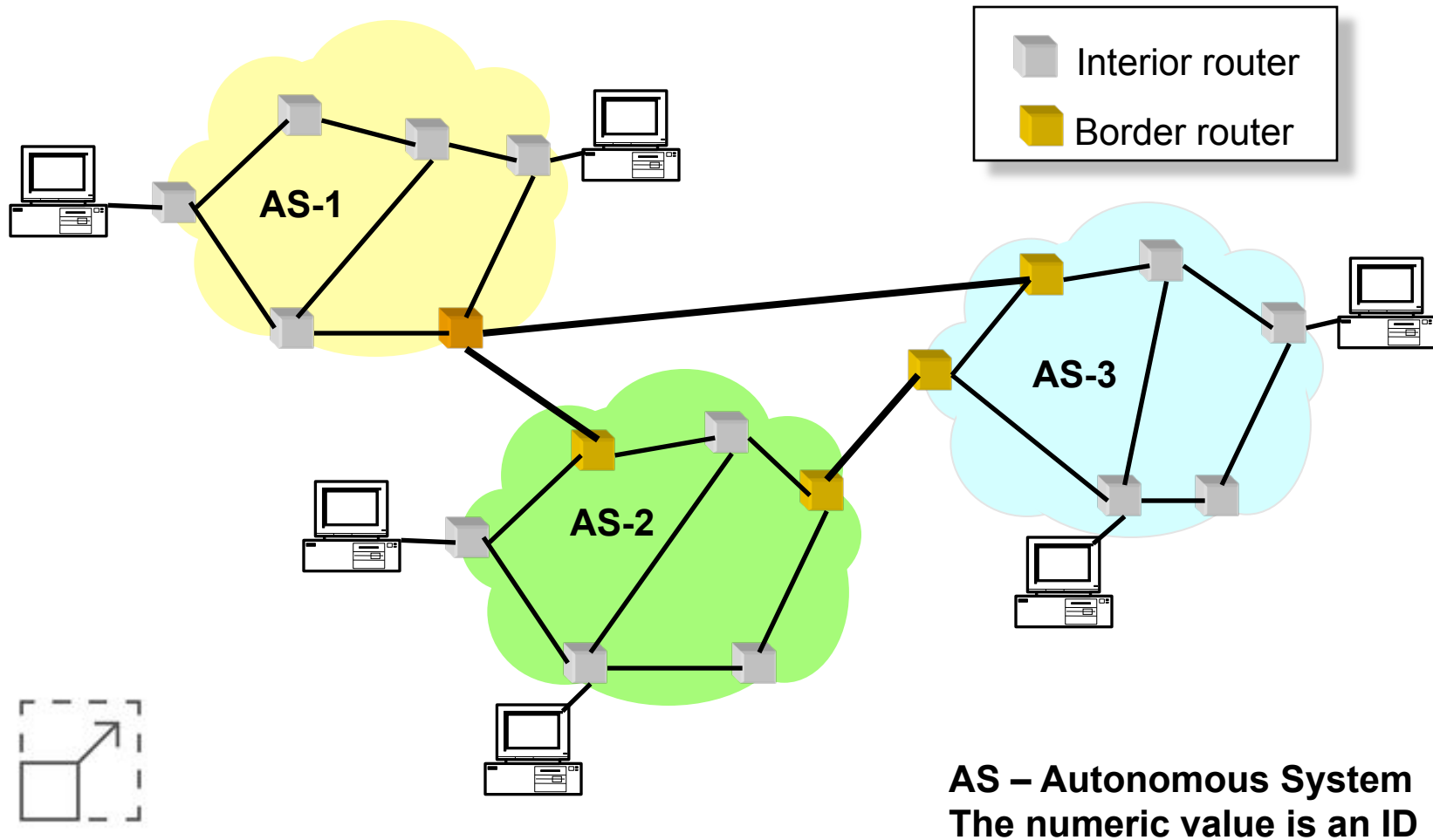
Some slides used with permissions from Edward W.
Knightly, T. S. Eugene Ng, Ion Stoica, Hui Zhang

What is routing?

- Routing
 - The process of determining the path to send data from a source to a destination
 - Routing decides the contents of the forwarding tables in routers
- Different from “Forwarding”
 - “Forwarding” is the act of moving packets from input to output according to a forwarding table
 - Forwarding happens at both IP routers and at Ethernet bridges
 - Ethernet performs forwarding, but not routing: Ethernet bridges flood a packet through the spanning tree, bridges learn the direction to the source and fill forwarding tables. There is no intentional path choice being made.



Two Level Hierarchy of Internet Routing



Why Need 2 Levels?

- Routing protocols are not efficient enough to deal with the size of the entire Internet
- Different organizations may want different internal routing policies
- Allow organizations to hide their internal network configurations from outside
- Allow organizations to choose how to route across multiple AS' s
- More scalable, more autonomy/independence

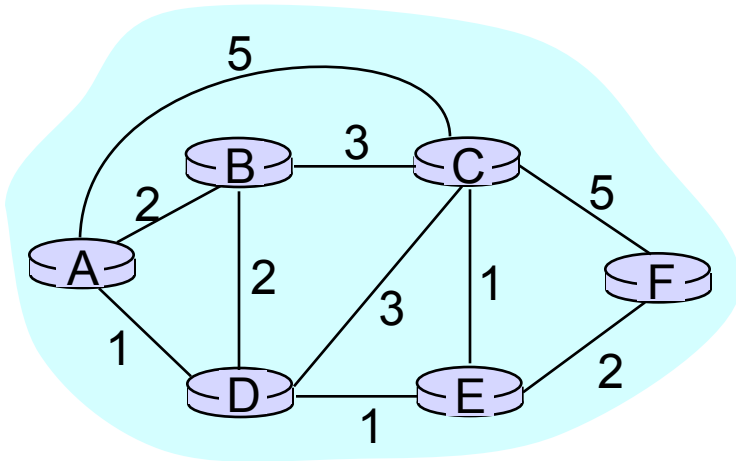


Two Level Hierarchy of Internet Routing

- First level – autonomous systems (AS)
 - AS: region of network under a single administrative domain
 - e.g. Rice's IP network, AT&T's US backbone IP network
- Each AS performs **intra-domain** routing
 - e.g. Routing Information Protocol (RIP)
 - e.g. Open Shortest Path First (OSPF) Protocol
 - e.g. Static routing, centralized routing...
- Second level – inter-connected ASs
- Between ASs perform **inter-domain** routing
 - e.g. Border Gateway Protocol (BGP)
 - De facto standard today, BGP-4

Routing Policy

- Given multiple alternative paths, which paths are chosen to route packets to destinations is a policy decision
- What are some possible policies?
 - Shortest path
 - Most load-balanced
 - Satisfies app's delay/bandwidth requirements
- Need routing protocols to realize these policies



Network modeled as a graph

- Routers → nodes
- Link → edges
 - Edge cost: delay, congestion level,...

Outline for this Module

- Two approaches for intra-domain routing
- Both try to achieve the “shortest path” routing policy
- Link state routing protocol
- Distance vector routing protocol
- In Project 3, you will get to implement fairly realistic variants of these and really understand how they work!
 - Distributed coordination in action



General Assumptions

- Nodes know only the cost to their neighbors; not the topology
- Nodes can talk only to their neighbors using messages
- All nodes run the same algorithm concurrently
- Nodes / links may fail, messages may be lost

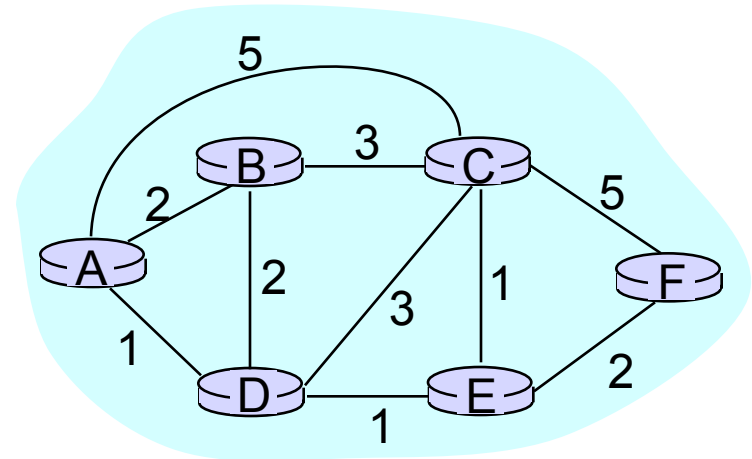
Link State Routing

- Nodes **flood**, in a controlled manner, topology information in the form of link state packets
 - Each node learns full topology
 - Do not confuse this with Ethernet spanning tree data packet flooding
- Each node **computes** its own routing table to realize the shortest path routing policy
 - By Dijkstra's algorithm


Link State Flooding

- Each node knows its connectivity and cost to a direct neighbor
 - How?
- Every node tells every other node this local connectivity/cost information
 - Via controlled flooding
- In the end, every node learns the complete topology of the network
- E.g. “A” floods message:

A connected to B cost 2
A connected to D cost 1
A connected to C cost 5

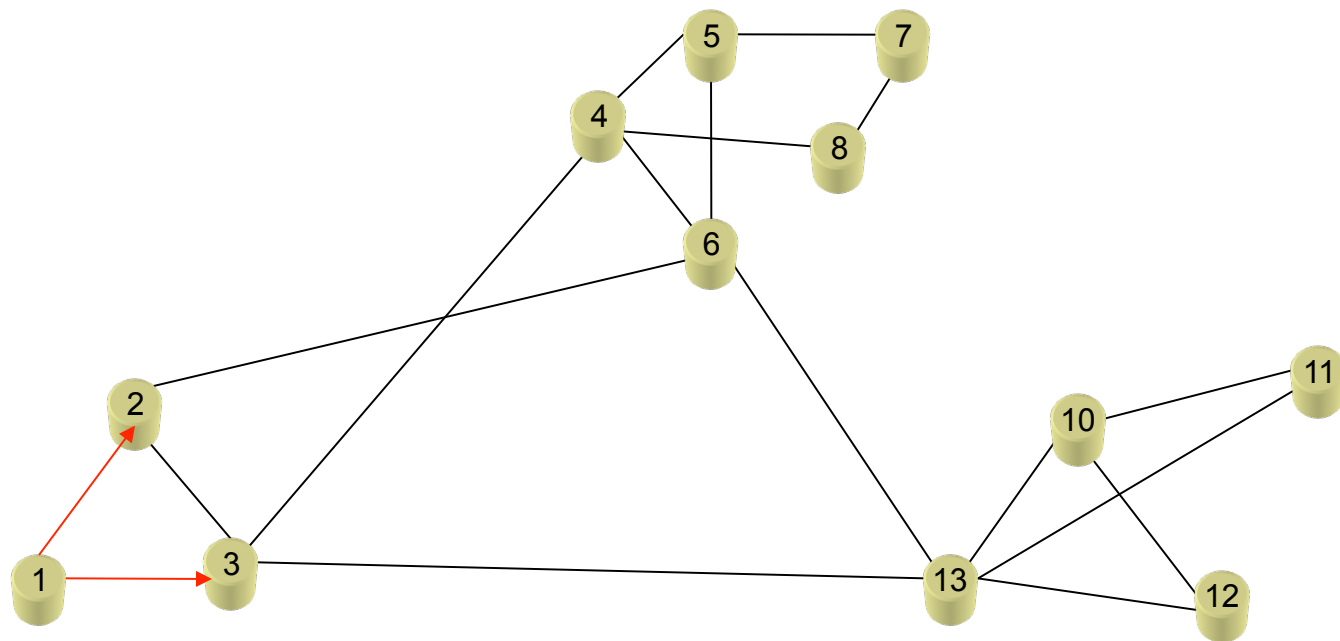


Flooding Details

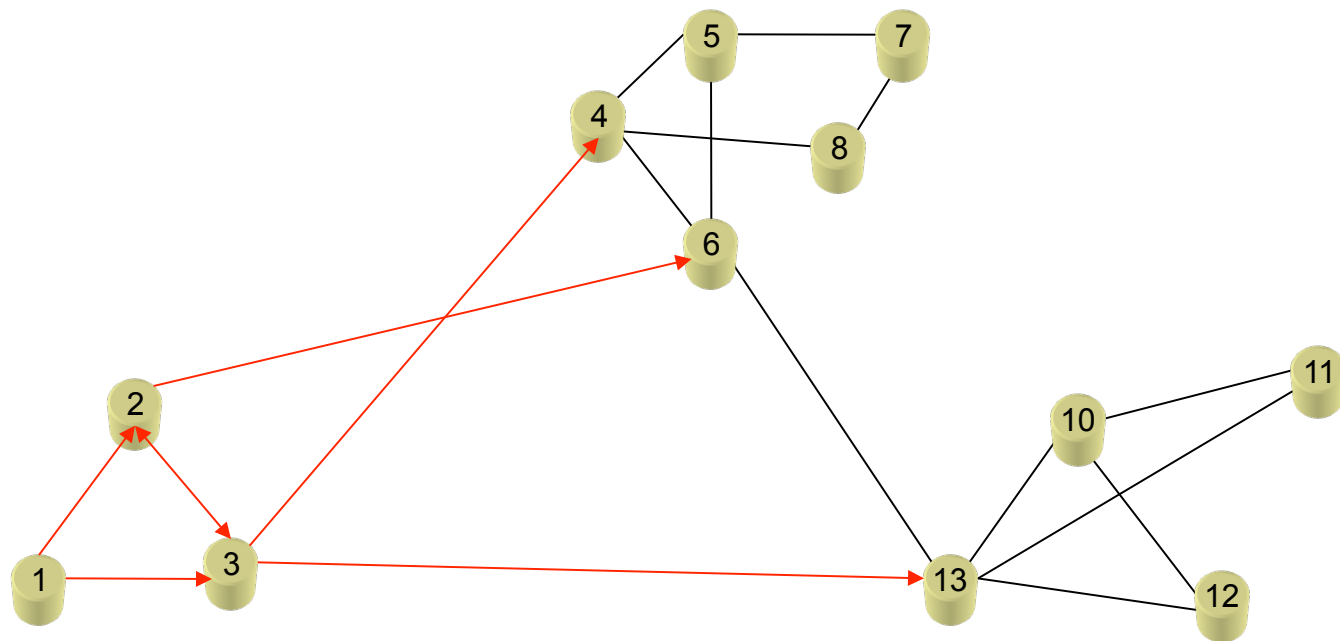
- Each node periodically generates Link State Packet (LSP) that contains
 - ID of node created LSP
 - List of direct neighbors and costs
 - Sequence number (say 64 bit, assume to never wrap around)
 - Incremented when a new LSP is created
- Sequence number used to identify newer LSP
 - An older LSP with a sequence number previously seen is discarded
 - What if a router reboots and lost the last sequence number used? 
- Receiving node flood newer LSP to all its neighbors except the neighbor where the LSP came from
- LSP is also triggered on-demand when a link's state changes (failed or restored)

From A Seq. #	
B	2
C	5
D	1

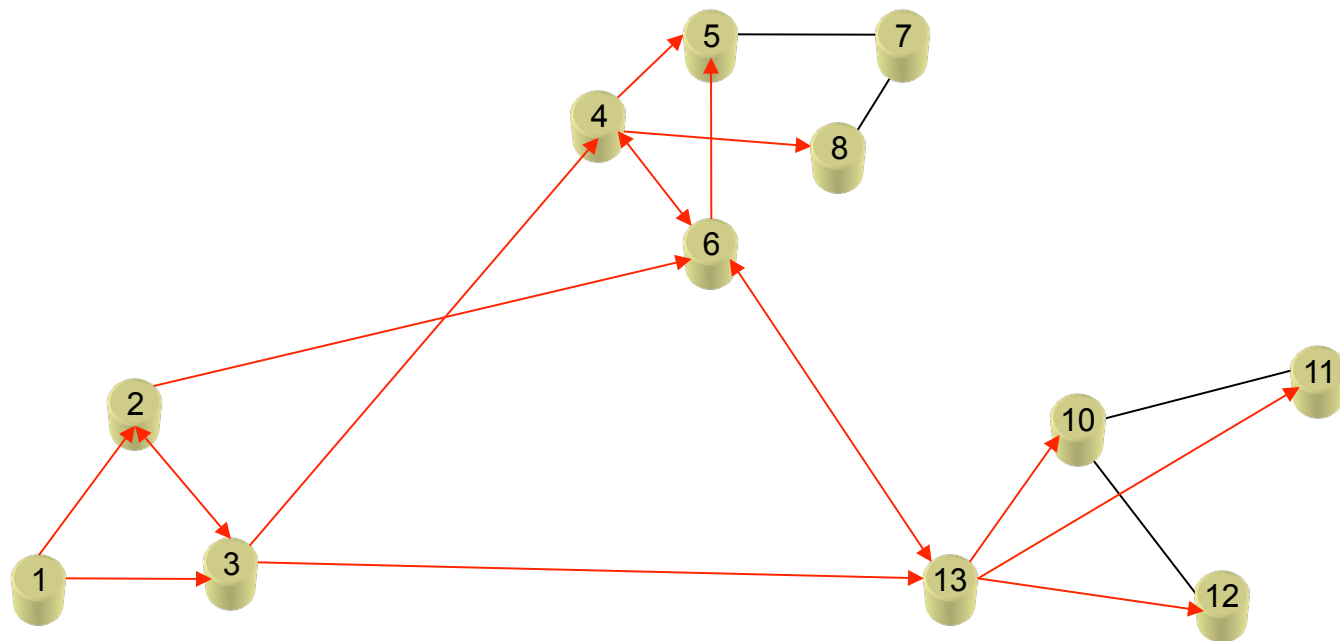
Link State Flooding Example



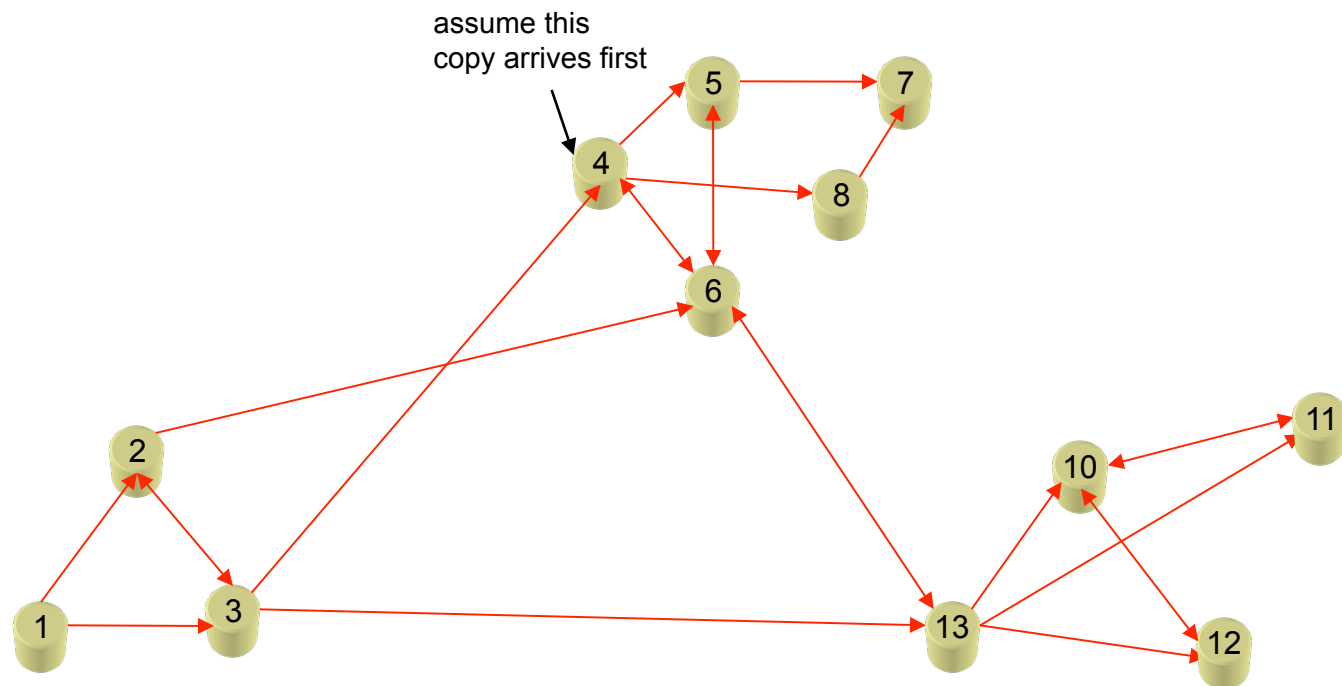
Link State Flooding Example



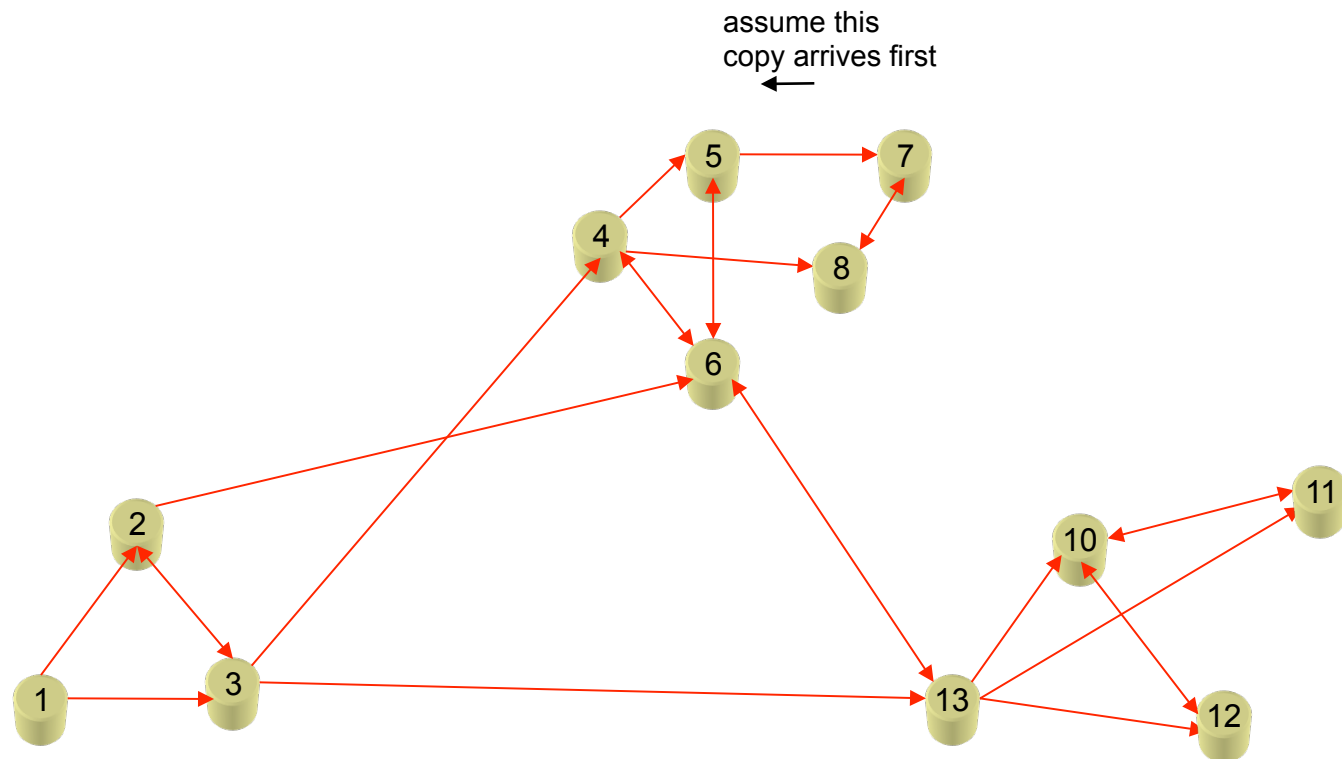
Link State Flooding Example



Link State Flooding Example



Link State Flooding Example



Computing Shortest Path Routes with Dijkstra's Shortest Path Algorithm

Dijkstra's algorithm

- Network topology, link costs learned by all nodes
 - Accomplished via “link state flooding”
- Each node computes least cost paths from itself to all other nodes in the topology

Notations

- $c(i,j)$: link cost from node i to j ; cost infinite if not direct neighbors
- $D(v)$: current value of cost of path from source to node v
- $p(v)$: predecessor node along path from source to v , that is next to v
- S : set of nodes whose least cost path definitively known

Dijkstra's Algorithm

1 **Initialization:**

2 $S = \{A\};$

3 for all nodes v

4 if v adjacent to A

5 then $D(v) = c(A,v);$

6 else $D(v) = \infty ;$

7

8 **Loop**

9 find w not in S such that $D(w)$ is a minimum;

10 add w to S ;

11 update $D(v)$ for all v adjacent to w and not in S :

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

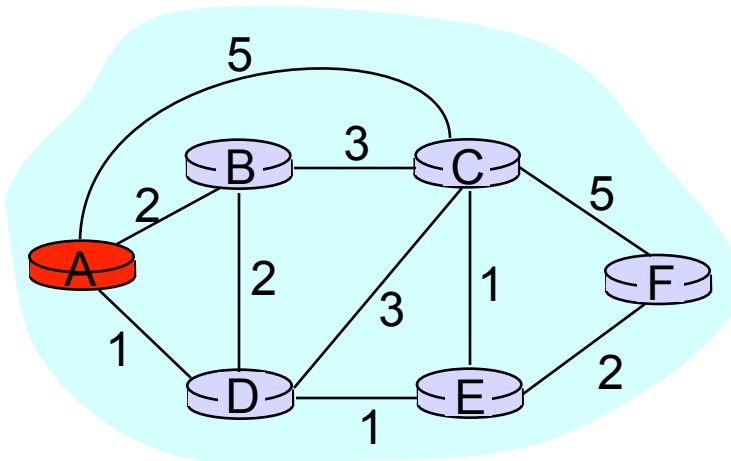
// new cost to v is either old cost to v or known

// shortest path cost to w plus cost from w to v

13 **until all nodes in S ;**

Example: Dijkstra's Algorithm

Step	start S	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						
2						
3						
4						
5						



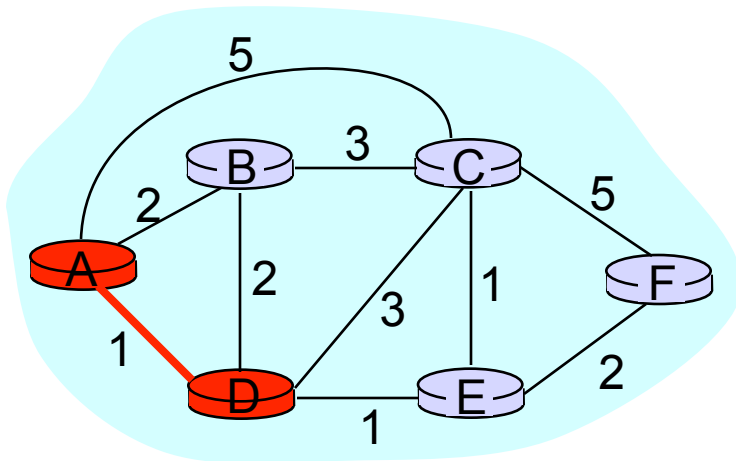
```

1 Initialization:
2 S = {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	start S	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		4,D		2,D	∞
2						
3						
4						
5						



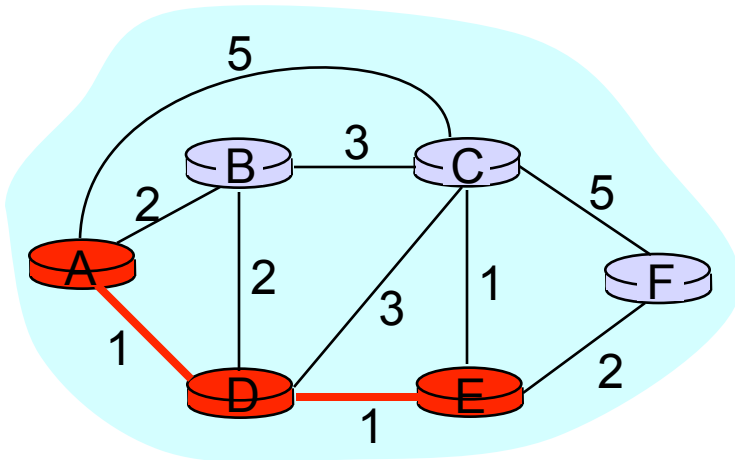
```

...
8  Loop
9  find w not in S s.t. D(w) is a minimum;
10 add w to S;
1  update D(v) for all v adjacent
   to w and not in S:
12   D(v) = min( D(v), D(w) + c(w,v) );
13 until all nodes in S;

```

Example: Dijkstra's Algorithm

Step	start S	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		4,D		2,D	∞
→ 2	ADE		3,E			4,E
3						
4						
5						



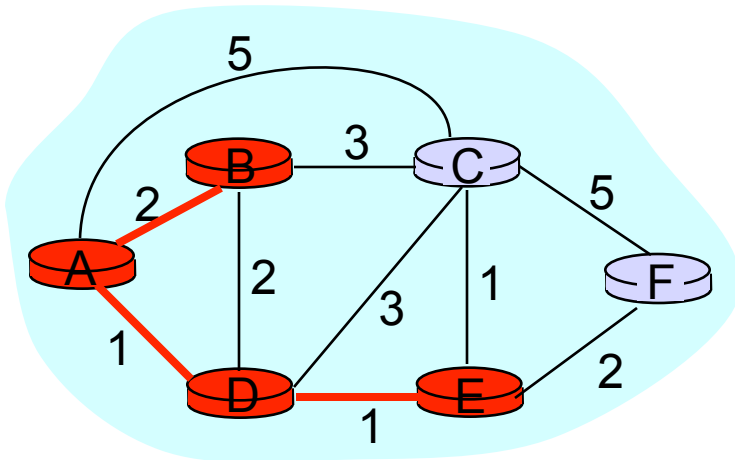
```

...
8  Loop
9  find w not in S s.t. D(w) is a minimum;
10 add w to S;
1  update D(v) for all v adjacent
   to w and not in S:
12   D(v) = min( D(v), D(w) + c(w,v) );
13 until all nodes in S;

```

Example: Dijkstra's Algorithm

Step	start S	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		4,D		2,D	∞
2	ADE		3,E			4,E
→ 3	ADEB					
4						
5						



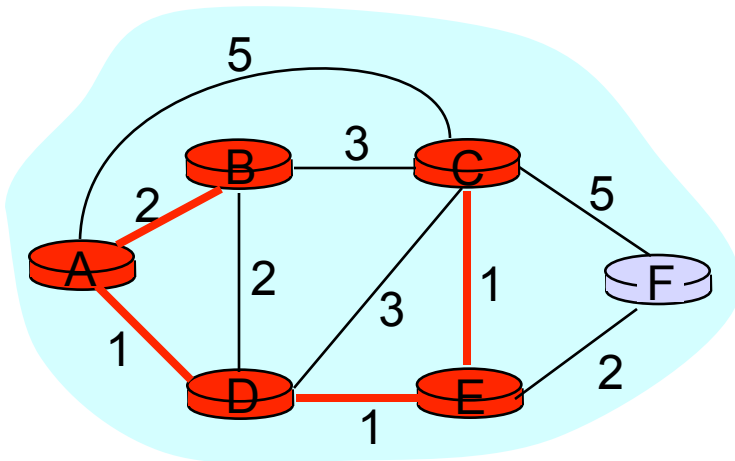
```

...
8  Loop
9  find w not in S s.t. D(w) is a minimum;
10 add w to S;
1  update D(v) for all v adjacent
   to w and not in S:
12   D(v) = min( D(v), D(w) + c(w,v) );
13 until all nodes in S;

```

Example: Dijkstra's Algorithm

Step	start S	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		4,D		2,D	∞
2	ADE		3,E			4,E
3	ADEB					
→ 4	ADEBC					
5						



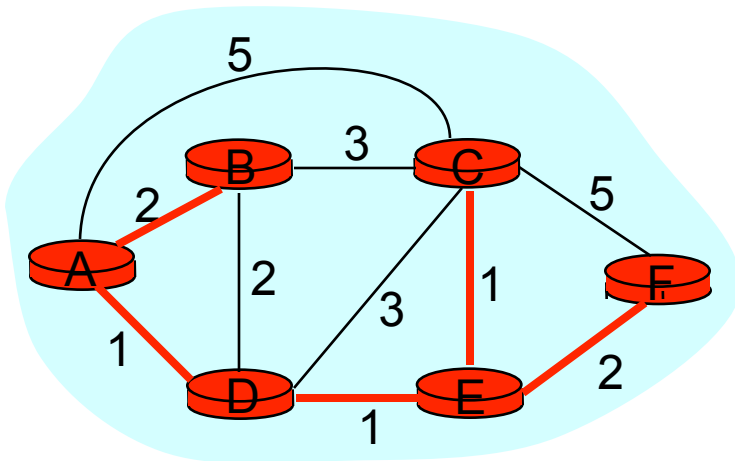
```

...
8  Loop
9  find w not in S s.t. D(w) is a minimum;
10 add w to S;
1  update D(v) for all v adjacent
   to w and not in S:
12   D(v) = min( D(v), D(w) + c(w,v) );
13 until all nodes in S;

```

Example: Dijkstra's Algorithm

Step	start S	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		4,D		2,D	∞
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
→ 5	ADEBCF					



```

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

```


Distance Vector Routing

- What is a distance vector?
 - Current best known cost to get to a destination

e.g. at a node C

Dest.	Cost
A	7
D	1
J	2
K	5
Q	1
T	3

At the beginning, distance vector only has information about directly attached neighbors

Eventually the vector is filled

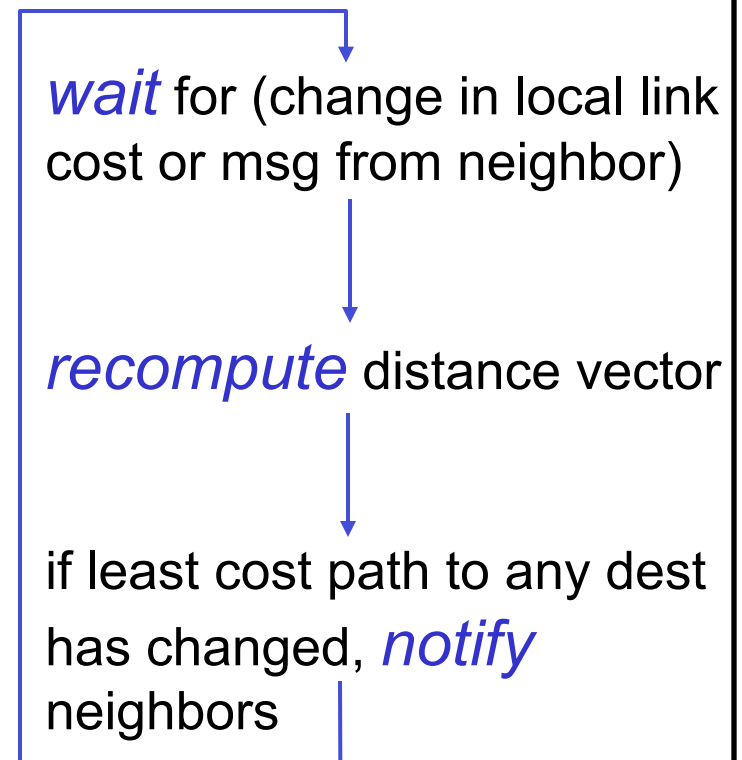
Distance Vector Routing

- Each router maintains
 - Row for each possible destination
 - Column for each directly-attached neighbor
 - Entry in row Y and column Z of node X represents best known distance from X to Y, via Z as next hop
- Idea: Exchange distance vectors among neighbors to learn about lowest cost paths
 - Periodically to refresh information
 - On-demand when distance vector information changes
- Iteratively exchange distance vectors until no new information is learned

Distance Vector Routing

- Each local iteration caused by:
 - Local link cost change
 - DV message from neighbor: its least cost path to destination changed
- Each node notifies neighbors when its least cost path to any destination changes
 - Neighbors then notify their neighbors if necessary
- Periodic messages used to refresh information

Each node:



Distance Vector Algorithm

1 **Initialization:**

2 **for all** nodes V **do**

3 **if** V adjacent to A

4 $D(A, V, V) = c(A, V);$ */* Distance from A to V via neighbor V */*

5 **else**

• $D(A, V, *) = \infty;$

loop:

→ 8 **wait** (until A sees a link cost change to neighbor V
9 or until A receives update from neighbor V)

10 **if** ($c(A, V)$ changes by d)

11 **for all** destinations Y through V **do**

12 $D(A, Y, V) = D(A, Y, V) + d$

13 **else if** (update $D(V, Y)$ received from V)

/ shortest path from V to some Y has changed */*

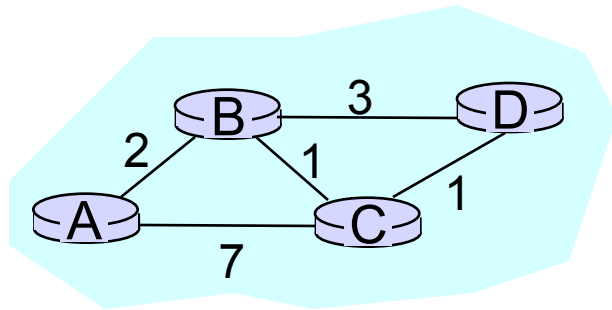
14 $D(A, Y, V) = c(A, V) + D(V, Y);$

15 **if** (there is a new minimum for destination Y)

16 **send** new message containing $D(A, Y)$ to all neighbors */* $D(A, Y)$ denotes
the min $D(A, Y, *)$ */*

17 **forever**

Example: Distance Vector Algorithm



Node A

Dest.	Cost	NextHop
B	2	B
C	7	C
D	∞	-

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	3	D

Node C

Dest.	Cost	NextHop
A	7	A
B	1	B
D	1	D

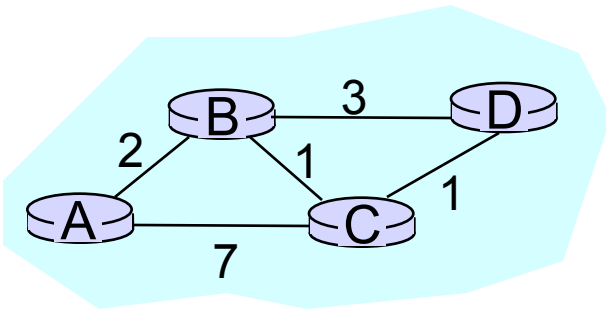
Node D

Dest.	Cost	NextHop
A	∞	-
B	3	B
C	1	C

```

1 Initialization:
2 for all nodes  $V$  do
3   if  $V$  adjacent to  $A$ 
4      $D(A, V, V) = c(A, V)$ ;
5   else
6      $D(A, V, *) = \infty$ ;
...
  
```

Example: 1st Iteration (C → A)



7 **loop:**

```

...
13 else if (update D(V, Y) received from V)
14   D(A, Y, V) = c(A, V) + D(V, Y);
15 if (there is a new min. for destination Y)
16   send D(A, Y) to all neighbors
17 forever
  
```

Node A

Dest.	Cost	NextHop
B	2	B
C	7	C
D	8	C

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	3	D

$$D(A, D, C) = c(A, C) + D(C, D) = 7 + 1 = 8$$

(D(C, A), D(C, B), D(C, D))

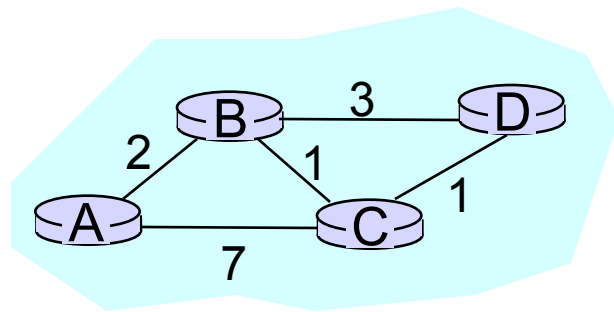
Node C

Dest.	Cost	NextHop
A	7	A
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	∞	-
B	3	B
C	1	C

Example: 1st Iteration (B→A, C→A)



Node A

Dest.	Cost	NextHop
B	2	B
C	3	B
D	5	B

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	3	D

$$D(A,D,B) = c(A,B) + D(B,D) = 2 + 3 = 5$$

$$D(A,C,B) = c(A,B) + D(B,C) = 2 + 1 = 3$$

Node C

Dest.	Cost	NextHop
A	7	A
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	∞	-
B	3	B
C	1	C

7 **loop:**

...

13 **else if** (update $D(V, Y)$ received from V)

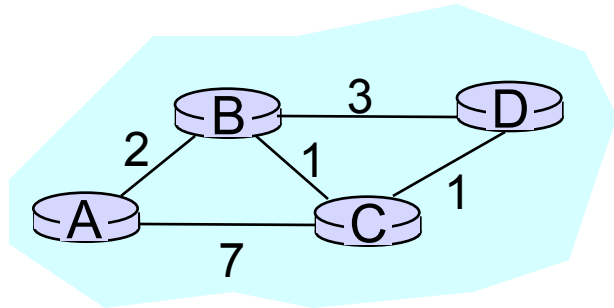
14 $D(A, Y, V) = c(A, V) + D(V, Y)$

15 **if** (there is a new min. for destination Y)

16 **send** $D(A, Y)$ to all neighbors

17 **forever**

Example: End of 1st Iteration



7 **loop:**

```

...
13 else if (update D(V, Y) received from V)
14   D(A, Y, V) = c(A, V) + D(V, Y);
15 if (there is a new min. for destination Y)
16   send D(A, Y) to all neighbors
17 forever
  
```

Node A

Dest.	Cost	NextHop
B	2	B
C	3	B
D	5	B

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	2	C

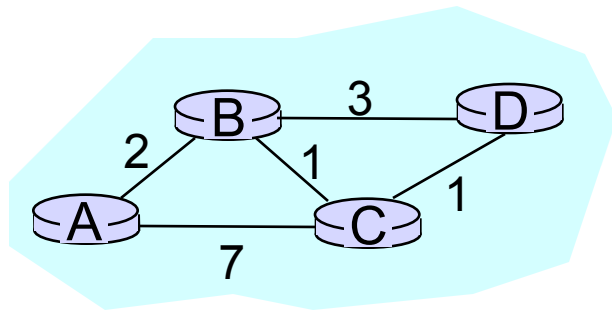
Node C

Dest.	Cost	NextHop
A	3	B
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	5	B
B	2	C
C	1	C

Example: End of 2nd Iteration



7 **loop:**

...

```

13 else if (update  $D(V, Y)$  received from  $V$ )
14    $D(A, Y, V) = c(A, V) + D(V, Y)$ ;
15 if (there is a new min. for destination  $Y$ )
16   send  $D(A, Y)$  to all neighbors
17 forever
  
```

Node A

Dest.	Cost	NextHop
B	2	B
C	3	B
D	4	B

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	2	C

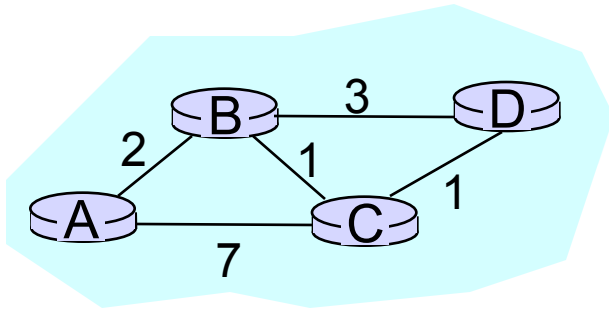
Node C

Dest.	Cost	NextHop
A	3	B
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	4	C
B	2	C
C	1	C

Example: End of 3rd Iteration



7 **loop:**

...

13 **else if** (update $D(V, Y)$ received from V)

14 $D(A, Y, V) = c(A, V) + D(V, Y);$

15 **if** (there is a new min. for destination Y)

16 **send** $D(A, Y)$ to all neighbors

17 **forever**

Node A

Dest.	Cost	NextHop
B	2	B
C	3	B
D	4	B

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	2	C

Node C

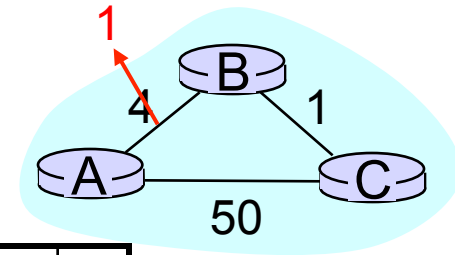
Dest.	Cost	NextHop
A	3	B
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	4	C
B	2	C
C	1	C

Nothing changes

Distance Vector: Link Cost Changes



Node B	D	C	N
A	4	A	
C	1	C	

Node C	D	C	N
A	5	B	
B	1	B	

Node B	D	C	N
A	1	A	
C	1	C	

Node C	D	C	N
A	2	B	
B	1	B	

Node B	D	C	N
A	1	A	
C	1	C	

Node C	D	C	N
A	2	B	
B	1	B	

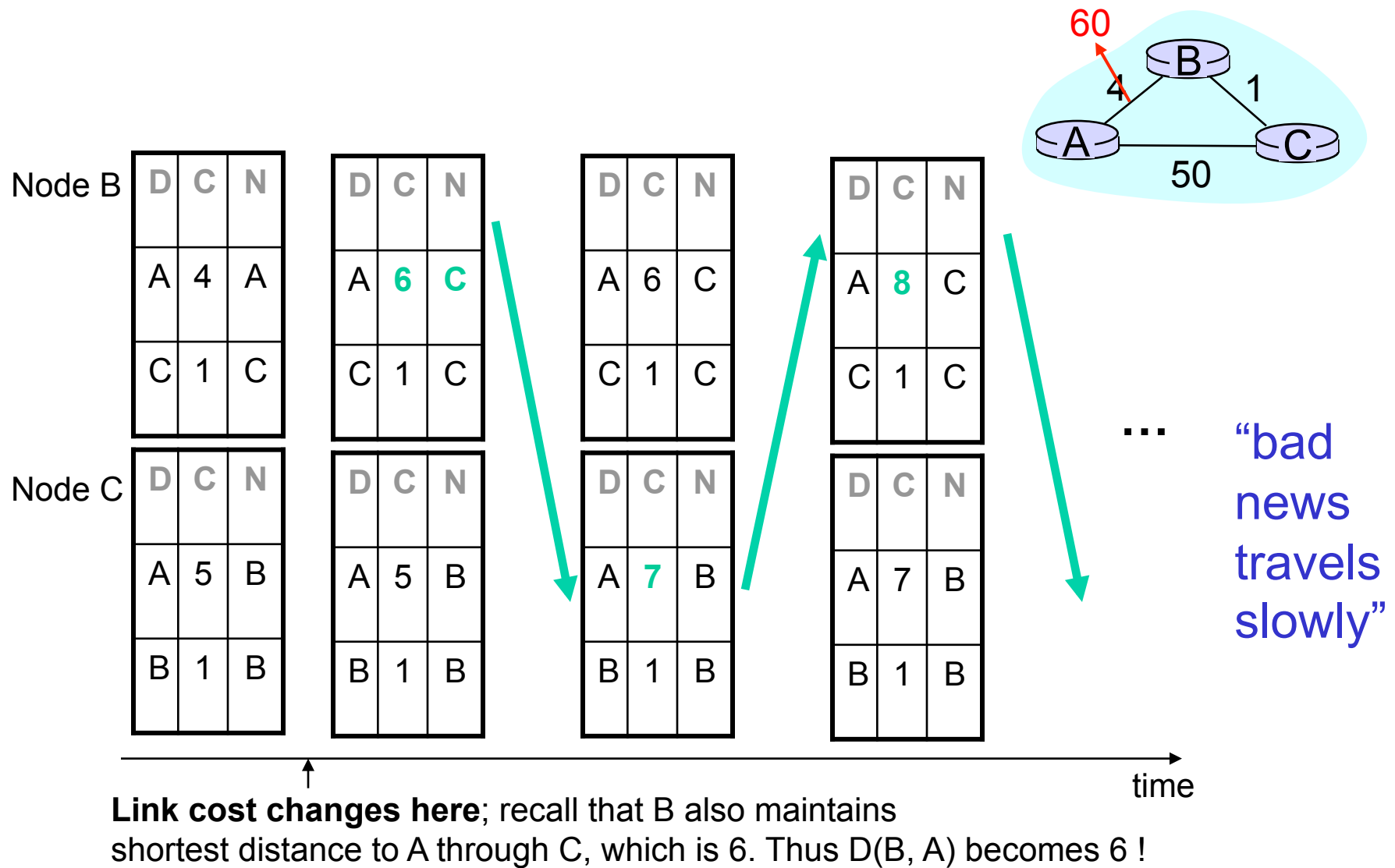
Link cost changes here

Nothing changes

time

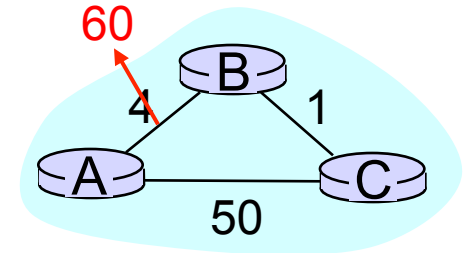
“good news travels fast”

Distance Vector: Count to Infinity Problem



Distance Vector: Poison Reverse

- If C routes through B to get to A:
 - C tells B its distance to A is infinite (so B won't route to A via C)
 - Will this completely solve count to infinity problem?



Node B

D	C	N
A	4	A
C	1	C

D	C	N
A	60	A
C	1	C

D	C	N
A	60	A
C	1	C

D	C	N
A	51	C
C	1	C

D	C	N
A	51	C
C	1	C

Node C

D	C	N
A	5	B
B	1	B

D	C	N
A	5	B
B	1	B

D	C	N
A	50	A
B	1	B

D	C	N
A	50	A
B	1	B

D	C	N
A	50	A
B	1	B

D	C	N
A	50	A
B	1	B

Link State vs. Distance Vector

- Achieve different engineering trade-offs in terms of messaging overhead, computational overhead, convergence time, reliability...
- Which one would you choose and why?

Link State vs. Distance Vector

Per node **message** complexity

- LS: $O(n \cdot d)$ messages to learn entire topology; n – number of nodes; d – average degree of nodes
- DV: $O(d)$ messages per iteration; number of iterations vary (count-to-infinity)

Computation complexity

- LS: $O(n \log n)$ with efficient priority queue for each execution of Dijkstra's alg.
 - Faster incremental algorithms do exist
- DV: $O(n)$ per iteration; total computation and convergence time vary (count-to-infinity)

Robustness

- How likely is it to have a routing loop with each protocol?
- What happens in each protocol if a router malfunctions and lies in messages?