CS4700/CS5700
Fundamentals of Computer Networks

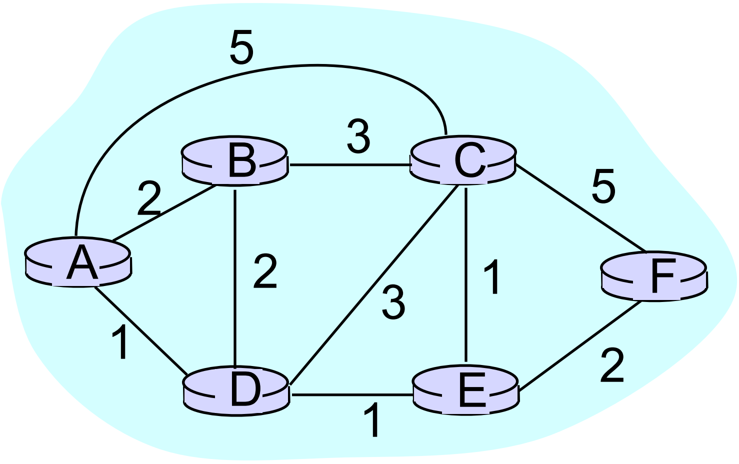
Lecture 11: Intra-domain routing

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

What is Routing?

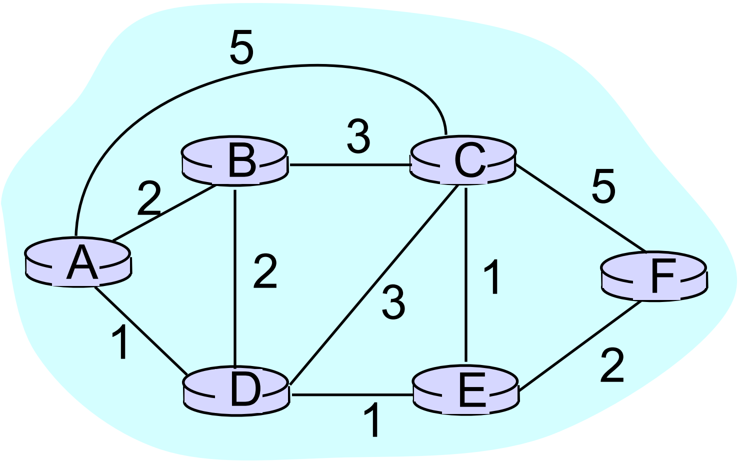
- To ensure information is delivered to the correct destination at a reasonable level of performance
- Forwarding
  - Given a forwarding table, move information from input ports to output ports of a router
  - Local mechanical operations
- Routing
  - Acquires information in the forwarding tables
  - Requires knowledge of the network
  - Requires distributed coordination of routers

Viewing Routing as a Policy



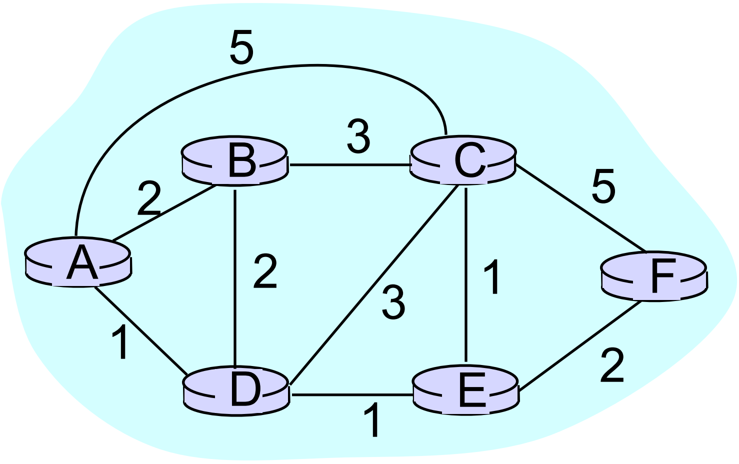
Viewing Routing as a Policy

- Given multiple alternative paths, how to route information to destinations should be viewed as a policy decision



Viewing Routing as a Policy

- Given multiple alternative paths, how to route information to destinations should be viewed as a policy decision
- What are some possible policies?
  - Shortest path (RIP, OSPF)
  - Most load-balanced
  - QoS routing (satisfies app requirements)
  - etc

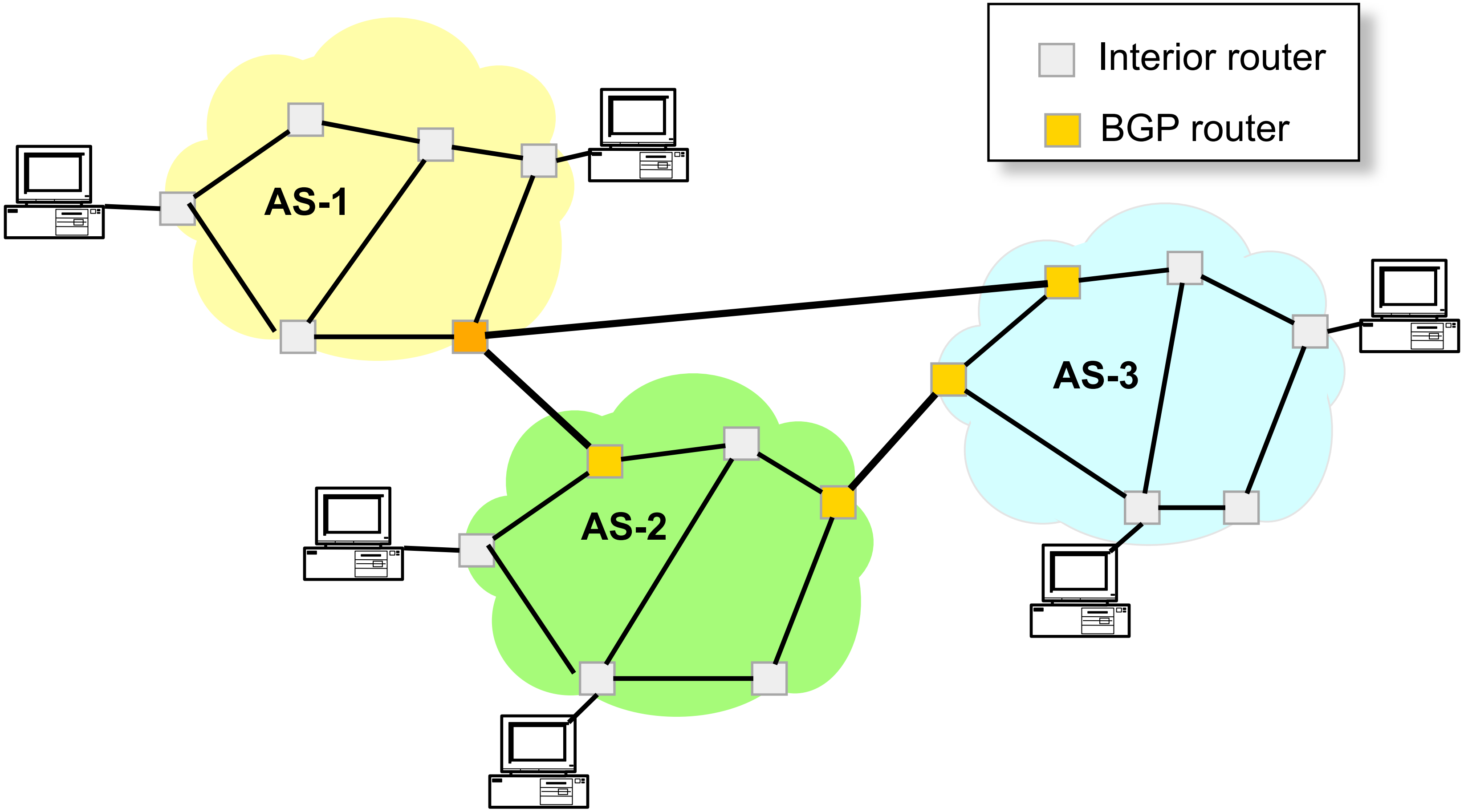


Internet Routing

- Internet topology roughly organized as a two level hierarchy
- First lower level – autonomous systems (AS's)
  - AS: region of network under a single administrative domain
- Each AS runs an intra-domain routing protocol
  - Distance Vector, e.g., Routing Information Protocol (RIP)
  - Link State, e.g., Open Shortest Path First (OSPF)
  - Possibly others

- Second level – inter-connected AS's
- Between AS's runs inter-domain routing protocols, e.g., Border Gateway Routing (BGP)
  - De facto standard today, BGP-4

Example



Why Need the Concept of AS or Domain?

Why Need the Concept of AS or Domain?

- Routing algorithms are not efficient enough to deal with the size of the entire Internet

Why Need the Concept of AS or Domain?

- Routing algorithms are not efficient enough to deal with the size of the entire Internet
- Different organizations may want different internal routing policies

Why Need the Concept of AS or Domain?

- Routing algorithms 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

Why Need the Concept of AS or Domain?

- Routing algorithms 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 organizations (BGP)

Why Need the Concept of AS or Domain?

- Routing algorithms 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 organizations (BGP)

Why Need the Concept of AS or Domain?

- Routing algorithms 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 organizations (BGP)

- Basically, easier to compute routes, more flexibility, more autonomy/independence

Outline

- Two intra-domain routing protocols
- Both try to achieve the “shortest path” routing policy
- Quite commonly used

- OSPF: Based on Link-State routing algorithm
- RIP: Based on Distance-Vector routing algorithm

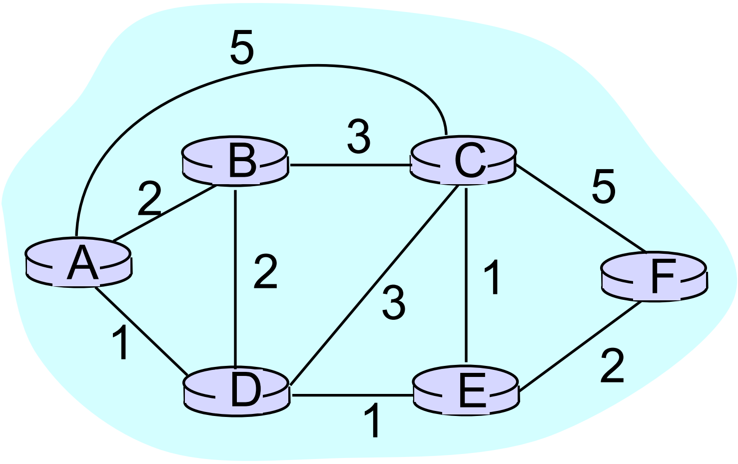
- In Project 2, you will get to implement and play around with these algorithms!
  - Distributed coordination in action

Intra-domain Routing Protocols

- Based on unreliable datagram delivery
- Distance vector
  - Routing Information Protocol (RIP), based on Bellman-Ford algorithm
  - Each neighbor periodically exchange reachability information to its neighbors
  - Minimal communication overhead, but it takes long to converge, i.e., in proportion to the maximum path length
- Link state
  - Open Shortest Path First (OSPF), based on Dijkstra's algorithm
  - Each router periodically floods *immediate* reachability information to other routers
  - Fast convergence, but high communication and computation overhead

Routing on a Graph

- Goal: determine a "good" path through the network from source to destination
  - Good often means the shortest path
- Network modeled as a graph
  - Routers → nodes
  - Link →edges
    - Edge cost: delay, congestion level,…

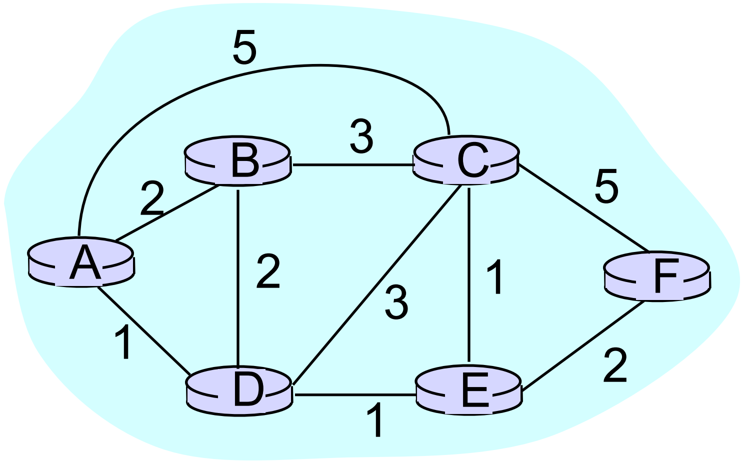


Link State Routing (OSPF): Flooding

- Each node knows its connectivity and cost to a direct neighbor
- Every node tells every other node this local connectivity/cost information
  - Via 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

Flooding Details

- Each node periodically generates Link State Packet (LSP) contains
  - ID of node created LSP
  - List of direct neighbors and costs
  - Sequence number (64 bit, assume to never wrap around)
  - Time to live

Flooding Details

- Each node periodically generates Link State Packet (LSP) contains
  - ID of node created LSP
  - List of direct neighbors and costs
  - Sequence number (64 bit, assume to never wrap around)
  - Time to live
- Flood is reliable
  - Use acknowledgement and retransmission

Flooding Details

- Each node periodically generates Link State Packet (LSP) contains
  - ID of node created LSP
  - List of direct neighbors and costs
  - Sequence number (64 bit, assume to never wrap around)
  - Time to live
- Flood is reliable
  - Use acknowledgement and retransmission
- Sequence number used to identify *newer* LSP
  - An older LSP is discarded
  - What if a router crash and sequence number reset to 0?

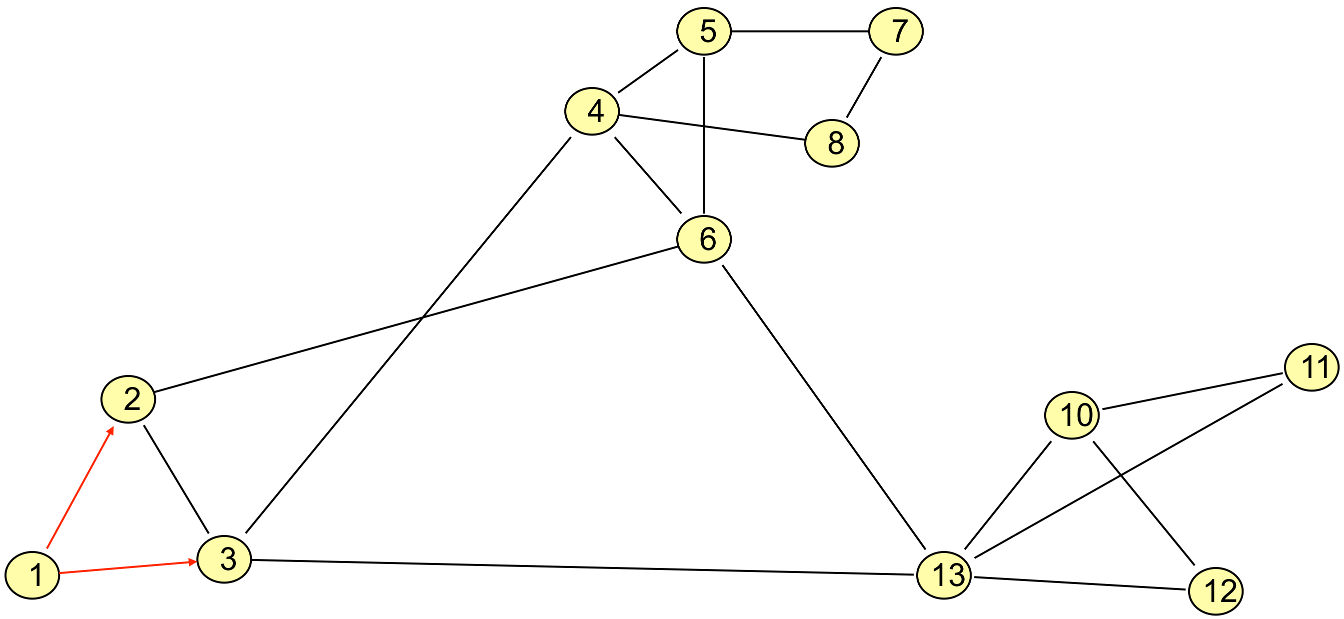
Flooding Details

- Each node periodically generates Link State Packet (LSP) contains
  - ID of node created LSP
  - List of direct neighbors and costs
  - Sequence number (64 bit, assume to never wrap around)
  - Time to live
- Flood is reliable
  - Use acknowledgement and retransmission
- Sequence number used to identify *newer* LSP
  - An older LSP is discarded
  - What if a router crash and sequence number reset to 0?
- Receiving node flood LSP to all its neighbors except the neighbor where the LSP came from

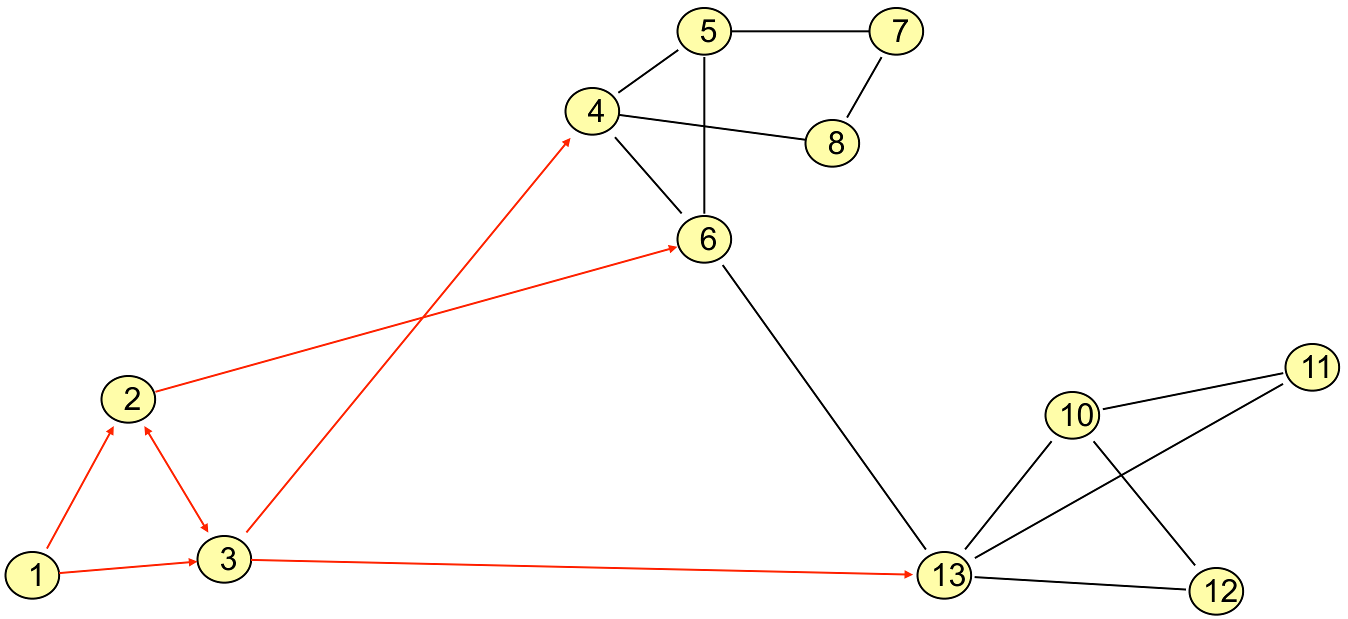
Flooding Details

- Each node periodically generates Link State Packet (LSP) contains
  - ID of node created LSP
  - List of direct neighbors and costs
  - Sequence number (64 bit, assume to never wrap around)
  - Time to live
- Flood is reliable
  - Use acknowledgement and retransmission
- Sequence number used to identify *newer* LSP
  - An older LSP is discarded
  - What if a router crash and sequence number reset to 0?
- Receiving node flood LSP to all its neighbors except the neighbor where the LSP came from
- LSP is also generated when a link's state changes (failed or restored)

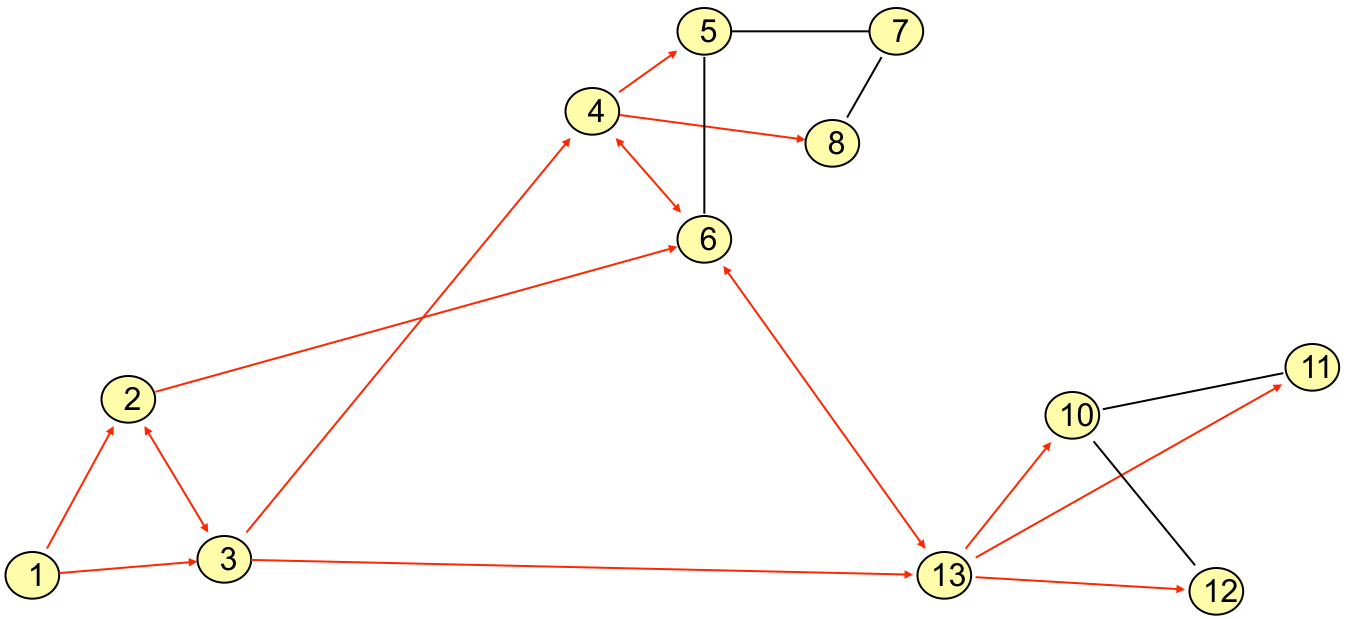
Link State Flooding Example



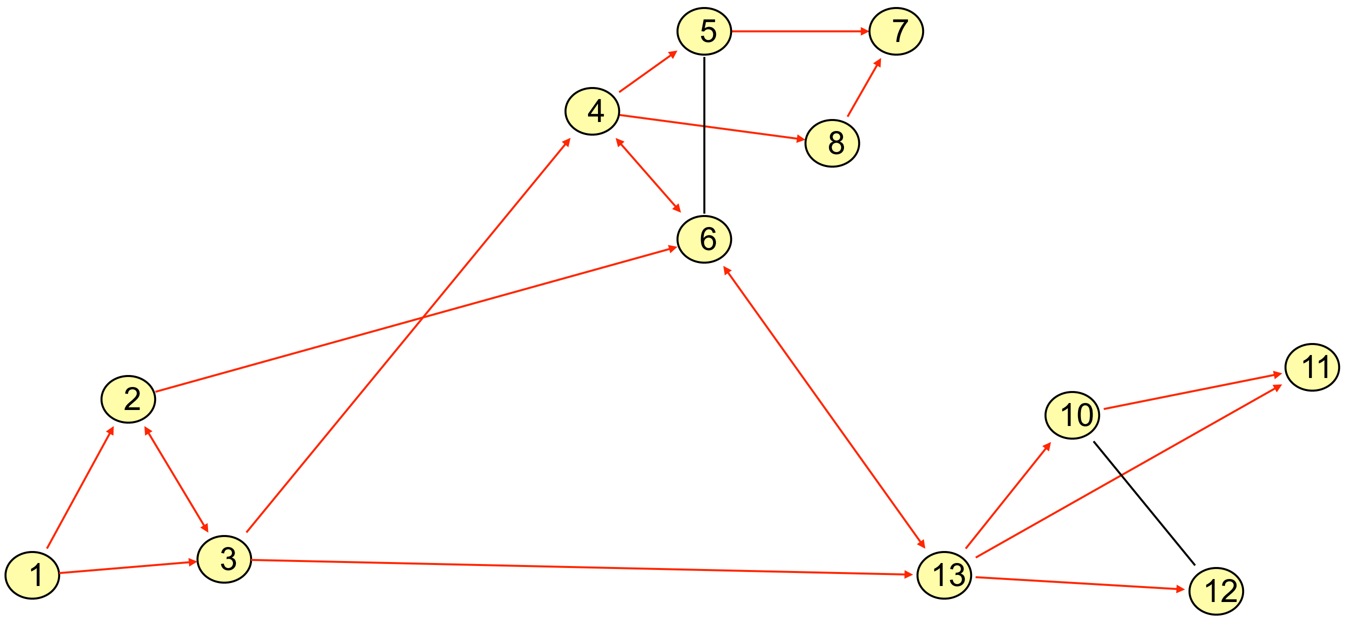
Link State Flooding Example



Link State Flooding Example



Link State Flooding Example



A Link State Routing Algorithm

Dijkstra's algorithm

- Net topology, link costs known to all nodes
  - Accomplished via "link state flooding"
  - All nodes have same info
- Compute least cost paths from one node ('source") to all other nodes
- Repeat for all sources

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

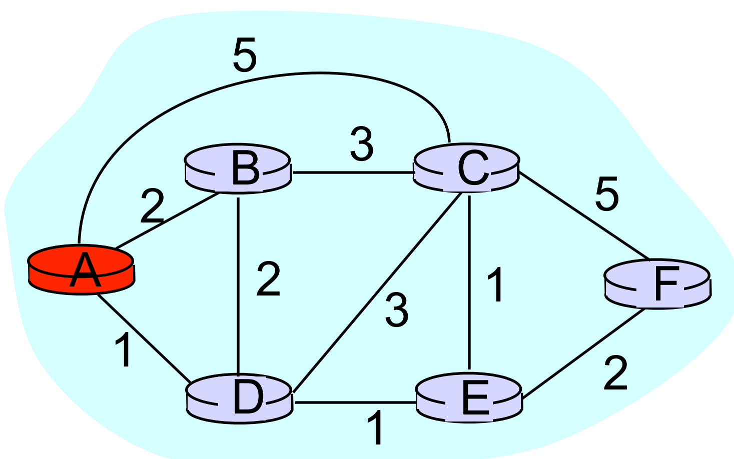
Dijsktra's Algorithm (A "Greedy" Algorithm)

```
Initialization:
  S = {A};
  for all nodes v
    if v adjacent to A
      then D(v) = c(A,v);
      else D(v) = ∞;

 Loop
    find w not in S such that D(w) is a minimum;
    add w to S;
    update D(v) for all v adjacent to w and not in S:
       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
 until all nodes in S;
```

Example: Dijkstra's Algorithm

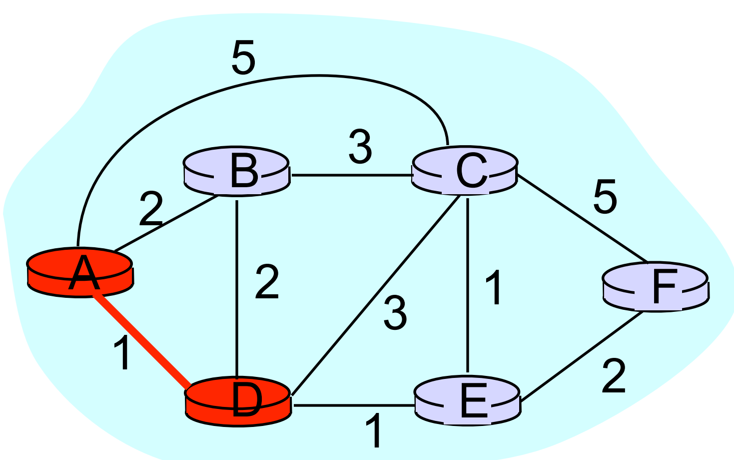
Step	start S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1						
2						
3						
4						
5						




```
Initialization:
  S = {A};
  for all nodes v
    if v adjacent to A
      then D(v) = c(A,v);
      else D(v) = ∞;
…
```

Example: Dijkstra's Algorithm

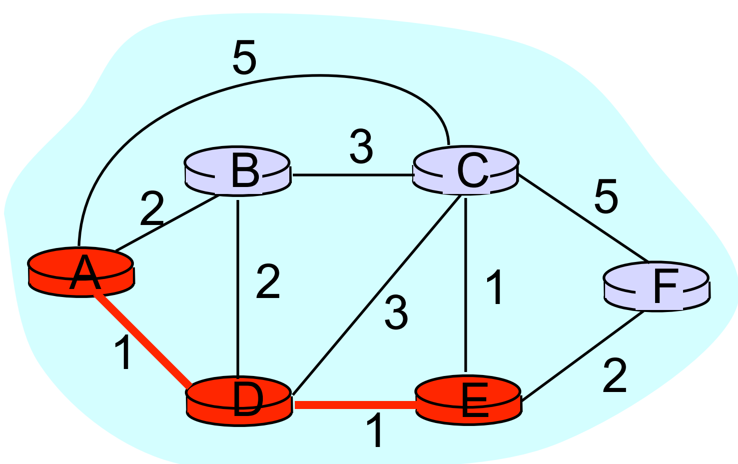
Step	start S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	∞
2						
3						
4						
5						




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

Example: Dijkstra’s Algorithm

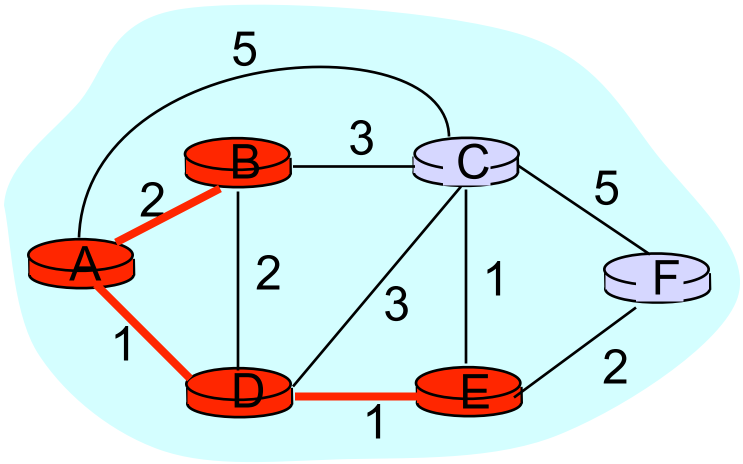
Step	start S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	∞
2	ADE		3,E			4,E
3						
4						
5						




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

Example: Dijkstra's Algorithm

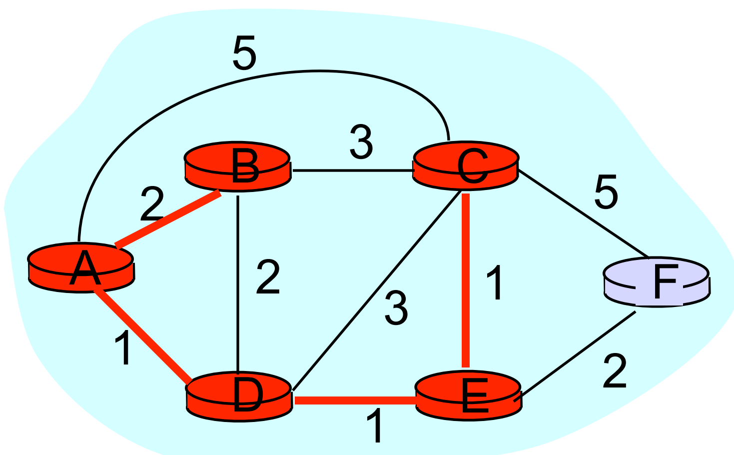
Step	start S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	∞
2	ADE		3,E			4,E
3	ADEB					
4						
5						




```
…
8   Loop
9       find w not in S s.t. D(w) is a minimum;
10      add w to S;
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;
```

Example: Dijkstra's Algorithm

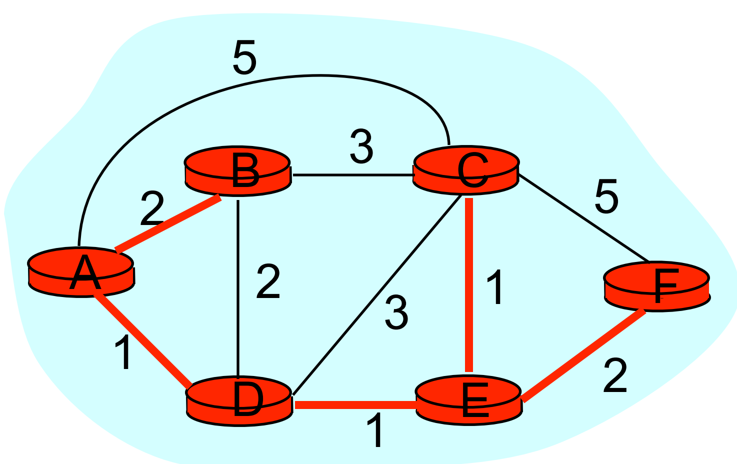
Step	start S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	∞
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5						




```
...
Loop
   find w not in S s.t. D(w) is a minimum;
   add w to S;
  update D(v) for all v adjacent
   to w and not in S:
      D(v) = min( D(v), D(w) + c(w,v) );
   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					




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

Distance Vector Routing (RIP)

- What is a distance vector?
  - Current best known cost to get to a destination
- Idea: Exchange distance vectors among neighbors to learn about lowest cost paths

Node C

Dest.	Cos
A	7
B	1
D	2
E	5
F	1
G	3

Note no vector entry for C itself

At the beginning, distance vector only has information about directly attached neighbors, all other dests have cost ∞

Eventually the vector is filled

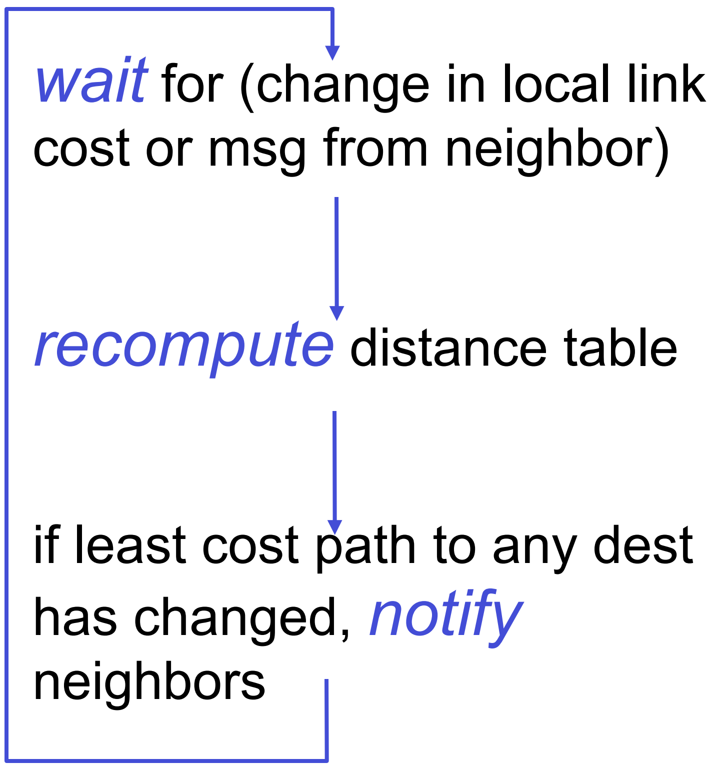
Distance Vector Routing Algorithm

- Iterative: continues until no nodes exchange info
- Asynchronous: nodes need *not* exchange info/iterate in lock steps
- Distributed: each node communicates *only* with directly-attached neighbors
- Each router maintains
  - Row for each possible destination
  - Column for each directly-attached neighbor to node
  - Entry in row Y and column Z of node X ➔ best known distance from X to Y, via Z as next hop
- *Note: for simplicity in this lecture examples we show only the shortest distances to each destination*

Distance Vector Routing

- Each local iteration caused by:
  - Local link cost change
  - Message from neighbor: its least cost path change from neighbor to destination
- Each node notifies neighbors *only* when its least cost path to any destination changes
  - Neighbors then notify their neighbors if necessary

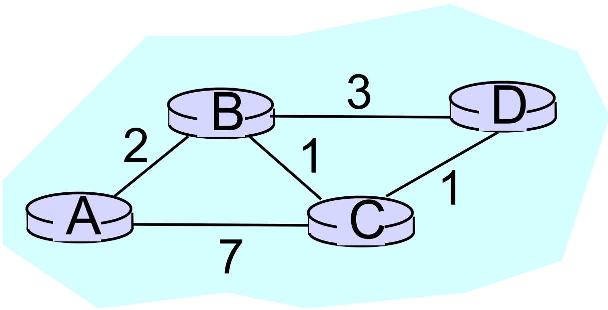
Each node:



Distance Vector Algorithm (cont'd)

```
Initialization:
  for all nodes V  do
    if V adjacent to A
      D(A, V, V) = c(A,V);     /* Distance from A to V via neighbor V */
    else
            D(A, V, *) = ∞;
 loop:
  wait (until A sees a link cost change to neighbor V
          or until A receives update from neighbor V)
  if (c(A,V) changes by d)
    for all destinations Y through V do
      D(A,Y, V) =  D(A,Y,V) + d
  else if (update D(V, Y) received from V)
    /* shortest path from V to some Y has changed  */
    D(A,Y,V) = c(A,V) + D(V, Y);
  if (there is a new minimum for destination Y)
    send D(A, Y) to all neighbors  /* D(A,Y) denotes the min D(A,Y,*) */
  forever
```

Example: Distance Vector Algorithm



Node A

Dest.	Cost	NextHop
B	2	B
C	7	C
D	∞	-

Node B

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

```
Initialization:
  for all nodes V do
    if V adjacent to A
      D(A, V, V) = c(A,V);
    else
      D(A, V, *) = ∞;
…
```

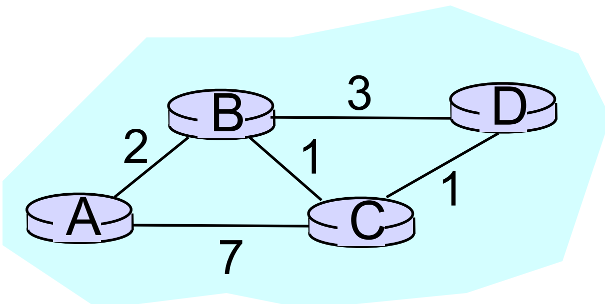
Node C

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

Node D

Dest.	Cost	NextHo
A	∞	-
B	3	B
C	1	C

Example: 1st Iteration (C → A)




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

Node A

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

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

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

Node B

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

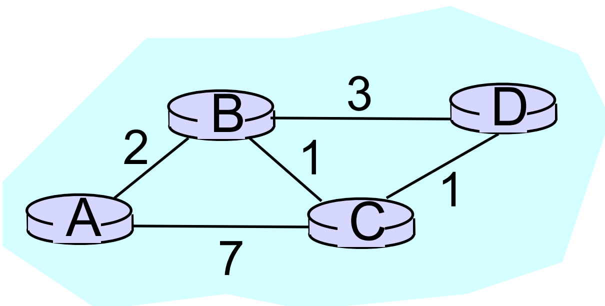
Node C

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

Node D

Dest.	Cost	NextHo
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	NextHo
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

```
        loop:
  …
  else if (update D(V, Y) received from V)
    D(A,Y,V) = c(A,V) + D(V, Y)
  if (there is a new min. for destination Y)
    send D(A, Y) to all neighbors
forever
```

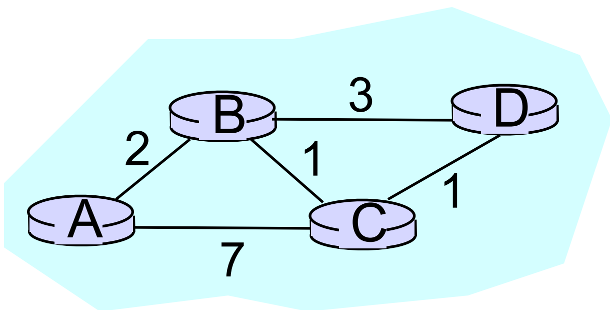
Node C

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

Node D

Dest.	Cost	NextHo
A	∞	-
B	3	B
C	1	C

Example: End of 1st Iteration




```
        loop:
  …
  else if (update D(V, Y) received from V)
    D(A,Y,V) = c(A,V) + D(V, Y);
  if (there is a new min. for destination Y)
    send D(A, Y) to all neighbors
 forever
```

Node A

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

Node B

Dest.	Cost	NextHo
A	2	A
C	1	C
D	**2**	**C**

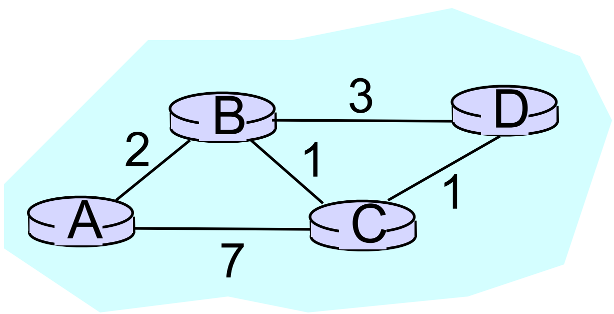
Node C

Dest.	Cost	NextHo
A	**3**	**B**
B	1	B
D	1	D

Node D

Dest.	Cost	NextHo
A	**5**	**B**
B	**2**	**C**
C	1	C

Example: End of 2nd Iteration




```
loop:
  …
  else if (update D(V, Y) received from V)
    D(A,Y,V) = c(A,V) + D(V, Y);
  if (there is a new min. for destination Y)
    send D(A, Y) to all neighbors
forever
```

Node A

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

Node B

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

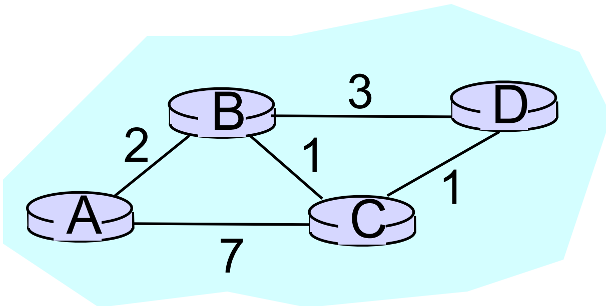
Node C

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

Node D

Dest.	Cost	NextHo
A	**4**	**C**
B	2	C
C	1	C

Example: End of 3rd Iteration



loop:

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

Node A

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

Node B

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

Node C

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

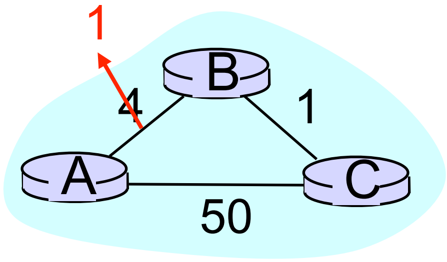
Node D

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

Nothing changes → algorithm terminates

Distance Vector: Link Cost Changes

```
loop:
  wait (until A sees a link cost change to neighbor V
           or until A receives update from neighbor V)
  if (c(A,V) changes by d)
    for all destinations Y through V do
       D(A,Y,V) =  D(A,Y,V) + d
  else if (update D(V, Y) received from V)
    D(A,Y,V) = c(A,V) + D(V, Y);
  if (there is a new minimum for destination Y)
    send D(A, Y) to all neighbors
forever
```



Node B

D	C	N
A	4	A
C	1	B

D	C	N
A	1	A
C	1	B

D	C	N
A	1	A
C	1	B

D	C	N
A	1	A
C	1	B

Node C

D	C	N
A	5	B
B	1	B

D	C	N
A	5	B
B	1	B

D	C	N
A	2	B
B	1	B

D	C	N
A	2	B
B	1	B

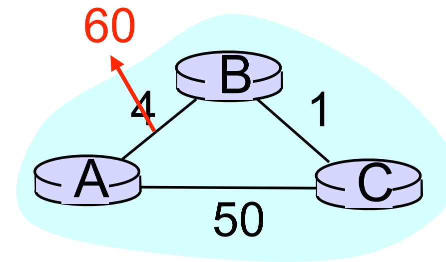
“good news travels fast”

time

Link cost changes here **Algorithm terminates**

Distance Vector: Count to Infinity Problem

```
loop:
 wait (until A sees a link cost change to neighbor V
         or until A receives update from neighbor V)
  if (c(A,V) changes by d)
    for all destinations Y through V do
      D(A,Y,V) =  D(A,Y,V) + d ;
  else if (update D(V, Y) received from V)
    D(A,Y,V) = c(A,V) + D(V, Y);
  if (there is a new minimum for destination Y)
    send D(A, Y) to all neighbors
 forever
```



Node B

D	C	N
A	4	A
C	1	B

D	C	N
A	6	C
C	1	B

D	C	N
A	6	C
C	1	B

D	C	N
A	8	C
C	1	B

Node C

D	C	N
A	5	B
B	1	B

D	C	N
A	5	B
B	1	B

D	C	N
A	7	B
B	1	B

D	C	N
A	7	B
B	1	B

…

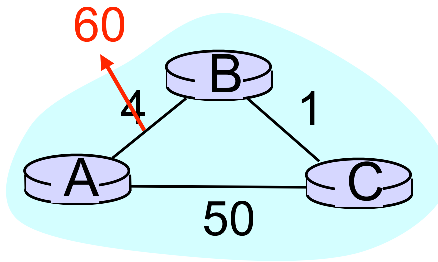
time

"bad news travels slowly"

Link cost changes here; recall that B also maintains shortest distance to A through C, which is 6. Thus D(B, A) becomes 6 !

Distance Vector: Poisoned Reverse

- If C routes through B to get to A:
  - C tells B its (C’s) 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	B

D	C	N
A	60	A
C	1	B

D	C	N
A	60	A
C	1	B

D	C	N
A	51	C
C	1	B

D	C	N
A	51	C
C	1	B

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

time

Link cost changes here; B updates D(B, A) = 60 as C has advertised D(C, A) = ∞

Algorithm terminates

Link State vs. Distance Vector

Per node message complexity

- LS: O(n*d) messages; n – number of nodes; d – degree of node
- DV: O(d) messages; where d is node's degree

Complexity

- LS: $O(n^2)$ with O(n*d) messages (with naïve priority queue)
- 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:
  - node can advertise incorrect *path* cost
  - each node's table used by others; error propagate through network