Routing Algorithm Classification

Global or decentralized information?

Global:

- All routers have complete topology, link cost info
- "Link state" algorithms

Decentralized:

- Router knows physically-connected neighbors, link costs to neighbors
- Iterative process of computation, exchange of info with neighbors
- "Distance vector" algorithms

Static or dynamic?

Static:

- Routes change slowly over time

Dynamic:

- Routes change more quickly
  - Periodic update
  - In response to link cost changes

A Distance Vector Routing Algorithm

Decentralized algorithm:

- Router knows its neighbors and link costs to neighbors
- Iterative computation, exchange of info with neighbors

Bellman-Ford Equation (dynamic programming)

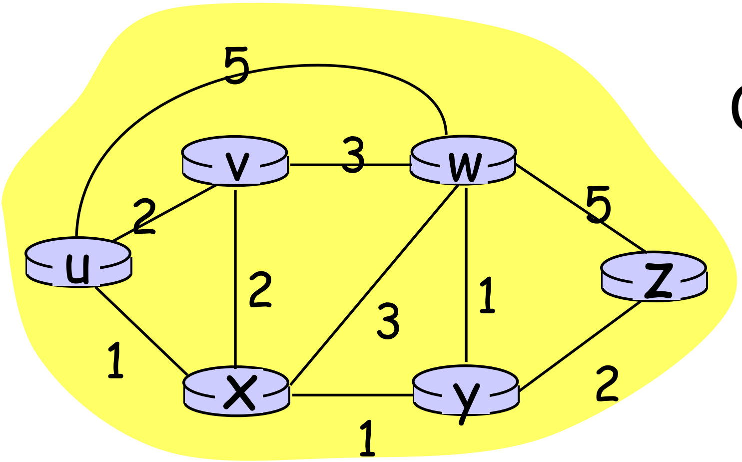
Define $d_x(y)$:= cost of least-cost path from x to y

Then

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

where min is taken over all neighbors v of x

Bellman-Ford Example



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

Bellman-Ford equation says:

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

Node that yields minimum is next hop in shortest path ➜ forwarding table

Distance Vector Algorithm

Iterative, asynchronous:

- Each local iteration caused by:
  - Local link cost change
  - DV update message from neighbor

Distributed:

- Each node notifies neighbors *only* when its Distance Vector changes
  - Neighbors then notify their neighbors if necessary

Each node:

wait for (change in local link cost of msg from neighbor)

↓

recompute estimates

↓

if Distance Vector to any dest has changed, *notify* neighbors

$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$
$= \min\{2+0 , 7+1\} = 2$

$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$
$= \min\{2+1 , 7+0\} = 3$

node x table

from \ cost to	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

node y table

from \ cost to	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

from \ cost to	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

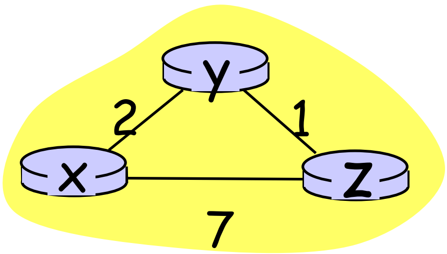
node z table

from \ cost to	x	y	z
x	∞	∞	
y	∞	∞	∞
z	7	1	0

from \ cost to	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

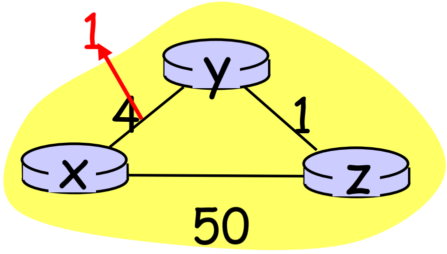
time



Distance Vector (DV): Link Cost Changes

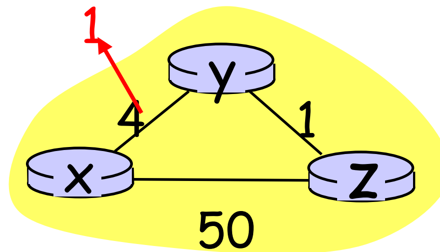
Link cost changes:

- Node detects local link cost change
- Updates routing info, recalculates distance vector
- If DV changes, notify neighbors



"good news travels fast"

- At time t_0, *y* detects link-cost change, updates its DV, and informs its neighbors.
- At time t_1, *z* receives update from *y* and updates its table, computes a new least cost to *x* and sends its neighbors its DV
- At time t_2, *y* receives *z*'s update and updates its distance table. As *y*'s least costs do not change *y* does not send updates to *z*.



node y table

cost to

from	x	y	z
x			
y	~~4~~ 1	0	1
z	5	1	0

cost to

from	x	y	z
x			
y	1	0	1
z	5	1	0

cost to

from	x	y	z
x			
y	1	0	1
z	2	1	0

node z table

cost to

from	x	y	z
x			
y	4	0	1
z	5	1	0

cost to

from	x	y	z
x			
y	1	0	1
z	~~5~~ 2	1	0

cost to

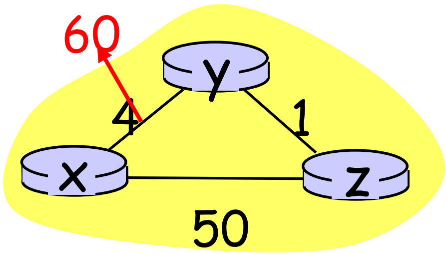
from	x	y	z
x			
y	1	0	1
z	2	1	0

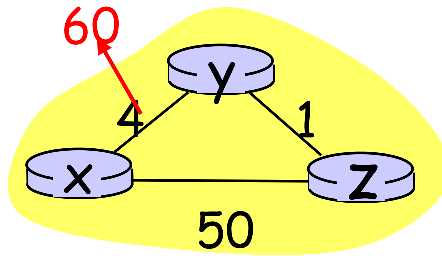
time

Distance Vector: Link Cost Changes

Link cost changes:

- Good news travels fast
- Bad news travels slow





$D_y(x) = \min\{c(y,x) + D_x(x), c(y,z) + D_z(x)\}$
$= \min\{60 + 0 , 1 + 5\} = 6$

$D_y(x) = \min\{c(y,x) + D_x(x),$
$c(y,z) + D_z(x)\}$
$= \min\{60 + 0 , 1 + 7\} = 8$

node y table

from \ cost to	x	y	z
x			
y	~~4~~ 6	0	1
z	5	1	0

from \ cost to	x	y	z
x			
y	6	0	1
z	5	1	0

from \ cost to	x	y	z
x			
y	~~6~~ 8	0	1
z	7	1	0

node z table

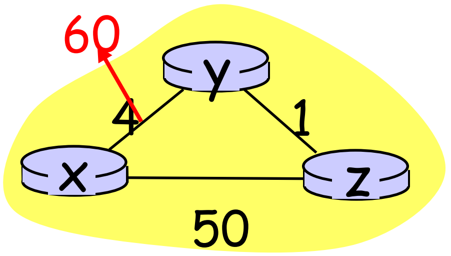
from \ cost to	x	y	z
x			
y	4	0	1
z	5	1	0

from \ cost to	x	y	z
x			
y	6	0	1
z	~~5~~ 7	1	0

from \ cost to	x	y	z
x			
y	6	0	1
z	7	1	0

time

Distance Vector: Link Cost Changes



Link cost changes:

- Good news travels fast
- Bad news travels slow – "count to infinity" problem!
- E.g., 44 iterations before algorithm stabilizes

Poissoned reverse:

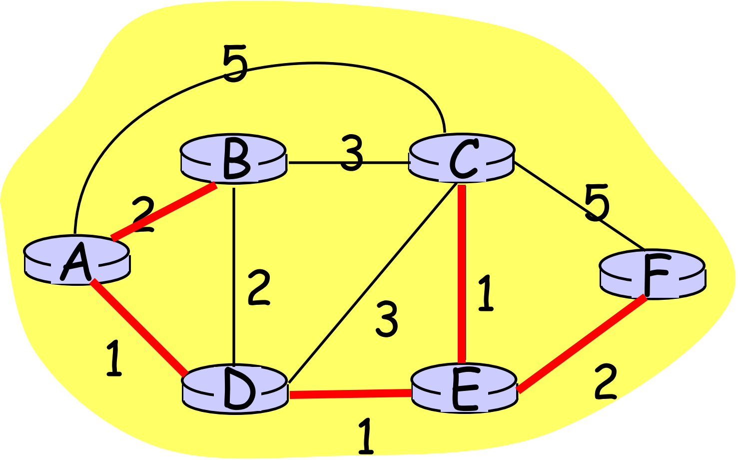
- If *Z* routes through *Y* to get to *X*:
  - *Z* tells *Y* its (*Z*'s) distance to *X* is infinite (so *Y* won't route to *X* via *Z*)
- Will this completely solve count to infinity problem?

A Link-State Routing 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 routing table for that node
- Example:
  Dijkstra's algorithm
  - Iterative: after k iterations, know least cost path to k dst.'s

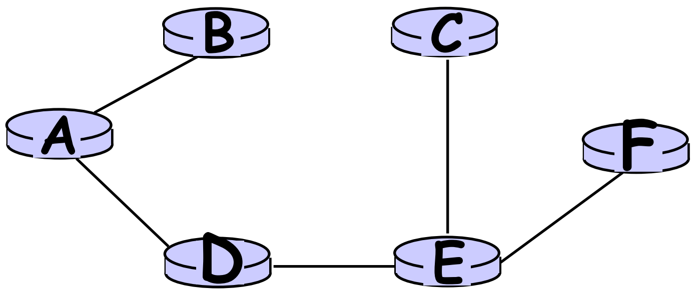
Dijkstra's Algorithm: Example

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	infinity	infinity
1	AD	2,A	4,D		2,D	infinity
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E
4	ADEBC					4,E
5	ADEBCF					



Dijkstra’s Algorithm: Example (2)

Resulting shortest-path tree from A:



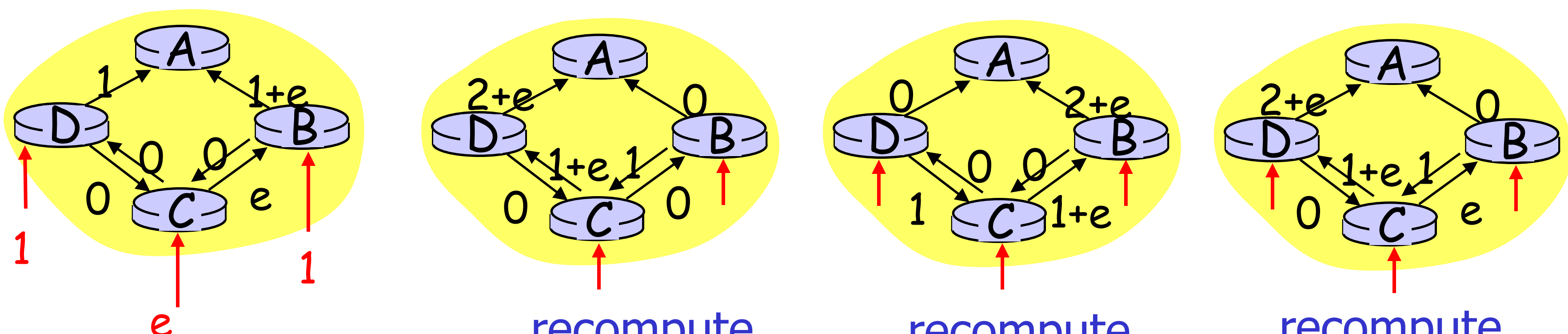
Resulting forwarding table at A:

destination	link
B	(A,B)
D	(A,D)
E	(A,D)
C	(A,D)
F	(A,D)

Dijkstra's Algorithm: Discussion

Oscillations possible:

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



initially

… recompute routing

… recompute

… recompute

Comparison of LS and DV Algorithms

Speed of Convergence

- LS: O(n log n) 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