

# Chapter 5

## Network Layer:

### The Control Plane

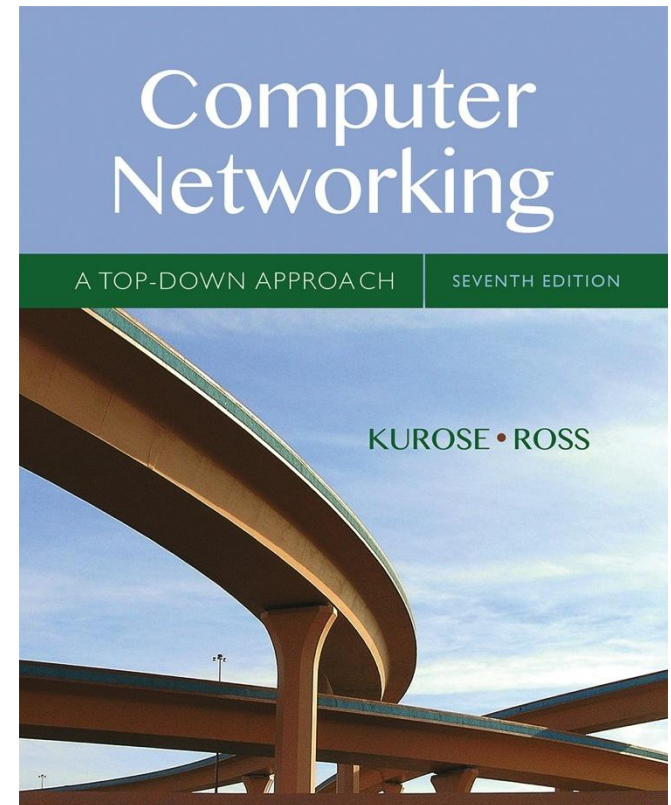
#### A note on the use of these Powerpoint slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

© All material copyright 1996-2016  
J.F Kurose and K.W. Ross, All Rights Reserved



## *Computer Networking: A Top Down Approach*

7<sup>th</sup> edition

Jim Kurose, Keith Ross

Pearson/Addison Wesley

April 2016

# Chapter 5: network layer control plane

*chapter goals:* understand principles behind network control plane

- traditional routing algorithms
- SDN controllers
- Internet Control Message Protocol
- network management

and their instantiation, implementation in the Internet:

- OSPF, BGP, OpenFlow, ODL and ONOS controllers, ICMP, SNMP

# Chapter 5: outline

5.1 introduction

5.2 routing protocols

- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs: BGP

5.5 The SDN control plane

5.6 ICMP: The Internet Control Message Protocol

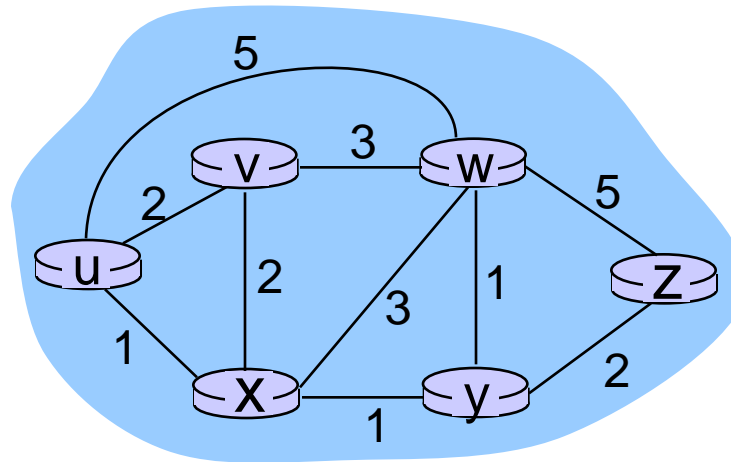
5.7 Network management and SNMP

# Routing protocols

*Routing protocol goal:* determine “good” paths (equivalently, routes), from sending hosts to receiving host, through network of routers

- path: sequence of routers packets will traverse in going from given initial source host to given final destination host
- “good”: least “cost”, “fastest”, “least congested”
- routing: a “top-10” networking challenge!

# Graph abstraction of the network



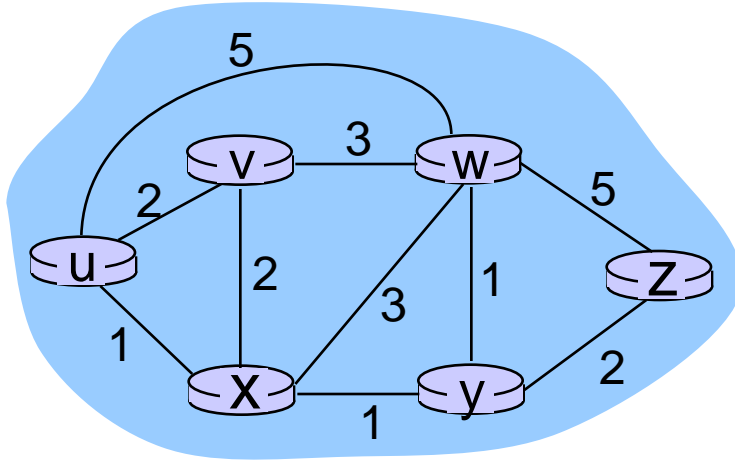
graph:  $G = (N, E)$

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

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

*aside:* graph abstraction is useful in other network contexts, e.g., P2P, where  $N$  is set of peers and  $E$  is set of TCP connections

# Graph abstraction: costs



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

cost could always be 1, or  
inversely related to bandwidth,  
or inversely related to  
congestion

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

# Routing algorithm classification

*Q: 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

*Q: static or dynamic?*

*static:*

- routes change slowly over time

*dynamic:*

- routes change more quickly
  - periodic update
  - in response to link cost changes

# Chapter 5: outline

5.1 introduction

5.2 routing protocols

- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs: BGP

5.5 The SDN control plane

5.6 ICMP: The Internet Control Message Protocol

5.7 Network management and SNMP



# 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  $v$  \*/

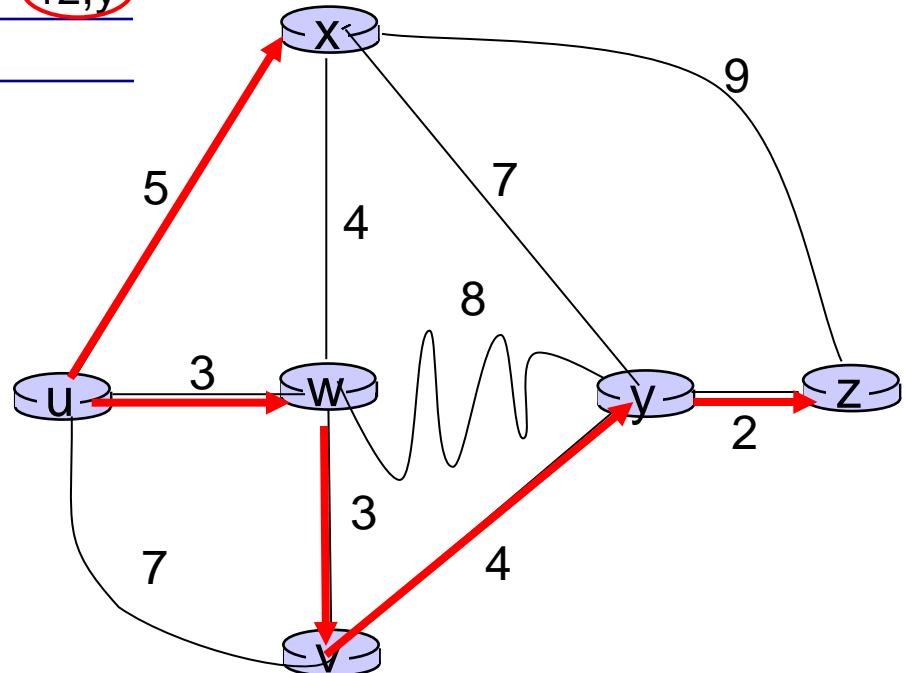
15 **until all nodes in  $N'$**

# Dijkstra's algorithm: example

Step	N'	D( <b>v</b> ) p(v)	D( <b>w</b> ) p(w)	D( <b>x</b> ) p(x)	D( <b>y</b> ) p(y)	D( <b>z</b> ) p(z)
0	u	7,u	<b>3,u</b>	5,u	$\infty$	$\infty$
1	uw	6,w		<b>5,u</b>	11,w	$\infty$
2	uwx	<b>6,w</b>			11,w	14,x
3	uwxv				<b>10,v</b>	14,x
4	uwxvy					<b>12,y</b>
5	uwxvyz					

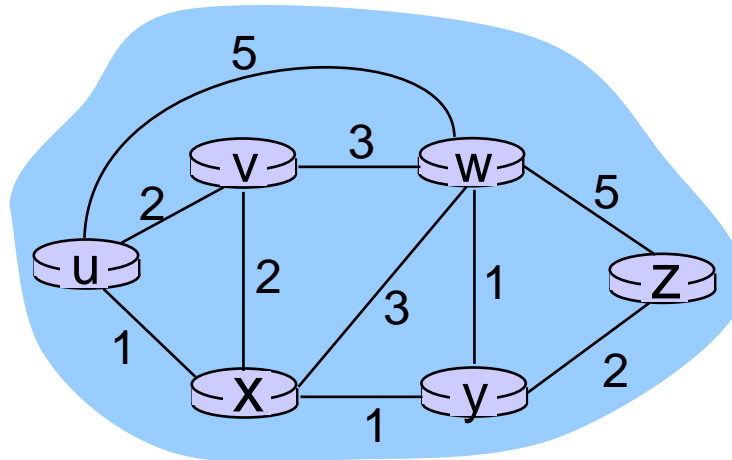
## notes:

- ❖ construct shortest path tree by tracing predecessor nodes
- ❖ ties can exist (can be broken arbitrarily)



# Dijkstra's algorithm: another example

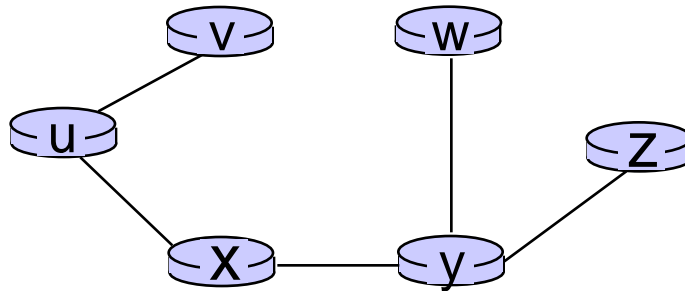
Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	$\infty$	$\infty$
1	ux	2,u	4,x		2,x	$\infty$
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



\* Check out the online interactive exercises for more examples: [http://gaia.cs.umass.edu/kurose\\_ross/interactive/](http://gaia.cs.umass.edu/kurose_ross/interactive/)

# Dijkstra's algorithm: example (2)

resulting shortest-path tree from u:



resulting forwarding table in u:

destination	link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

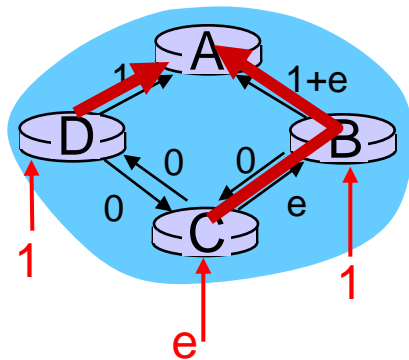
# Dijkstra's algorithm, discussion

*algorithm complexity:* n nodes

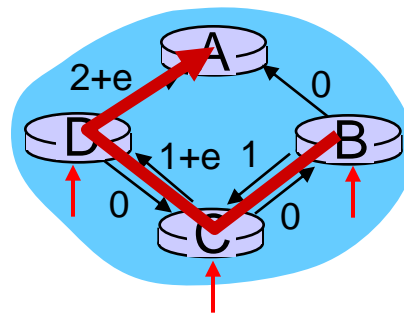
- each iteration: need to check all nodes, w, not in N
- $n(n+1)/2$  comparisons:  $O(n^2)$
- more efficient implementations possible:  $O(n \log n)$

*oscillations possible:*

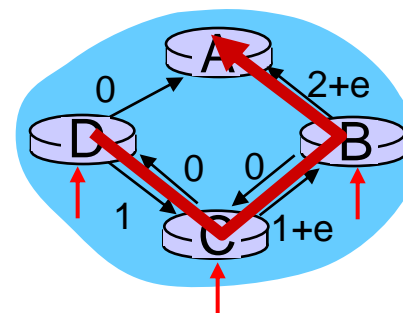
- e.g., support link cost equals amount of carried traffic:



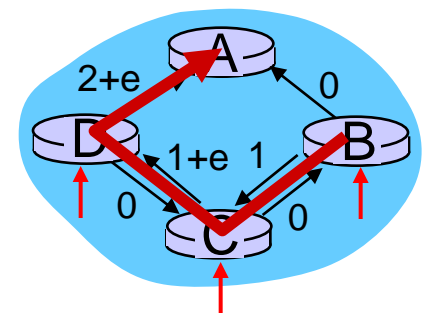
initially



given these costs,  
find new routing....  
resulting in new costs



given these costs,  
find new routing....  
resulting in new costs



given these costs,  
find new routing....  
resulting in new costs

# Chapter 5: outline

5.1 introduction

5.2 routing protocols

- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs: BGP

5.5 The SDN control plane

5.6 ICMP: The Internet Control Message Protocol

5.7 Network management and SNMP

# Distance vector algorithm

*Bellman-Ford equation (dynamic programming)*

let

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

then

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

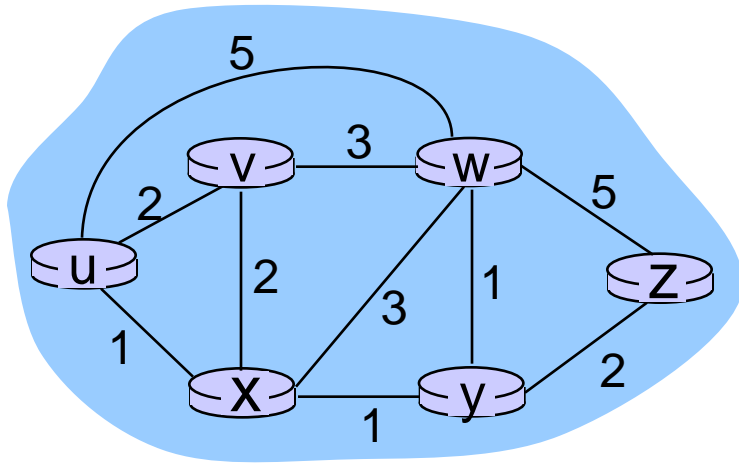
cost from neighbor  $v$  to destination  $y$

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

# Distance vector algorithm

- $D_x(y)$  = estimate of least cost from  $x$  to  $y$ 
  - $x$  maintains distance vector  $\mathbf{D}_x = [D_x(y): y \in N]$
- node  $x$ :
  - knows cost to each neighbor  $v$ :  $c(x,v)$
  - maintains its neighbors' distance vectors. For each neighbor  $v$ ,  $x$  maintains  $\mathbf{D}_v = [D_v(y): y \in N]$

# Distance vector algorithm

## *key idea:*

- from time-to-time, each node sends its own distance vector estimate to neighbors
- when  $x$  receives new DV estimate from neighbor, it updates its own DV using B-F equation:

$$D_x(y) \leftarrow \min_v \{c(x, v) + D_v(y)\} \text{ for each node } y \in N$$

- ❖ under minor, natural conditions, the estimate  $D_x(y)$  converge to the actual least cost  $d_x(y)$

# Distance vector algorithm

## *iterative, asynchronous:*

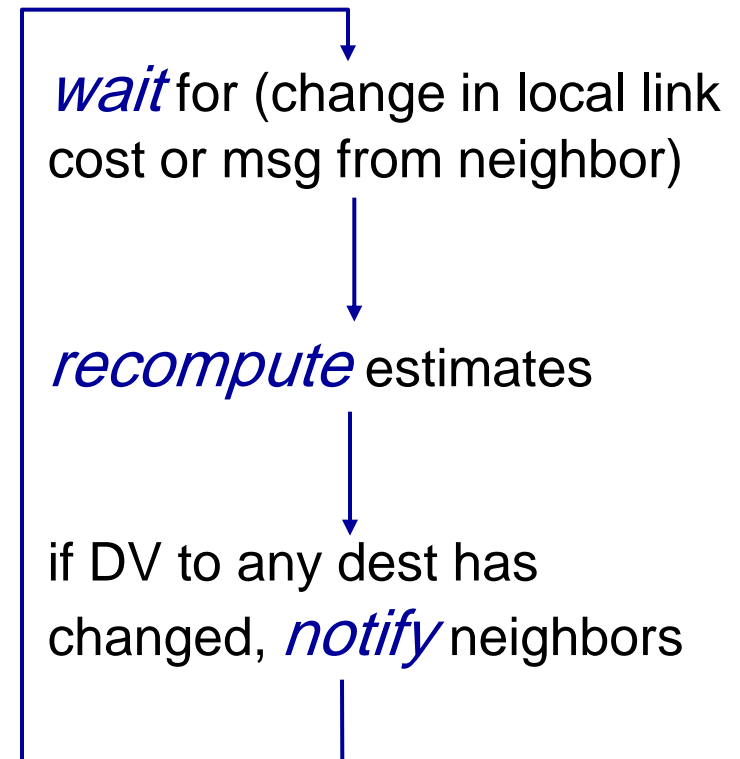
each local iteration  
caused by:

- local link cost change
- DV update message from neighbor

## *distributed:*

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

## *each node:*



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

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

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

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

**node x  
table**

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

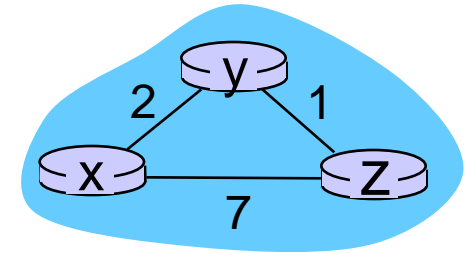
		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	7	1	0

**node y  
table**

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

**node z  
table**

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



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**

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

**node y  
table**

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

**node z  
table**

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

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

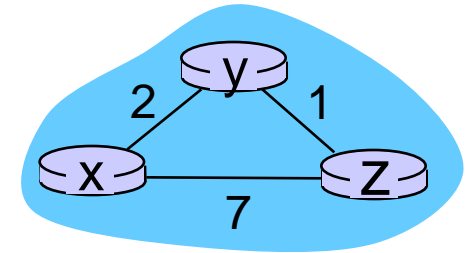
	cost to		
	x	y	z
from x	0	2	7
from y	2	0	1
from z	7	1	0

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

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

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

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

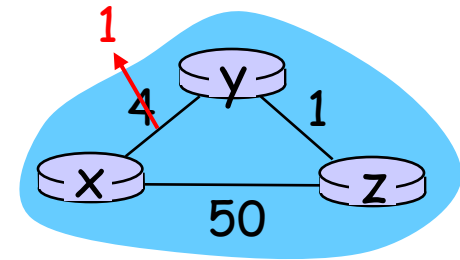


time →

# Distance vector: 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”

$t_0$ :  $y$  detects link-cost change, updates its DV, informs its neighbors.

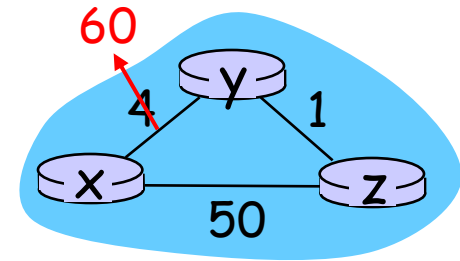
$t_1$ :  $z$  receives update from  $y$ , updates its table, computes new least cost to  $x$ , sends its neighbors its DV.

$t_2$ :  $y$  receives  $z$ 's update, updates its distance table.  $y$ 's least costs do *not* change, so  $y$  does *not* send a message to  $z$ .

# Distance vector: link cost changes

## *link cost changes:*

- ❖ node detects local link cost change
- ❖ *bad news travels slow* - “count to infinity” problem!
- ❖ 44 iterations before algorithm stabilizes: see text



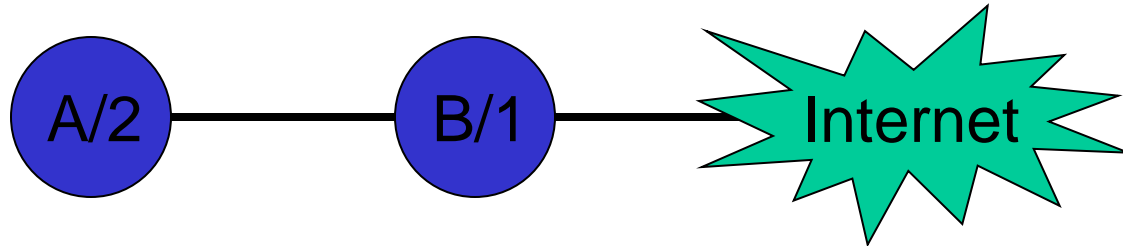
## *poisoned reverse:*

- ❖ 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?



# Count To Infinity Problem

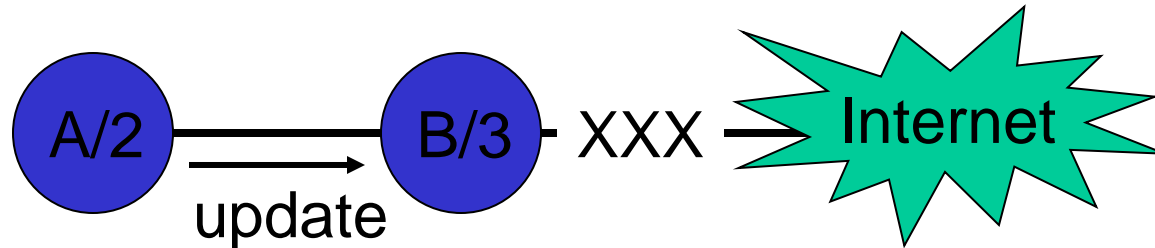
- Simple example
  - Costs in nodes are to reach Internet



- Now link between B and Internet fails ...

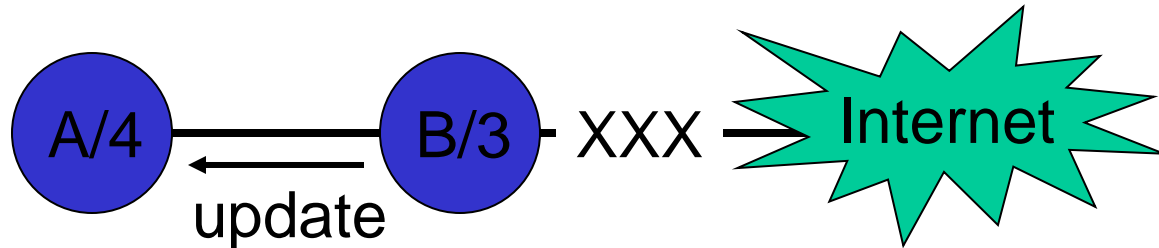
# Count To Infinity Problem

- B hears of a route to the Internet via A with cost 2
- So B switches to the “better” (but wrong!) route



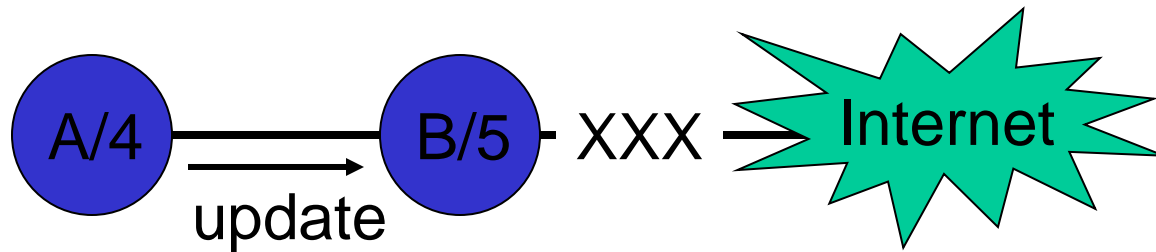
# Count To Infinity Problem

- A hears from B and increases its cost



# Count To Infinity Problem

- B hears from A and (surprise) increases its cost
- Cycle continues and we “count to infinity”



- Packets caught in the crossfire loop between A and B

# Split Horizon

- Solves trivial count-to-infinity problem
- Router never advertises the cost of a destination back to its next hop – that's where it learned it from!
- Poison reverse: go even further – advertise back infinity
- However, DV protocols still subject to the same problem with more complicated topologies
  - Many enhancements suggested

# 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