

# SC250

# Computer Networking I

## Network Layer Principles

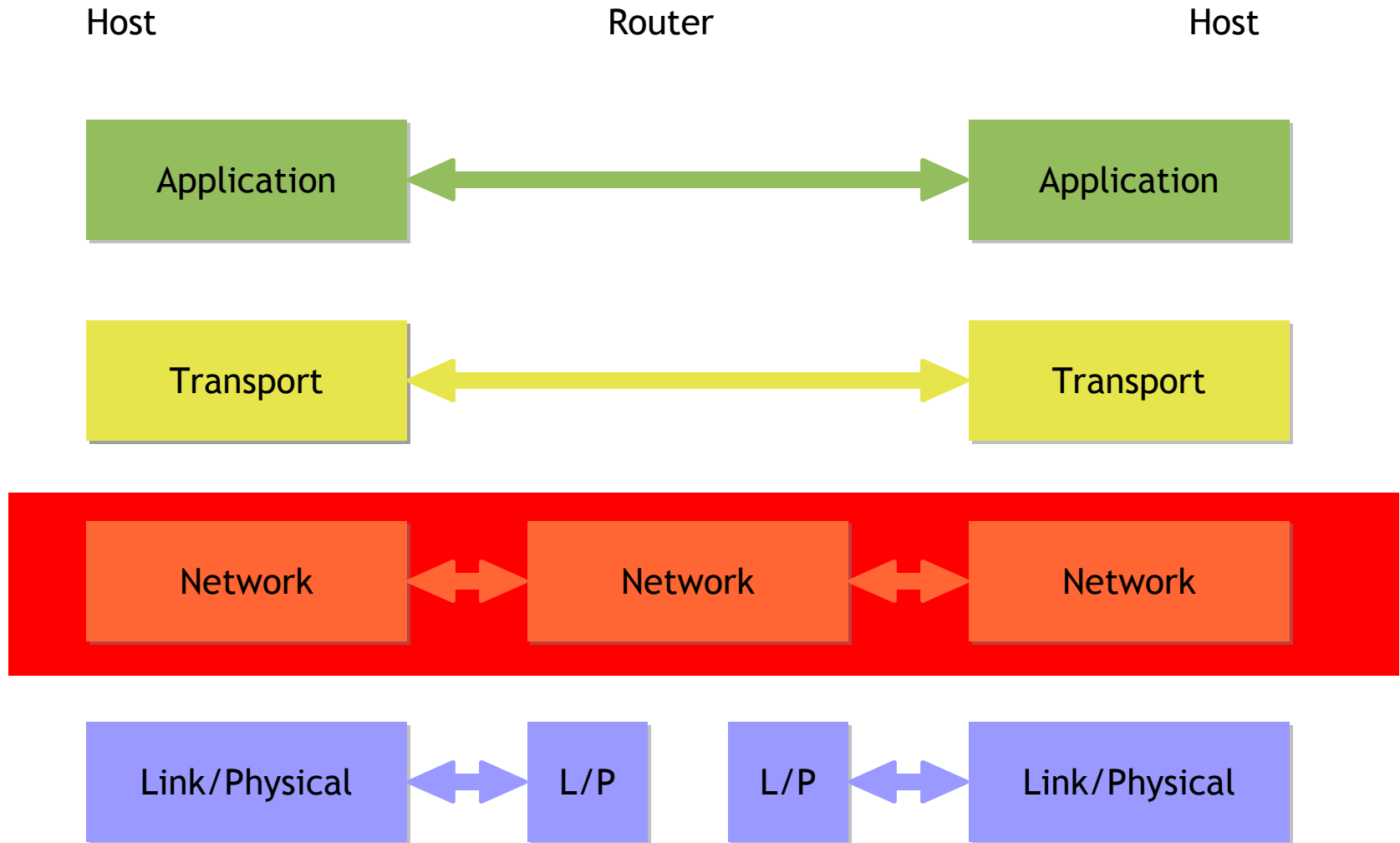
Prof. Matthias Grossglauser

School of Computer and Communication Sciences  
EPFL

<http://lcawww.epfl.ch>



# Network Layer



# Objectives

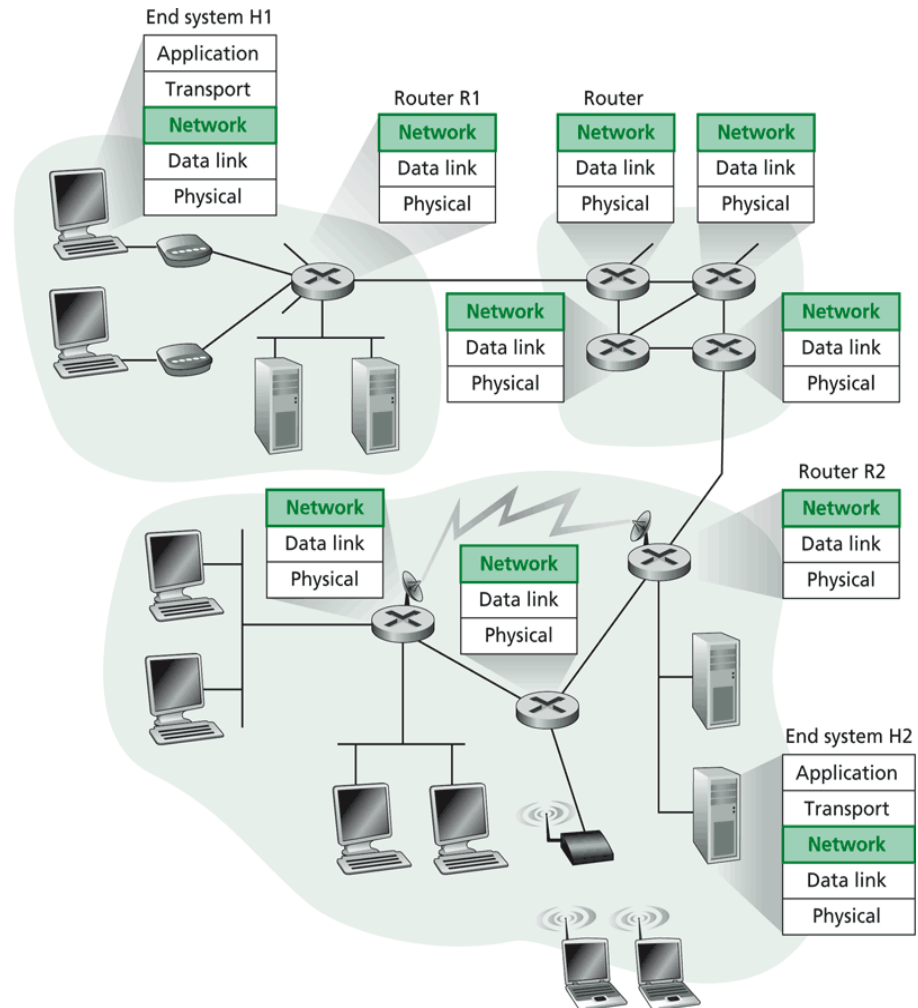
- Understand principles behind network layer services:
  - Routing (path selection)
  - Dealing with scale: routing hierarchy and autonomous systems (AS)
- Instantiation and implementation in the Internet
  - Next week
- Overview:
  - Today:
    - network layer services
    - routing principles:
      - path selection
      - hierarchical routing
  - Next week:
    - IP protocol
    - Internet routing protocols
      - intra-domain
      - inter-domain

# Network Layer Functions

- Transport packet from sending to receiving hosts
- Network layer protocols in *every* host, router

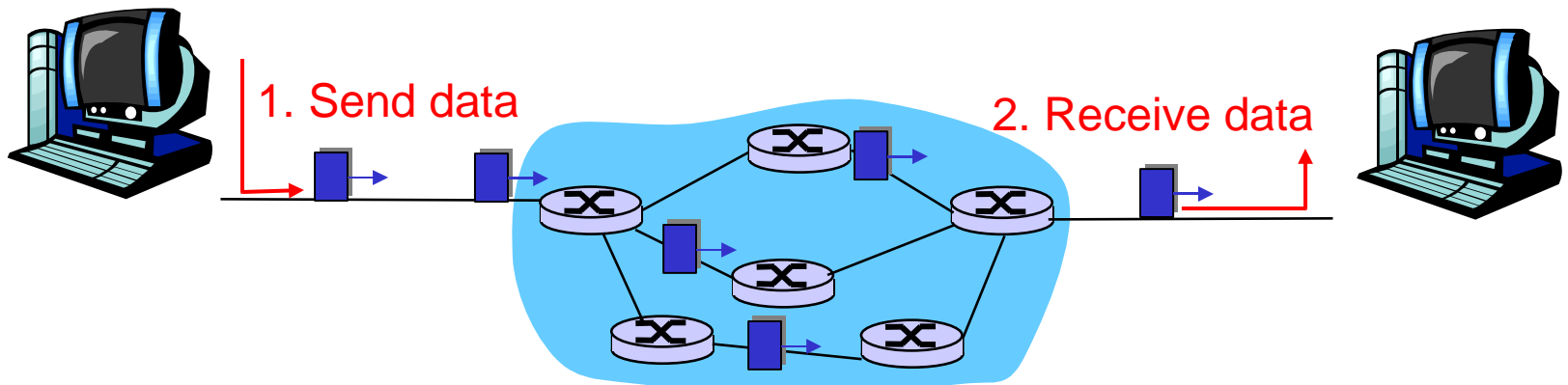
## Three important functions:

- *Path determination*: route taken by packets from source to dest. *Routing algorithms*
- *Forwarding*: move packets from router's input to appropriate router output
- *Call/circuit setup*: some network architectures require router call setup along path before data flows



# Datagram Networks: The Internet Model

- No call setup at network layer
- Routers: no state about end-to-end connections
  - no network-level concept of “connection”
- Packets forwarded using destination host address
  - packets between same source-dest pair may take different paths



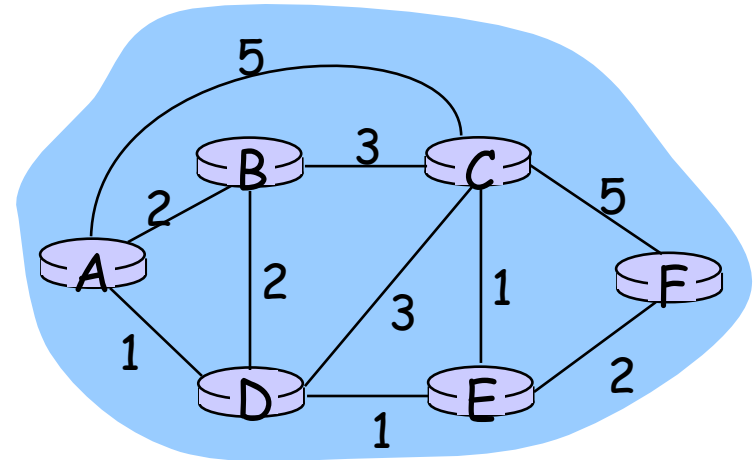
# Routing

## Routing protocol:

**Goal:** determine “good” path (sequence of routers) through network from source to destination

Graph abstraction for routing algorithms:

- Graph nodes are routers
- Graph edges are physical links
  - link cost: delay, \$ cost, or congestion level



- “Good” path:
  - typically means minimum cost path
  - other definitions possible

# 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 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 **routing table** for that node
- Iterative: after  $k$  iterations, know least cost path to  $k$  destinations

## Notation:

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



# Dijkstra's Algorithm

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  $D(v) = \text{infinity}$

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

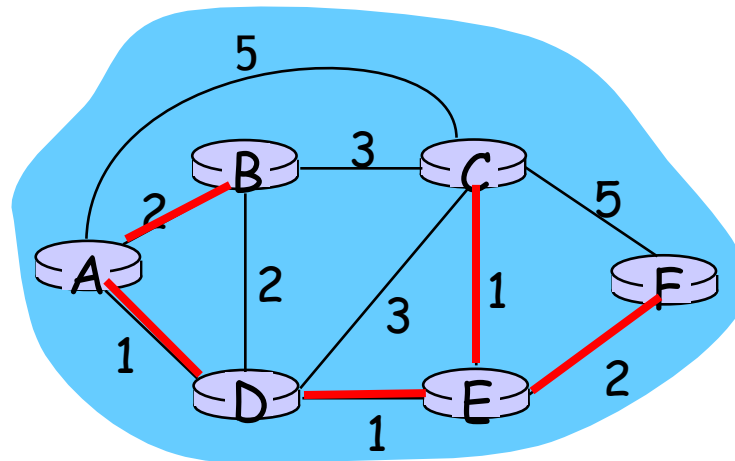
14 shortest path cost to  $w$  plus cost from  $w$  to  $v$  \*/

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



# Dijkstra's Algorithm: Example

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



# Dijkstra's Algorithm: Discussion

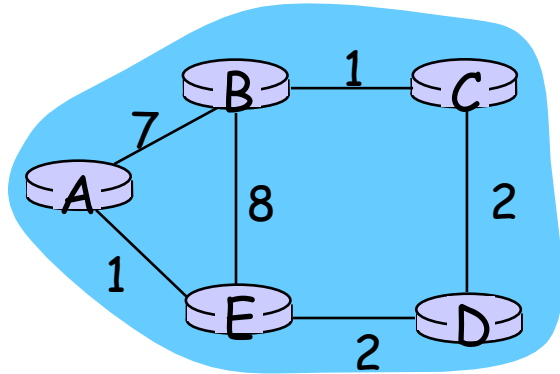
- Algorithm complexity:  $n$  nodes
  - each iteration: need to check all nodes,  $w$ , not in  $N$
  - $n(n+1)/2$  comparisons:  $O(n^2)$
  - more efficient implementations possible:  $O(n \log n)$

# Distance Vector Routing Algorithm

- **Iterative:**
  - Continues until no nodes exchange info.
  - Self-terminating: no “signal” to stop
- **Asynchronous:**
  - Nodes need not exchange info/iterate in lock step!
- **Distributed:**
  - Each node communicates only with directly-attached neighbors
- **Distance Table data structure**
  - Each node has its own
  - Row for each possible destination
  - Column for each directly-attached neighbor to node
- **Example: in node X, for dest. Y via neighbor Z:**

$$\begin{aligned} D^X(Y,Z) &= \text{distance from } X \text{ to } Y, \\ &\quad \text{via } Z \text{ as next hop} \\ &= c(X,Z) + \min_w \{D^Z(Y,w)\} \end{aligned}$$

# Distance Table: Example



cost to destination via

$D^E()$	A	B	D
A	1	14	5
B	7	8	5
C	6	9	4
D	4	11	2

destination

$$D^E(C,D) = c(E,D) + \min_w \{D^D(C,w)\} \\ = 2 + 2 = 4$$

$$D^E(A,D) = c(E,D) + \min_w \{D^D(A,w)\} \\ = 2 + 3 = 5 \quad \text{loop!}$$

$$D^E(A,B) = c(E,B) + \min_w \{D^B(A,w)\} \\ = 8 + 6 = 14 \quad \text{loop!}$$

# Distance Table Gives Forwarding Table

cost to destination via

$D^E()$	A	B	D
A	1	14	5
B	7	8	5
C	6	9	4
D	4	11	2

destination

Outgoing link to use, cost

A	A,1
B	D,5
C	D,4
D	D,4

destination

Distance table

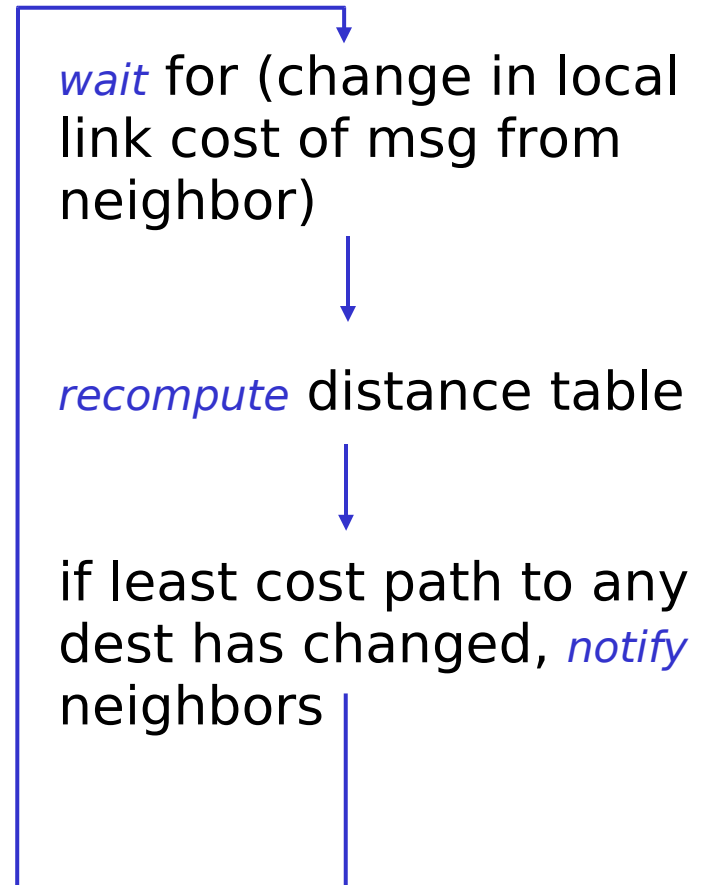


Forwarding table

# Distance Vector Routing: Overview

- Iterative, asynchronous:
  - each local iteration caused by:
    - local link cost change
    - message from neighbor: its least cost path change from neighbor
- Distributed:
  - 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

At all nodes X:

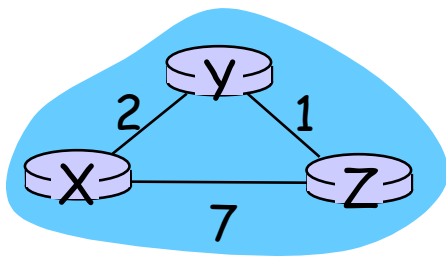
- 1 Initialization:
- 2 for all adjacent nodes v:
- 3      $D^X(*,v) = \text{infinity}$      /\* the \* operator means "for all rows" \*/
- 4      $D^X(v,v) = c(X,v)$
- 5 for all destinations, y
- 6     send  $\min_w D^X(y,w)$  to each neighbor /\* w over all X's neighbors \*/



# Distance Vector Algorithm (more)

```
8 loop
9   wait (until I see a link cost change to neighbor V
10         or until I receive update from neighbor V)
11
12   if (c(X,V) changes by d)
13     /* change cost to all dest's via neighbor v by d */
14     /* note: d could be positive or negative */
15     for all destinations y:  $D^X(y,V) = D^X(y,V) + d$ 
16
17   else if (update received from V wrt destination Y)
18     /* shortest path from V to some Y has changed */
19     /* V has sent a new value for its  $\min_w D^V(Y,w)$  */
20     /* call this received new value is "newval" */
21     for the single destination y:  $D^X(Y,V) = c(X,V) + \text{newval}$ 
22
23   if we have a new  $\min_w D^X(Y,w)$  for any destination Y
24     send new value of  $\min_w D^X(Y,w)$  to all neighbors
25
26 forever
```

# Distance Vector Algorithm: Example



		cost via	
		Y	Z
dest	D <sup>X</sup>		
	Y	2	∞
Z	∞	7	

		cost via	
		Y	Z
dest	D <sup>X</sup>		
	Y	2	8
Z	3	7	

		cost via	
		Y	Z
dest	D <sup>X</sup>		
	Y		
Z			

		cost via	
		X	Z
dest	D <sup>Y</sup>		
	X	2	∞
Z	∞	1	

		cost via	
		X	Z
dest	D <sup>Y</sup>		
	X	2	8
Z	9	1	

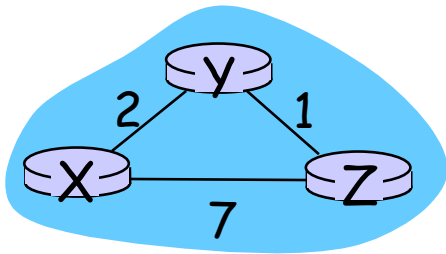
		cost via	
		X	Z
dest	D <sup>Y</sup>		
	X		
Z			

		cost via	
		X	Y
dest	D <sup>Z</sup>		
	X	7	∞
Y	∞	1	

		cost via	
		X	Y
dest	D <sup>Z</sup>		
	X	7	3
Y	9	1	

		cost via	
		X	Y
dest	D <sup>Z</sup>		
	X		
Y			

# Distance Vector Algorithm: Example



		cost via	
		Y	Z
d e s t	D <sup>X</sup>		
	Y	2	∞
Z	∞	7	

		cost via	
		X	Z
d e s t	D <sup>Y</sup>		
	X	2	∞
Z	∞	1	

		cost via	
		X	Y
d e s t	D <sup>Z</sup>		
	X	7	∞
Y	∞	1	

		cost via	
		Y	Z
d e s t	D <sup>X</sup>		
	Y	2	8
Z	3	7	

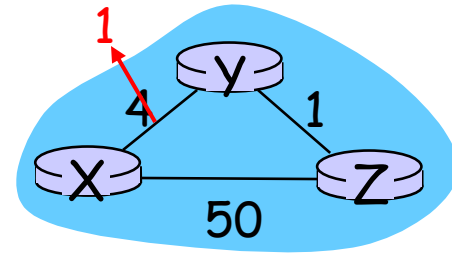
$$D^X(Y,Z) = c(X,Z) + \min_w \{D^Z(Y,w)\} \\ = 7 + 1 = 8$$

$$D^X(Z,Y) = c(X,Y) + \min_w \{D^Y(Z,w)\} \\ = 2 + 1 = 3$$

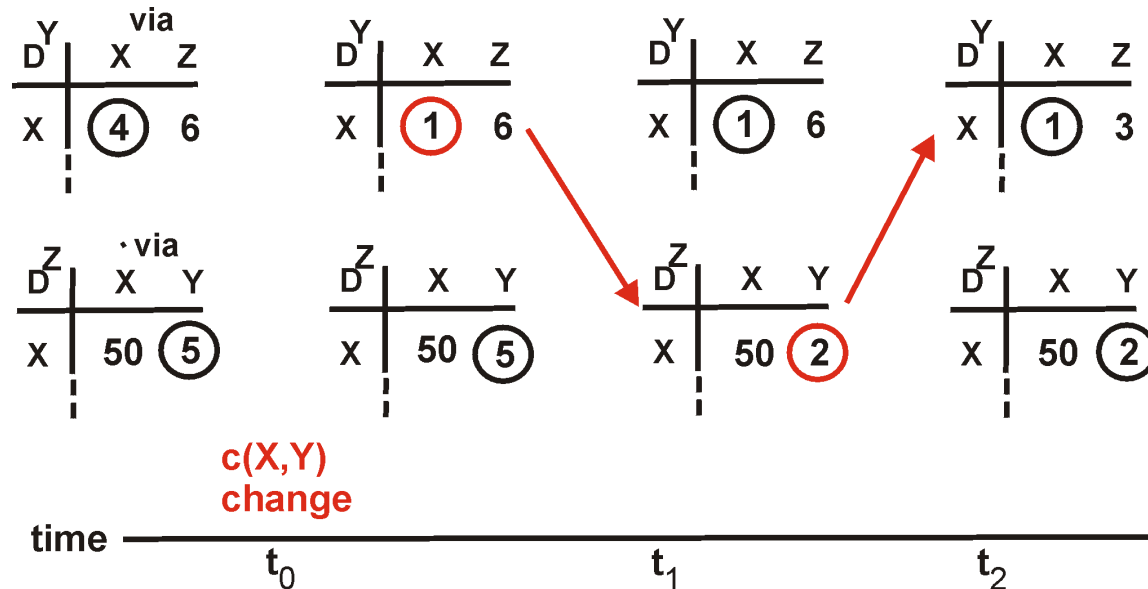
# Distance Vector: Link Cost Changes

## Link cost changes:

- node detects local link cost change
- updates distance table (line 15)
- if cost change in least cost path, notify neighbors (lines 23,24)



“good news travels fast”

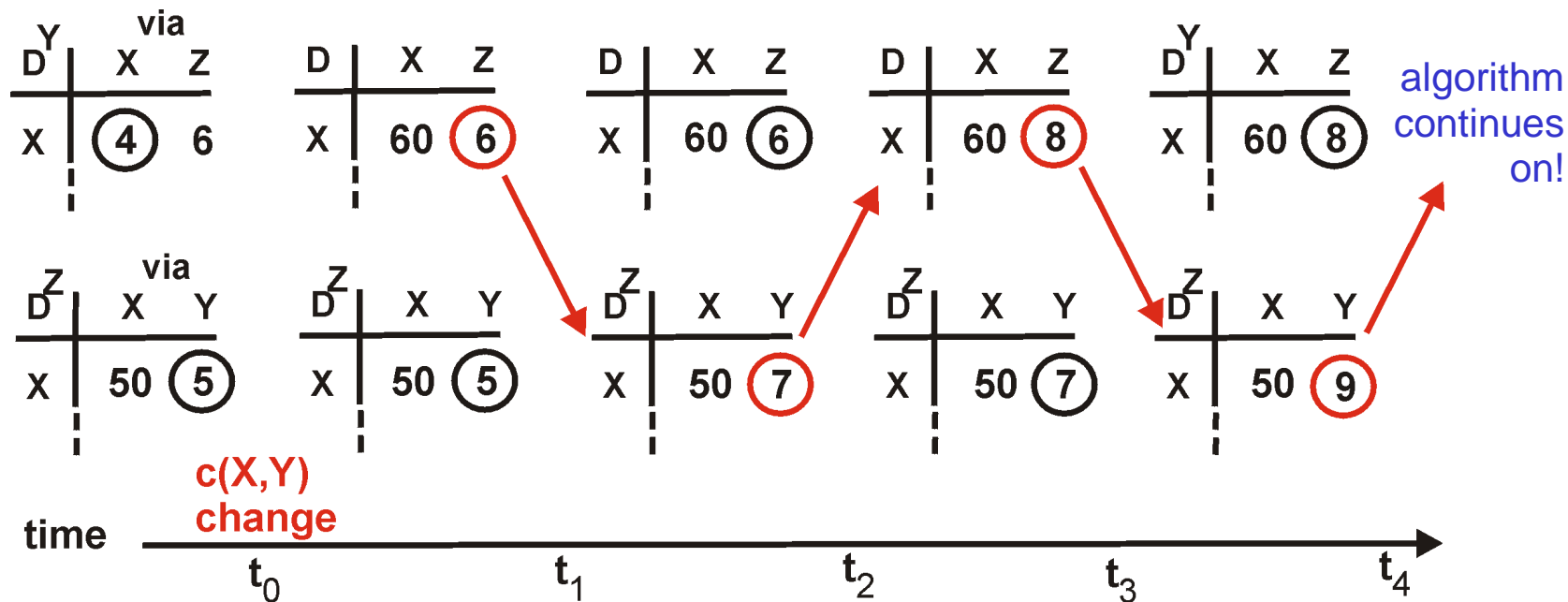
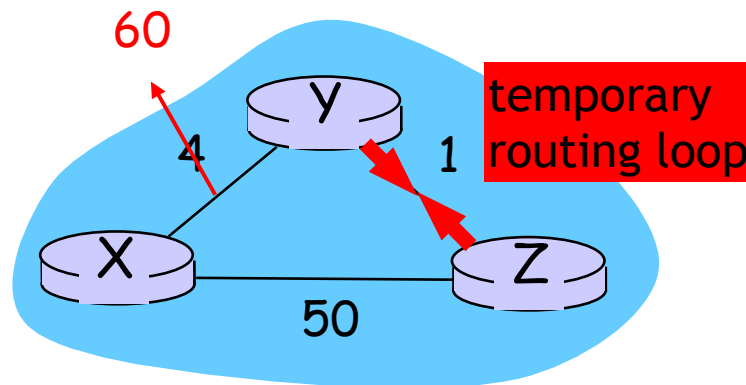


algorithm terminates

# Distance Vector: Link Cost Changes

## Link cost changes:

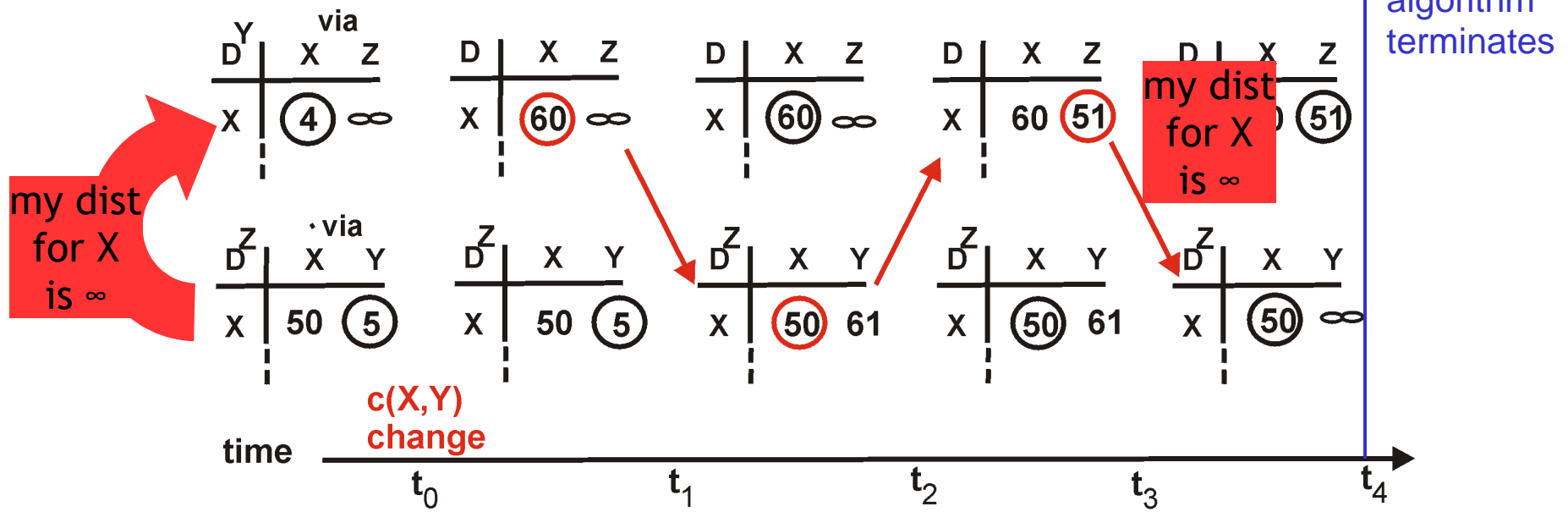
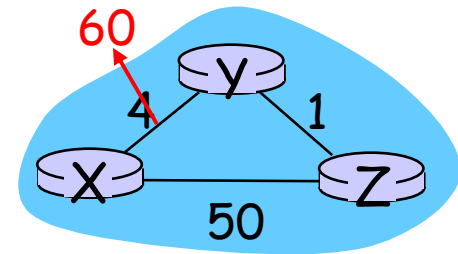
- good news travels fast
- bad news travels slow -  
“count to infinity” problem!



# DV Count-to- $\infty$ : 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?



# Comparison of LS and DV Algorithms

- **Message complexity**
  - LS: with  $n$  nodes,  $E$  links,  $O(nE)$  msgs sent each
  - 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- $\infty$  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
      - errors propagate through network

# Summary: Routing Principles

- Network layer service model:
  - Host-to-host datagram service
  - Best-effort
- Routing algorithms
  - Link-state: Dijkstra
    - centralized algorithm to compute shortest-path tree to/from a vertex
  - Distance-vector: Bellman-Ford
    - distributed algorithm to compute all shortest paths