Network Layer

goals:

- understand principles behind network layer services:
  - routing (path selection)
  - how a router works
- instantiation and implementation in the Internet

Overview:

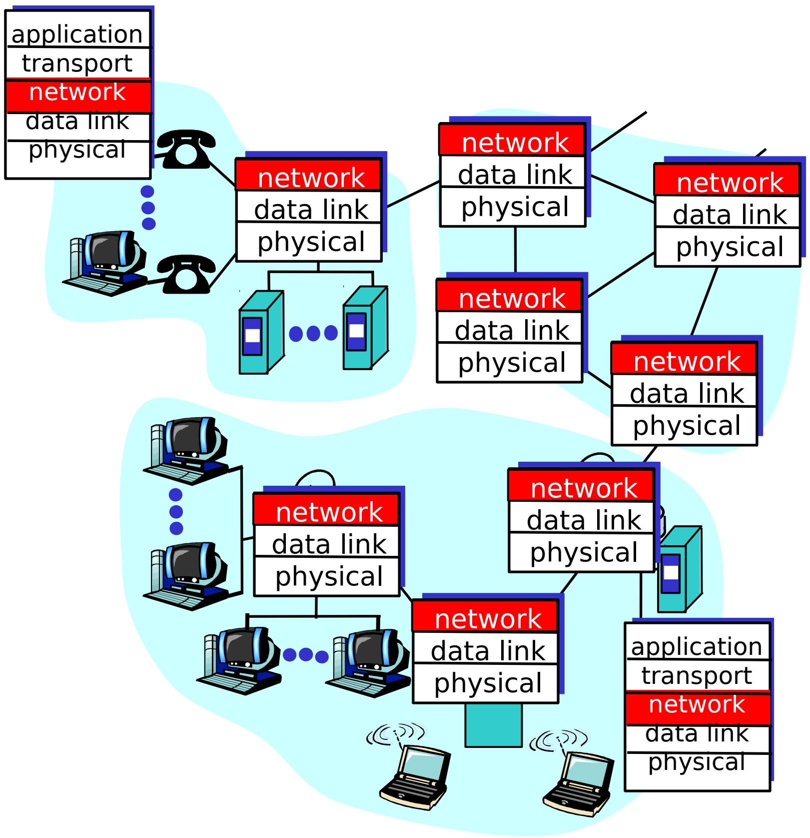
- network layer services
- routing principle: path selection
- hierarchical routing
- IP
- what’s inside a router?

Network layer functions

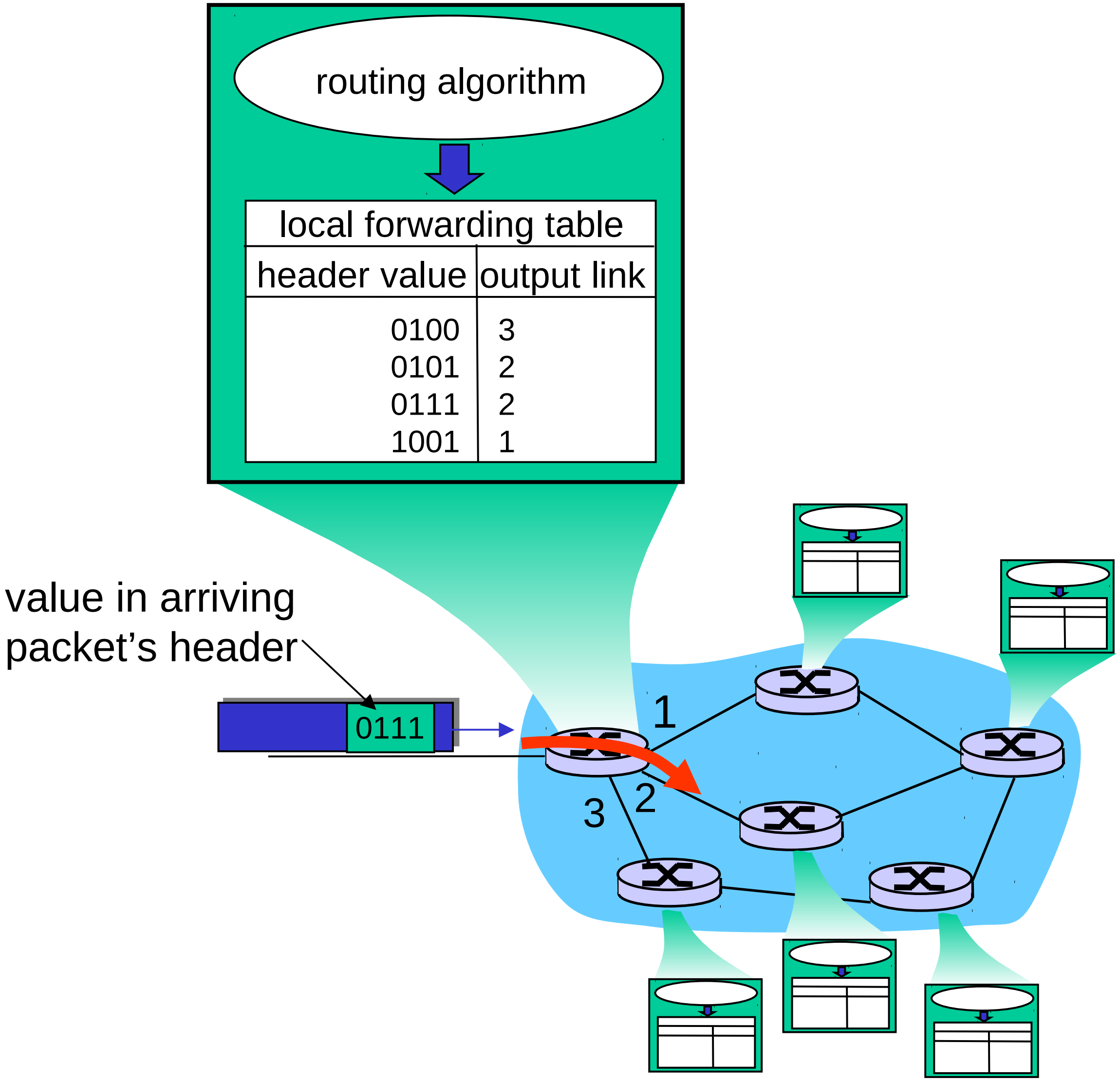
- transport packet from sending to receiving hosts
- network layer protocols in *every* host, router

important functions:

- *path determination:* route taken by packets from source to dest. *Routing algorithms*
- *switching:* move packets from router's input to appropriate router output

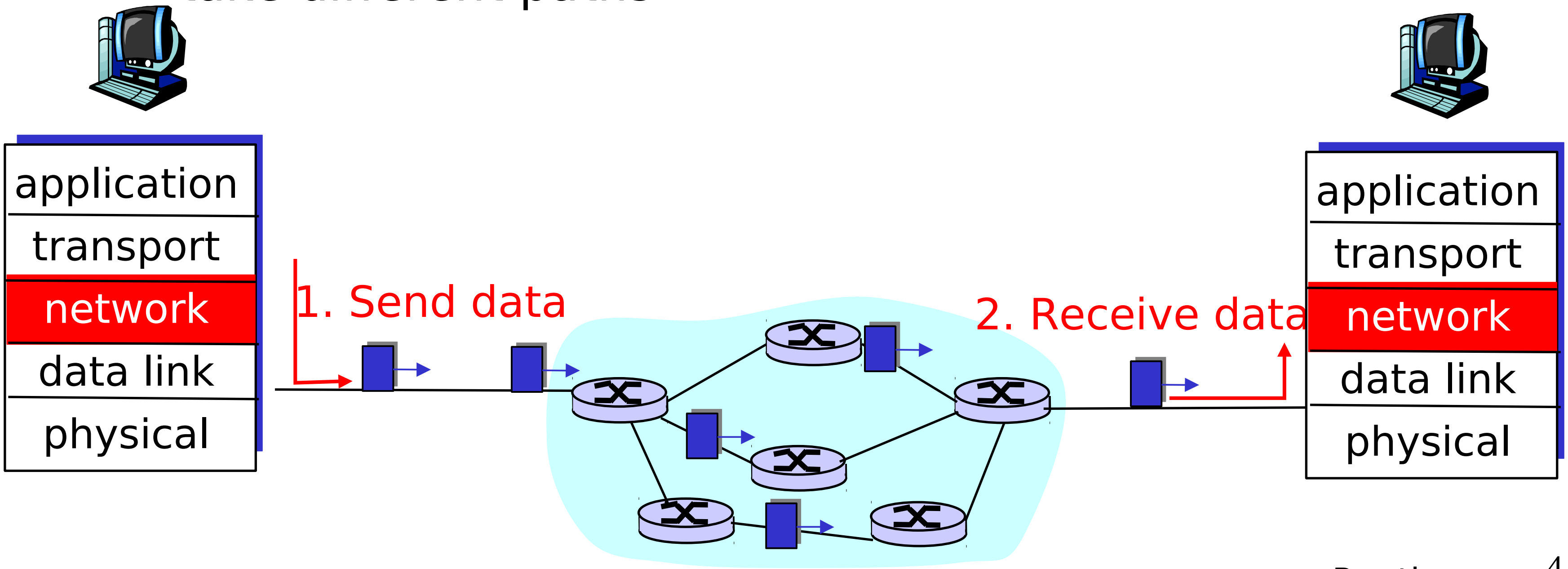


Interplay between routing and forwarding



The internet model

- routers: no state about end-to-end connections
  - no network-level concept of 'connection'
- packets typically routed using destination host ID
  - packets between same source-destination pair may take different paths



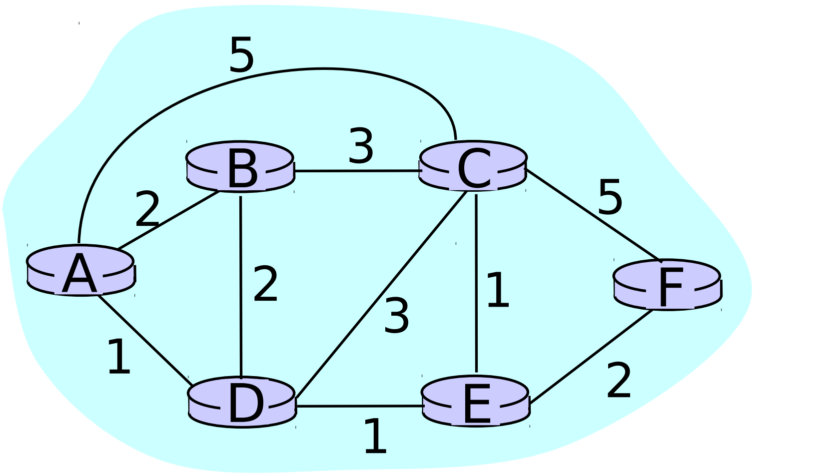
Routing

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 neighbours, link costs to neighbours
- iterative process of computation, exchange of info with neighbours
- 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 forwarding table for that node
- iterative: after k iterations, know least cost path to k dest.'s

Notation:

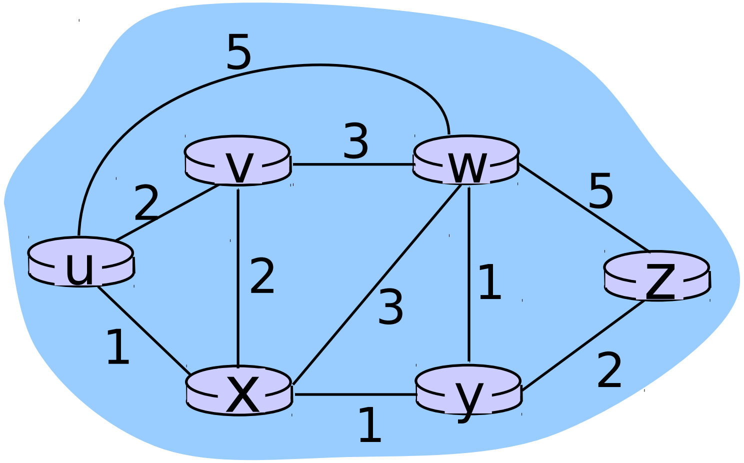
- c(x,y): link cost from node x to y; = ∞ if not direct neighbors
- D(v): current value of cost of path from source to dest. v
- p(v): predecessor node along path from source to v
- N': set of nodes whose least cost path definitively known

Dijsktra’s Algorithm

```
Initialization:
  N' = {u}
  for all nodes v
    if v adjacent to u
        then D(v) = c(u,v)
    else D(v) = ∞

Loop
  find w not in N' such that D(w) is a minimum
  add w to N'
  update D(v) for all v adjacent to w and not in N' :
     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 N'
```

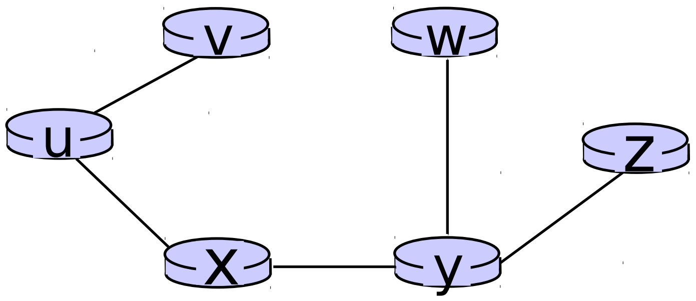
Dijkstra's algorithm: example

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	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



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)

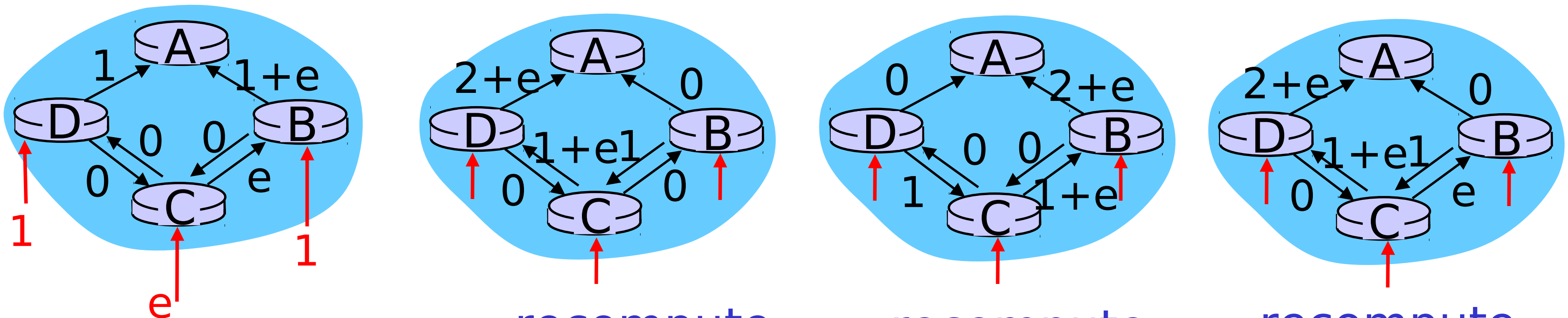
Dijkstra’s algorithm, discussion

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., link cost = amount of carried traffic



initially

… recompute routing

… recompute

… recompute

Distance Vector Routing Algorithm

Bellman-Ford 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 neighbours

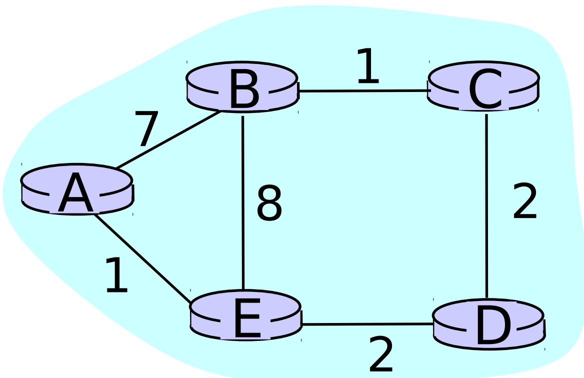
Distance Table data structure

- each node has its own
- row for each possible destination
- column for each directly-attached neighbor to node
- example: in node X, for destination Y via neighbor Z:

$D^X(Y,Z)$ = distance *from* X *to* Y, *via* Z as next hop

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

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

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

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

loop!

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

loop!

Distance table gives routing 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,2

destination

Distance table → Routing table

Distance Vector Routing: overview

Iterative, asynchronous: each local iteration caused by:

- local link cost change
- message from neighbour: its least cost path change from neighbor

Distributed:

- each node notifies neighbours *only* when its least cost path to any destination changes
  - neighbours then notify their neighbours if necessary

Each node:

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

recompute distance table

if least cost path to any destination has changed, *notify* neighbors

Distance Vector Algorithm:

At all nodes, X:

Initialization:

for all adjacent nodes v:

$D^X(*,v)$ =infinite (the * operator means "for all rows")

$D^X(v,v) = c(X,v)$

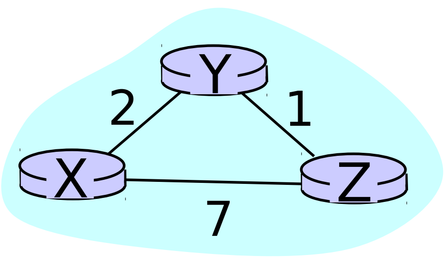
for all destinations, y

send $\min_w D^X(y,w)$ to each neighbor (w over all X's neighbors)

Distance Vector Algorithm (cont.):

```
loop
  wait (until I see a link cost change to neighbor V
        or until I receive update from neighbor V)

  if (c(X,V) changes by d)
    /* change cost to all dest's via neighbor v by d */
    /* note: d could be positive or negative */
    for all destinations y:  D^X(y,V) =  D^X(y,V) + d

  else if (update received from V wrt destination Y)
    /* shortest path from V to some Y has changed  */
    /* V has sent a new value for its  min_w DV(Y,w) */
    /* call this received new value is newval     */
    for the single destination y: D^X(Y,V) = c(X,V) + newval

  if we have a new min_w D^X(Y,w)for any destination Y
     send new value of min_w D^X(Y,w) to all neighbors

forever
```

Distance Vector Algorithm: example



D^X	cost via Y	cost via Z
dest Y	2	∞
dest Z	∞	7

D^Y	cost via X	cost via Z
dest X	2	∞
dest Z	∞	1

D^Z	cost via X	cost via Y
dest X	7	∞
dest Y	∞	1

D^X	cost via Y	cost via Z
dest Y	2	8
dest Z	3	7

D^Y	cost via X	cost via Z
dest X	2	8
dest Z	9	1

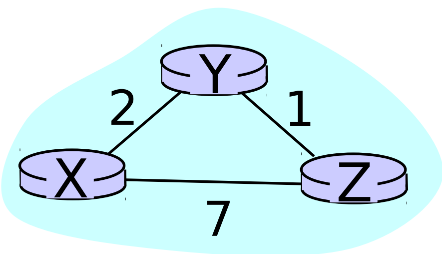
D^Z	cost via X	cost via Y
dest X	7	3
dest Y	9	1

D^X	cost via Y	cost via Z
dest Y		
dest Z		

D^Y	cost via X	cost via Z
dest X		
dest Z		

D^Z	cost via X	cost via Y
dest X		
dest Y		

Distance Vector Algorithm: example



D^X (dest)	cost via Y	cost via Z
Y	2	∞
Z	∞	7

D^Y (dest)	cost via X	cost via Z
X	2	∞
Z	∞	1

D^Z (dest)	cost via X	cost via Y
X	7	∞
Y	∞	1

D^X (dest)	cost via Y	cost via Z
Y	2	8
Z	3	7

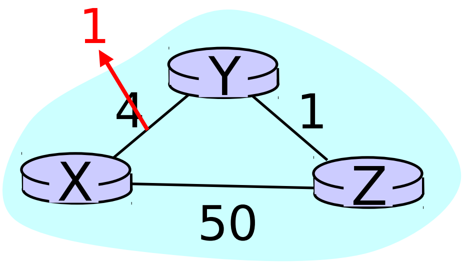
$$D^X(Y,Z) = c(X,Z) + \min_w\{D^Z(Y,w)\} = 7+1 = 8$$

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

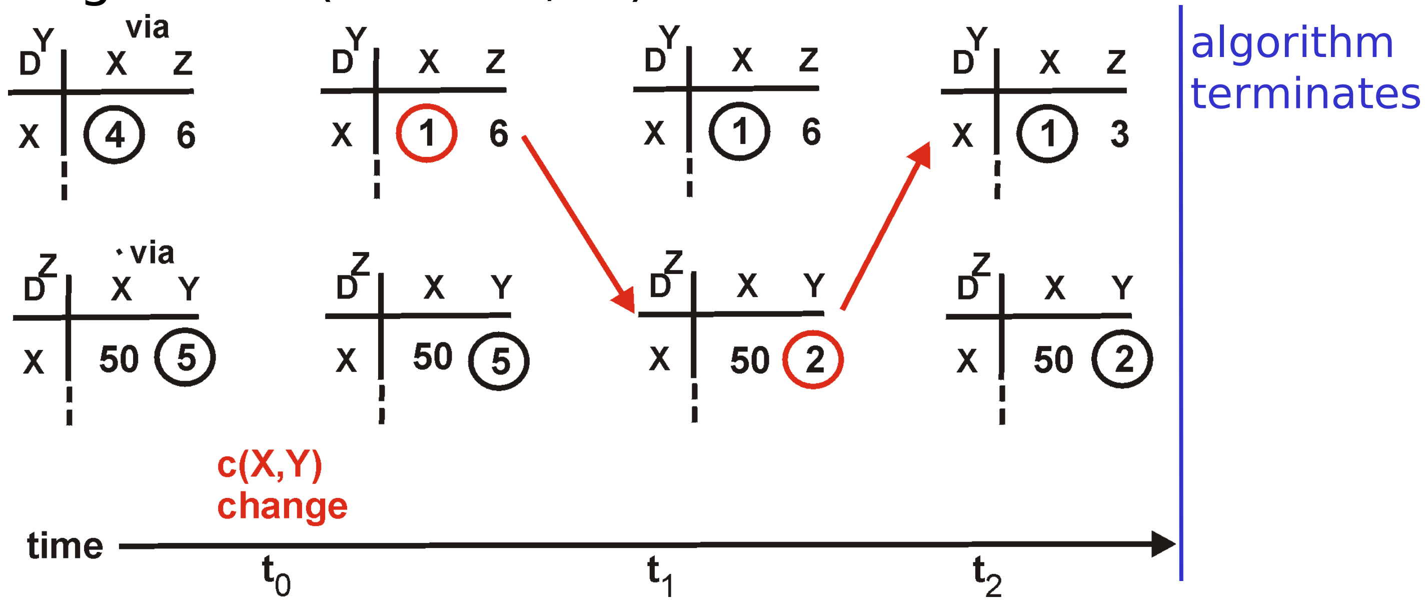
Distance Vector: link cost changes

Link cost changes:

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



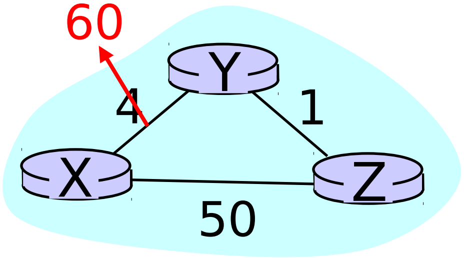
“good news travels fast”

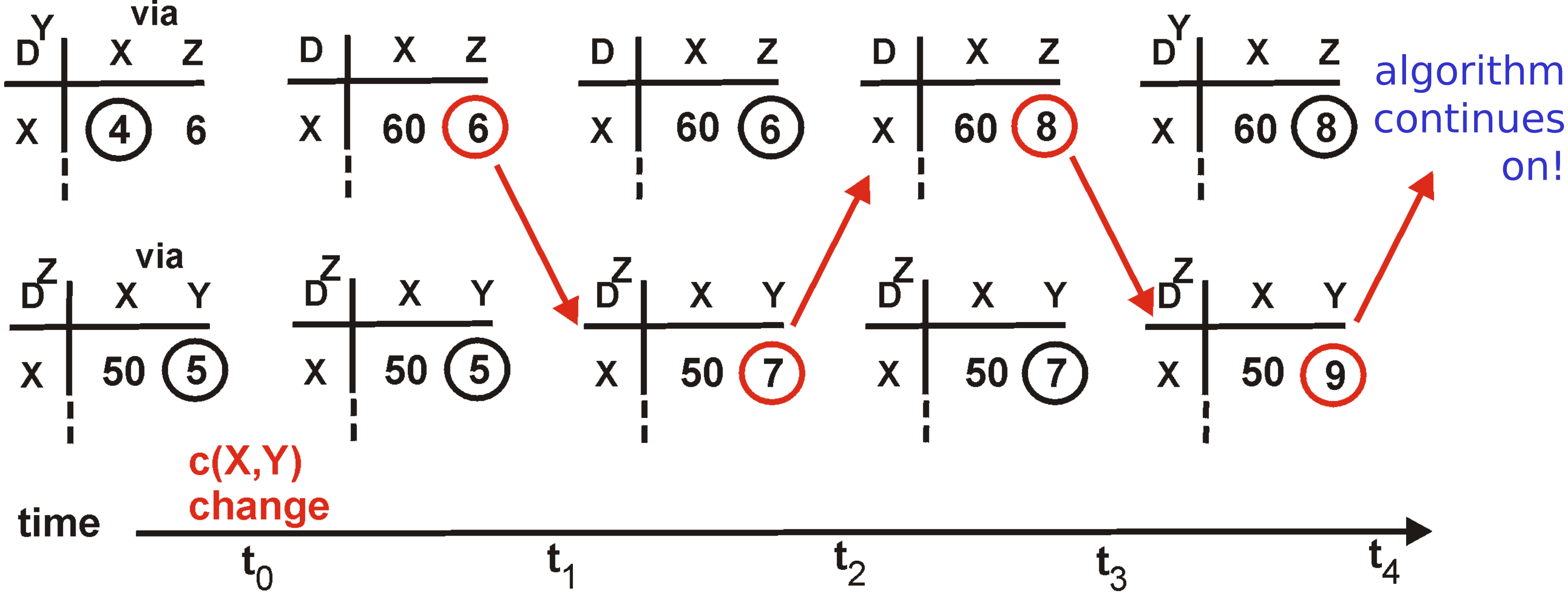


Distance Vector: link cost changes

Link cost changes:

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





Distance Vector: 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?