DIPARTIMENTO DI INFORMATICA
Via Salaria 113 - 00198, Roma

SAPIENZA UNIVERSITÀ DI ROMA

Chapter 4
Network Layer

Reti di Elaboratori
Corso di Laurea in Informatica
Università degli Studi di Roma "La Sapienza"
Canale A-L

Prof.ssa Chiara Petrioli

Parte di queste slide sono state prese dal materiale associato al libro *Computer Networking: A Top Down Approach* , 5th edition.

Thanks also to Antonio Capone, Politecnico di Milano, Giuseppe Bianchi and Francesco LoPresti, Un. di Roma Tor Vergata

Chapter 4: Network Layer

- 4. 1 Introduction
- 4.2 Virtual circuit and datagram networks
- 4.3 What's inside a router
- 4.4 IP: Internet Protocol
  - Datagram format
  - IPv4 addressing
  - ICMP
  - IPv6
- 4.5 Routing algorithms
  - Link state
  - Distance Vector
  - Hierarchical routing
- 4.6 Routing in the Internet
  - RIP
  - OSPF
  - BGP
- 4.7 Broadcast and multicast routing

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

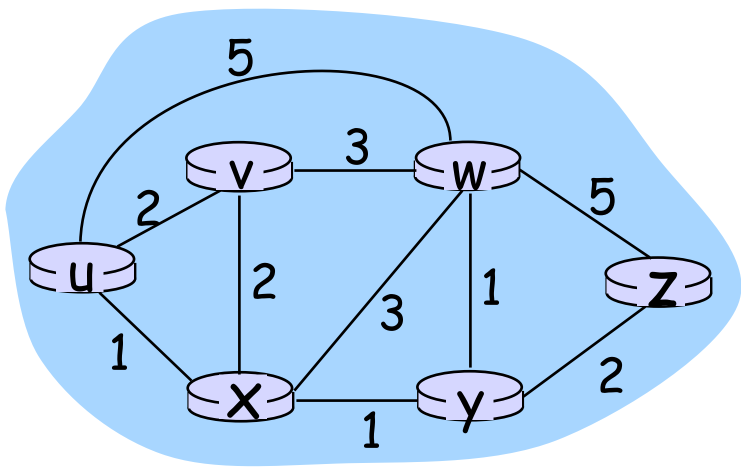
D(v)=D(u,v)

```
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'
```

Grafo pesato diretto con pesi non negativi

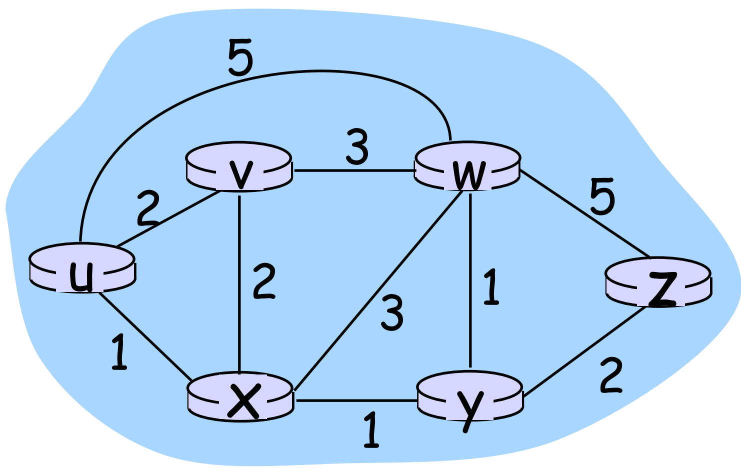
Dijkstra's algorithm: example

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					



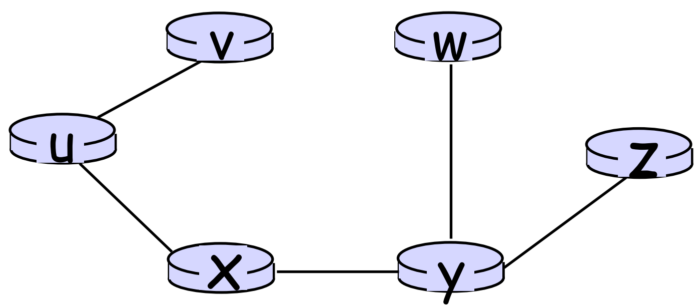
Dijkstra's algorithm: example

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)

Correttezza

Se eseguiamo l'algoritmo di Dijkstra su un grafo pesato diretto $G=(N,E)$ con pesi non negativi, sorgente u, e funzione peso c allora alla terminazione $D(v)=\delta(u,v)$, per ogni nodo v in N. (dove $\delta(u,v)$ indica la lunghezza del cammino di peso minimo tra u e v).

Dim.

D(v) non è più aggiornato nel momento in cui v è inserito in N'. Dovremmo quindi mostrare che $D(v)=\delta(u,v)$ nel momento in cui v è inserito in N', per ogni v. Ragioniamo per assurdo. Sia x il primo nodo (nell'ordine di inserimento in N') per cui vale $D(x)!=\delta(u,x)$ al momento in cui x è inserito nell'insieme N' (linea 10 dell'algoritmo). x!=u dato che u, nodo sorgente, è inserito nella fase di inizializzazione e per lui vale $D(u)=\delta(u,v)=0$. Inoltre deve esistere un percorso di costo non infinito da u a x dato che altrimenti varrebbe che il valore a cui D(x) è inizializzato (infinito) sarebbe uguale a $\delta(u,v)$. Quindi esiste un percorso di costo minimo p=u…v→y…x dove y è il primo nodo sul percorso di costo minimo NON in N' (quindi u….v sono TUTTI in N'). Il percorso p può quindi essere diviso in due percorsi: p1 che va da u a y e p2 che va da y a x.

Da notare che il percorso p1 è anch'esso il percorso di costo minimo che unisce u a y (se non lo fosse e ci fosse un percorso p3 che unisce u a y di costo < del costo di p1, allora la concatenazione di p2 e p2 sarebbe un percorso p' da u a x di costo < di p,contro l'assunto che p sia un percorso di costo minimo).

…Correttezza

Se eseguiamo l'algoritmo di Dijkstra su un grafo pesato diretto $G=(N,E)$ con pesi non negativi, sorgente u, e funzione peso c allora alla terminazione $D(v)=\delta(u,v)$, per ogni nodo v in N. (dove $\delta(u,v)$ indica la lunghezza del cammino di peso minimo tra u e v).

Dim (…continua).

Quando x è inserito in N' $D(y)=\delta(u,y)$. Ifnatti in quel momento v è stato già inserito in N' e dopo il suo inserimento y ha ricalcolato $D(y)=D(v)+c(v,y)=\delta(u,v)+c(v,y)$ (dato che per ipotesi x è il primo nodo per cui all'inserimento in N' la stima dei costi non corrisponde al percorso di costo minimo)$=\delta(u,y)$.

Dato che y precede x sul percorso minimo ed i pesi sugli archi sono non negativi vale che:

$\delta(u,x)>=\delta(u,y)=D(y)$

e quindi anche che

$D(x)>=\delta(u,x)>=\delta(u,y)=D(y)$

D'altro canto dato che x viene inserito in N' prima di y vale che

$\delta(u,x)<=D(x)<=D(y)=\delta(u,y)$

Quindi

$\delta(u,x)=D(x)=D(y)=\delta(u,y)$

Cosa che porta alla contraddizione.

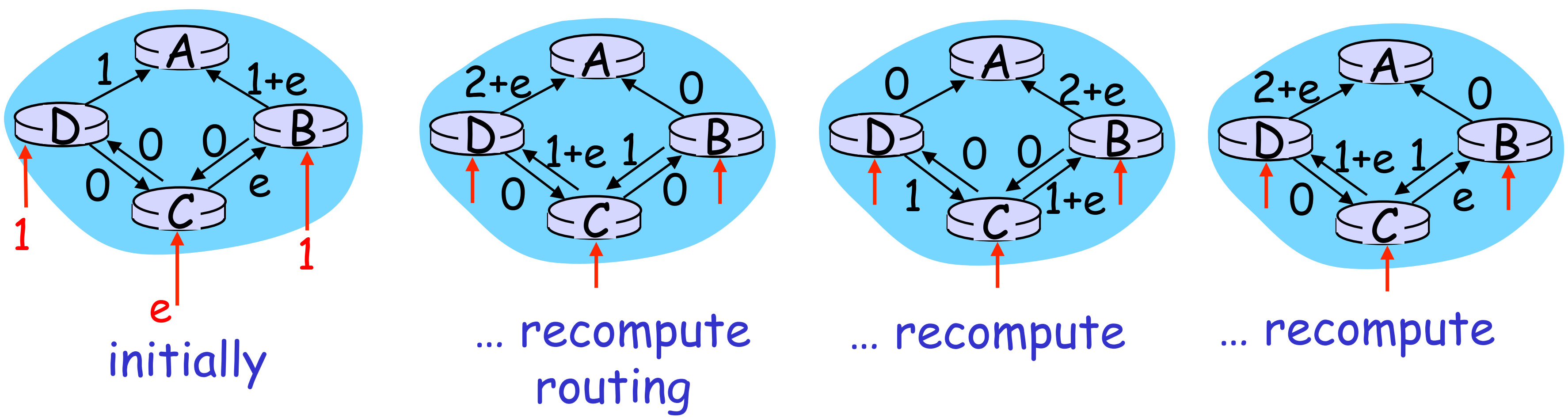
Dijkstra's algorithm, discussion

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(nlogn)

Oscillations possible:

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



Chapter 4: Network Layer

- 4. 1 Introduction
- 4.2 Virtual circuit and datagram networks
- 4.3 What's inside a router
- 4.4 IP: Internet Protocol
  - Datagram format
  - IPv4 addressing
  - ICMP
  - IPv6
- 4.5 Routing algorithms
  - Link state
  - Distance Vector
  - Hierarchical routing
- 4.6 Routing in the Internet
  - RIP
  - OSPF
  - BGP
- 4.7 Broadcast and multicast routing

Bellman-Ford

Given a graph $G=(N,A)$ and a node *s* finds the shortest path from *s* to every node in N.

A shortest walk from *s* to i subject to the constraint that the walk contains at most h arcs and goes through node *s* only once, is denoted shortest(<=h) walk and its length is D^h_i.

Bellman-Ford rule:

Initiatilization $D^h_s=0$, for all h; $w_{i,k}$ = infinity if (i,k) NOT in *A*; $w_{k,k}=0$; **D^0_i=infinity for all i!=s.**

Iteration:

$$D^{h+1}_i=\min_k [w_{i,k} + D^h_k]$$

Assumption: non negative cycles **(this is the case in a network!!)**

The Bellman-Ford algorithm first finds the one-arc shortest walk lengths, then the two-arc shortest walk length, then the three-arc…etc. →distributed version used for routing

Bellman-Ford

$$D^{h+1}_i = \min_k [w_{i,k} + D^h_k]$$

Can be computed locally.
What do I need?

For each neighbor *k*, I need to know
-the cost of the link to it (known info)
-The cost of the best route from the neighbor *k* to the destination (←this is an info that each of my neighbor has to send to me via messages)

In the real world: I need to know the best routes among each pair of nodes → we apply distributed Bellman Ford to get the best route for each of the possible destinations

Distance Vector Routing Algorithm -Distributed Bellman Ford

iterative:
- continues until no nodes exchange info.
- *self-terminating*: no "signal" to stop

asynchronous:
- nodes need *not* exchange info/iterate in lock step!

Distributed, based on local info:
- 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:

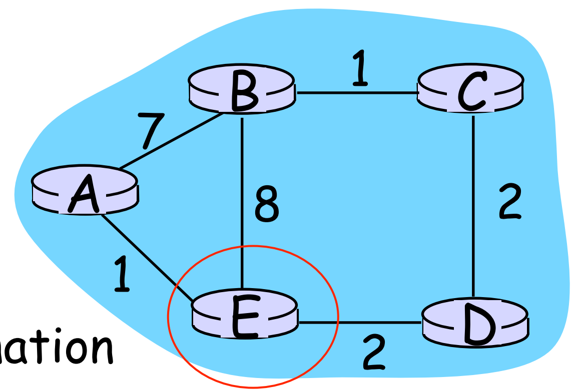
Cost associated to the (X,Z) link

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

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

Info maintained at Z. Min must be communicated

Distance Table: example



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! Best path from D goes through E

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

loop!

Path B-C-D-E-A

Distance table in node E after the algorithm has converged

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

First example

Distance table gives routing table

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

cost to destination via (columns); destination (rows)

	Outgoing link to use, cost
A	A,1
B	D,5
C	D,4
D	D,2

destination (rows)

Distance table → Routing 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:

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

recompute distance table

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

Distance Vector Algorithm:

At all nodes, X:

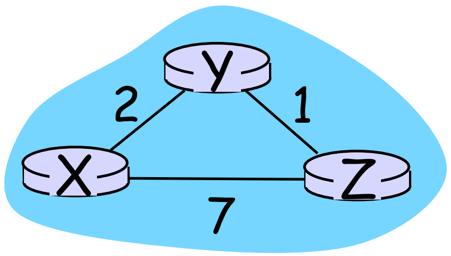
```
Initialization:
 for all adjacent nodes v:
    D^X(*,v) = infinity        /* the * operator means "for all rows" */
    D^X(v,v) = c(X,v)
 for all destinations, y
    send min_w D^X(y,w) to each neighbor  /* w over all X's neighbors */
```

From the node to whatever destination going through v

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

dest	Y	Z
Y	2	∞
Z	∞	7

D^Y — cost via

dest	X	Z
X	2	∞
Z	∞	1

D^Z — cost via

dest	X	Y
X	7	∞
Y	∞	1

D^X — cost via

dest	Y	Z
Y	2	8
Z	3	7

D^Y — cost via

dest	X	Z
X	2	8
Z	9	1

D^Z — cost via

dest	X	Y
X	7	3
Y	9	1

D^X — cost via

dest	Y	Z
Y		
Z		

D^Y — cost via

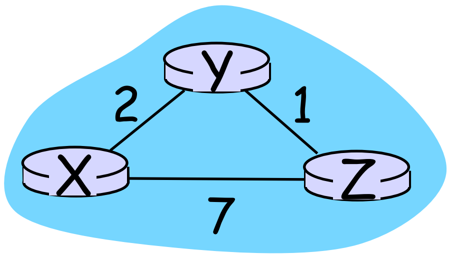
dest	X	Z
X		
Z		

D^Z — cost via

dest	X	Y
X		
Y		

Cost updates from the neighbors are used for sake of recomputing
The best routes and may lead to new cost updates…

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

Line 21 of the algorithm description

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

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

Distributed Bellman Ford correctness

- Completely asynchronous
- Starting from arbitrary estimates of the cost of the 'best route' from node i to the destination, if:
  - links weights are constant for enough time for the protocol to converge
  - stale info expire after a while
  - once in a while updated info are sent from a node to its neighbors

the Distributed Bellman Ford algorithm converges, i.e. each node correctly estimates the cost of the best route to the destination

Previous lecture. Summary:

Distributed Belman Ford

- Based on Distributed Bellman Ford Equation

Cost associated to the (X,Z) link

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

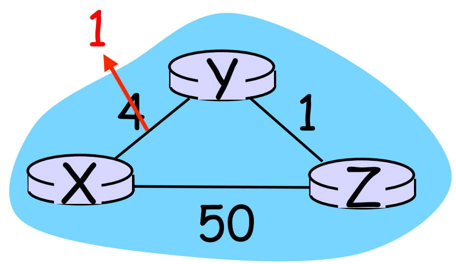
- $D^x(Y,Z)$ recomputed:
  - Upon reception of updates from the neighbors
  - Upon link cost change
- $\min_z D^X(Y,Z)$ communicated to the neighbors whenever its value changes, or periodically
- How long does it take for the algorithm to converge? 'good news travel fast, bad news may not→count to infinity'

Distance Vector: link cost changes

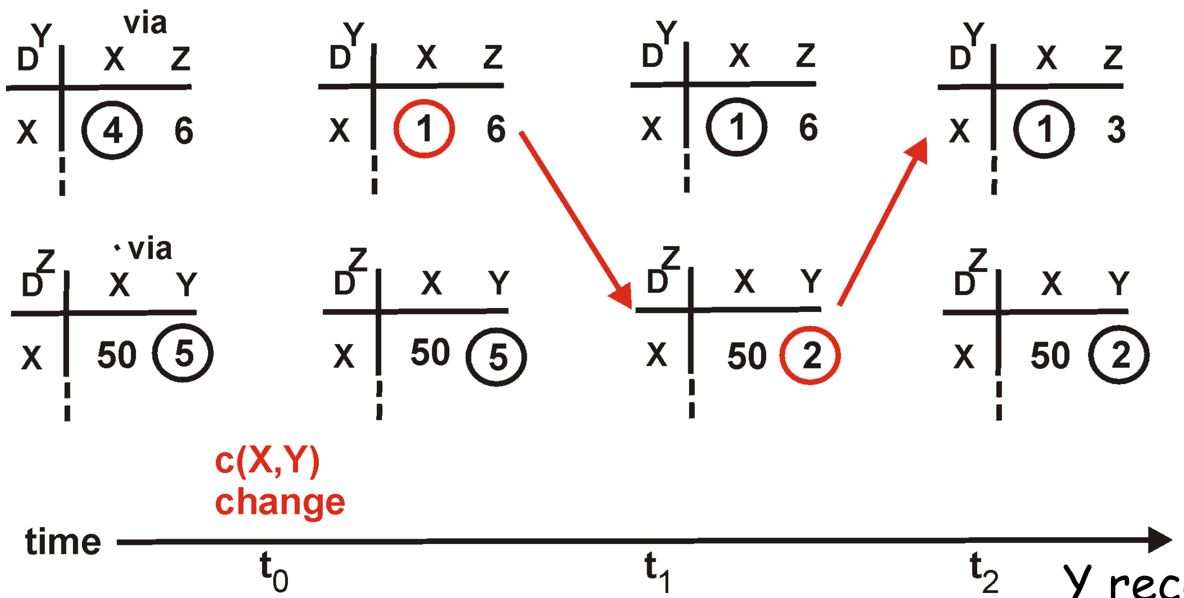
Subtitle: Distributed Bellman Ford converges *but* how fast?

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

Ydetects change informs neighbors

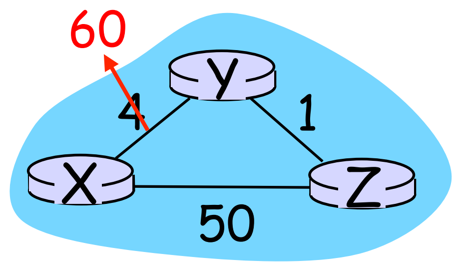
Z receives update updates table, new least Cost→informs neighbors

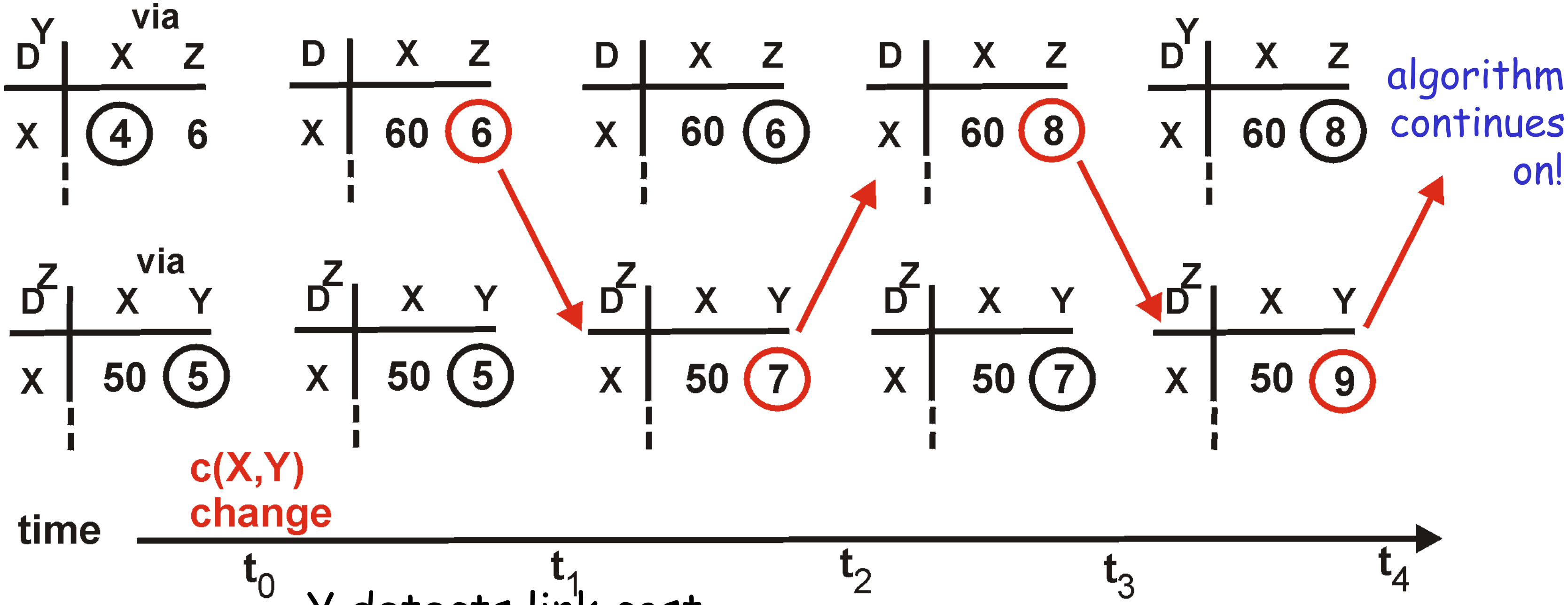
Y receives new least cost; no modification in the routing table, No updates sent

Distance Vector: link cost changes

Link cost changes:

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





Y detects link cost Increase but think can Reach X through Z at a total cost of 6 (wrong!!)

The path is Y-Z-Y-X

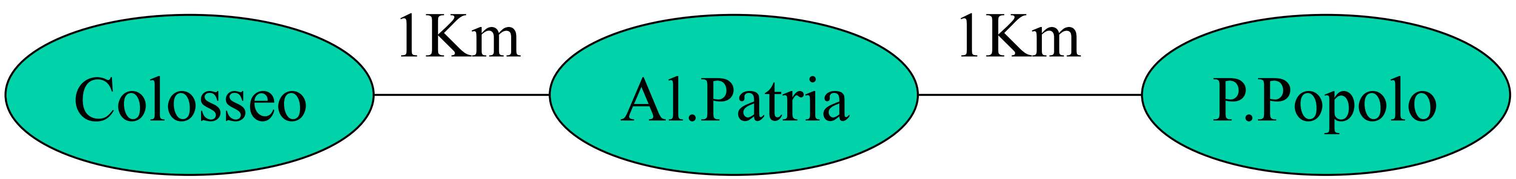
Count-to-infinity -an everyday life example

Which is the problem here?

the info exchanged by the protocol!! 'the best route to X I have has the following cost…' (no additional info on the route)

A Roman example…
-assumption: there is only one route going from Colosseo to Altare della Patria: Via dei Fori Imperiali. Let us now consider a network, whose nodes are Colosseo., Altare della Patria, Piazza del Popolo

Count-to-infinity –everyday life example (2/2)



The Colosseo. and Alt. Patria nodes exchange the following info

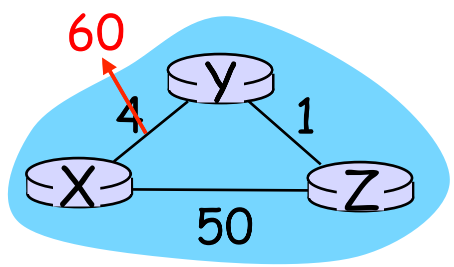
- Colosseo says 'the shortest route from me to P. Popolo is 2 Km'
- Alt. Patria says 'the shortest path from me to P. Popolo is 1Km'

Based on this exchange from Colosseo you go to Al. Patria, and from there to Piazza del Popolo OK **Now due to the big dig they close Via del Corso (Al. Patria—P.Popolo)**

- Al. Patria thinks 'I have to find another route from me to P.Popolo. Look there is a route from Colosseo to P.Popolo that takes 2Km, I can be at Colosseo in 1Km → I have found a 3Km route from me to P.Popolo!!' Communicates the new cost to Colosseo that updates 'OK I can go to P.Popolo via Al. Patria in 4Km'

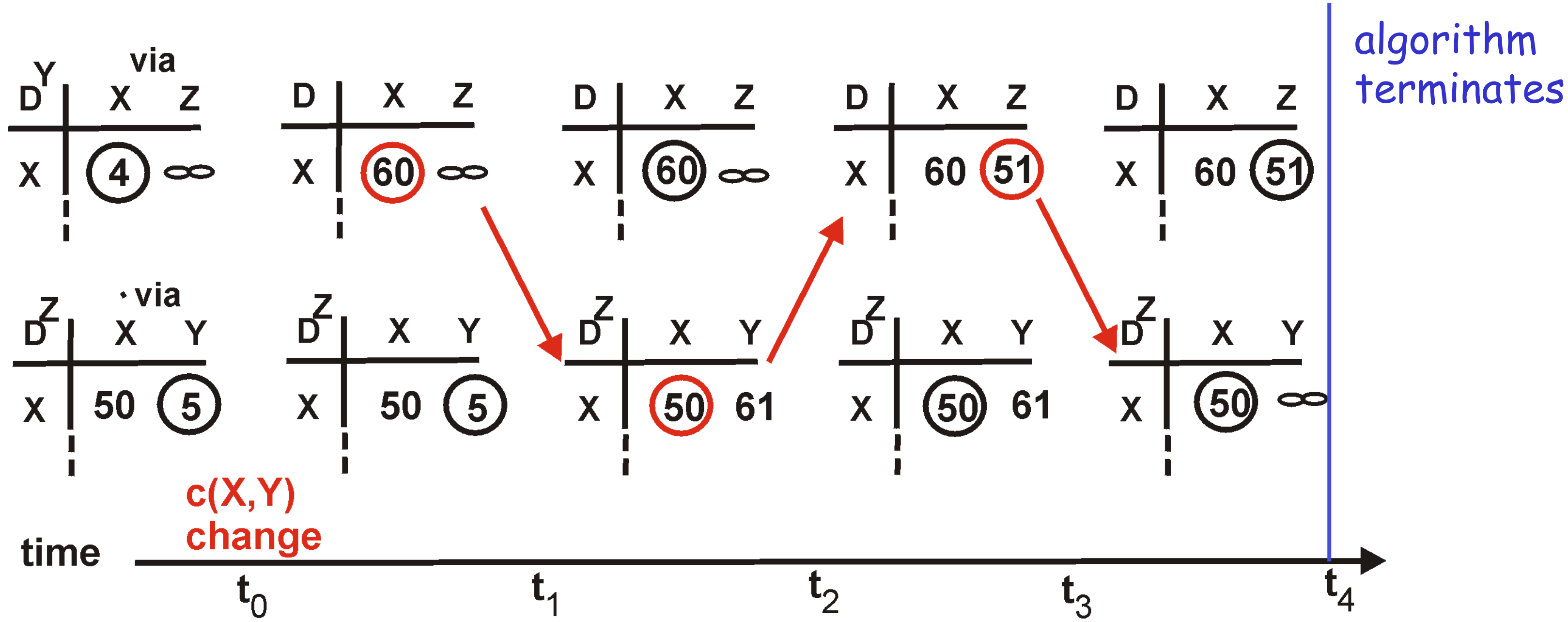
VERY WRONG!! Why is it so? I didn't know that the route from Colosseo to P.Popolo was going through Via del Corso from Al.Patria to P.Popolo (which is closed)!!

Distance Vector: poisoned reverse



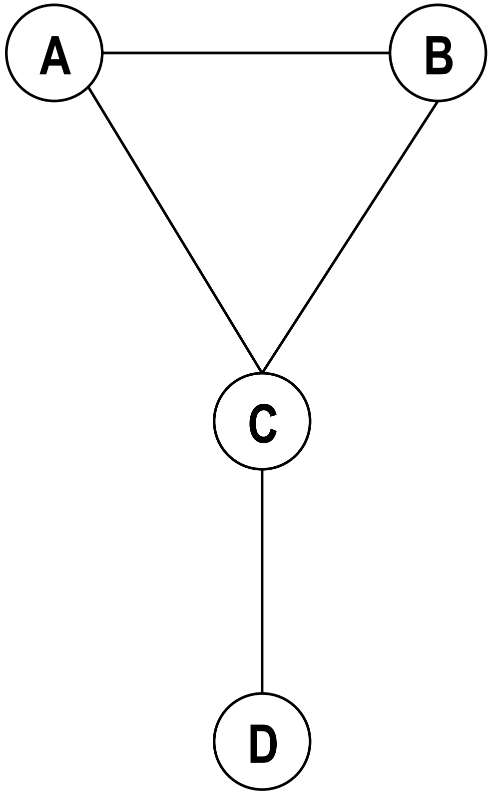
If Z routes through Y to get to X :

- 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?



Infinity is advertized by Y (poisoned reverse)

Split horizon poison reverse failure



Line CD goes down…

1) because of split horizon rule,
   A and B tell C that dist(D)=inf
2) C concludes that D is unreachable
   and reports this to A and B
3) but A knows from B that dist(D)=2, and
   sets its dist=3
4) similarly, B knows from A distance from D…

C estimates new value 4; A and B again through C estimate a value of 5….then again 1)
… etc until distance = infinite

Regardless the hack used, there is always a network topology that makes the trick fail!