Corso di Laurea in Ingegneria Informatica

Corso di Reti di Calcolatori (studenti A-I)

Prof. Roberto Canonico

Routing Distance Vector

I lucidi presentati al corso sono uno strumento didattico che NON sostituisce i testi indicati nel programma del corso

Nota di copyright per le slide COMICS

Autori:
Simon Pietro Romano, Antonio Pescapè, Stefano Avallone, Marcello Esposito, Roberto Canonico, Giorgio Ventre

Nota: alcune delle slide di questa lezione sono direttamente prese dal materiale didattico preparato dagli autori del libro di testo Kurose e Ross

Algoritmo di routing Distance Vector

- Ogni router mantiene una tabella di tutti gli instradamenti noti
  - inizialmente, solo le reti a cui è connesso direttamente
- Ogni entry della tabella indica:
  - una rete raggiungibile
  - il *next hop*
  - il numero di hop necessari per raggiungere la destinazione
- Periodicamente, ogni router invia a tutti i vicini (due router sono vicini se sono collegati alla stessa rete fisica):
  - un messaggio di aggiornamento contenente tutte le informazioni della propria tabella (***vettore delle distanze – distance vector***)
- I router che ricevono tale messaggio aggiornano la tabella nel seguente modo:
  - eventuale modifica di informazioni relative a cammini già noti
  - eventuale aggiunta di nuovi cammini
  - eventuale eliminazione di cammini non più disponibili

Distance Vector: un esempio

Destin.	Dist.	Route
net 1	0	direct
net 2	0	direct
net 4	8	router L
net 17	5	router M
net 24	6	router A
net 30	2	router Q
net 42	2	router A

Tabella del router B

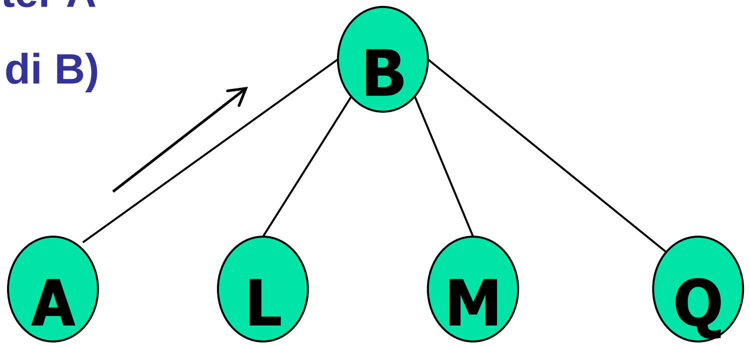
Destin.	Dist.
net 1	2
net 4	3
net 17	6
net 21	4
net 24	5
net 30	10
net 42	3

Messaggio di aggiornamento del router A

(vicino di B)

Destin.	Dist.	Route
net 1	0	direct
net 2	0	direct
net 4	4	router A
net 17	5	router M
net 24	6	router A
net 30	2	router Q
net 42	4	router A
net 21	5	router A

Tabella aggiornata del router B



Distance Vector: elaborazione

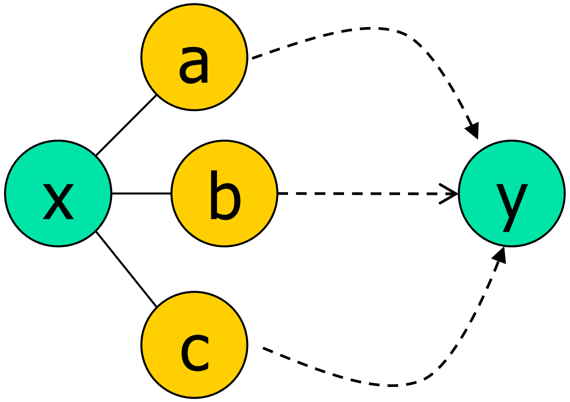
- Il calcolo consiste nella fusione di tutti i distance vector delle linee attive
- Un router ricalcola le sue tabelle se:
  - cade una linea attiva
  - riceve un distance vector, da un nodo adiacente, diverso da quello memorizzato
- Se le tabelle risultano diverse da quelle precedenti:
  - invia ai nodi adiacenti un nuovo distance vector

Equazione di Bellman-Ford

Definito

$d_x(y)$:= costo del percorso a costo minore tra x ed y

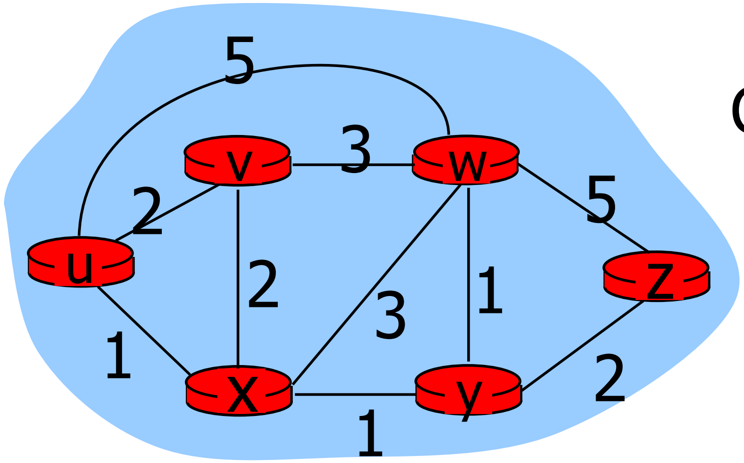
allora



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

dove il minimo è calcolato tra tutti i nodi v adiacenti ad x

Bellman-Ford example



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

B-F equation says:

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

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

Distance Vector Algorithm

- Define
  - $D_x(y)$ = estimate of least cost from x to y
  - Distance vector: $\mathbf{D}_x = [D_x(y): y \in N]$

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

Distance Vector Algorithm

Basic idea:

- Each node periodically sends its own distance vector estimate to neighbors
- When a node x receives new DV estimate from neighbor, it updates its own DV using B-F equation:

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

- **Under "natural conditions" the estimate $D_x(y)$ converges to the actual least cost $d_x(y)$**

Distance Vector: ricapitolando…

Iterativo, asincrono: ogni iterazione locale è causata da:

- Cambiamento di costo di un collegamento
- Messaggi dai vicini

Distribuito: ogni nodo contatta i vicini *solo* quando un suo cammino di costo minimo cambia

- i vicini, a loro volta, contattano i propri vicini se necessario

Ogni nodo:

aspetta notifica modifica costo da un vicino

ricalcola distance table

se il cammino meno costoso verso una qualunque destinazione e' cambiato, *allora* invia *notifica* ai vicini

Distance Vector: l'algoritmo (1/2)

Ad ogni nodo, x:

Inizializzazione:
per tutti i nodi adiacenti v:
$D^X(*,v)$ = *infinito* {*il simbolo* * *significa "per ogni riga"* }
$D^X(v,v) = c(x,v)$
per tutte le destinazioni, y
manda $\min_W D(y,w)$ *a ogni vicino*

Distance Vector : algoritmo (2/2)

loop
aspetta (fino a quando vedo una modifica nel costo di un
collegamento oppure ricevo un messaggio da un vicino v)

if (c(x,v) cambia di d)
{ cambia il costo a tutte le dest. via vicino v di d }
{ nota: d puo' essere positivo o negativo }
per tutte le destinazioni y: $D^X(y,v) = D^X(y,v) + d$

else if (ricevo mess. aggiornamento da v verso destinazione y)
{ cammino minimo da v a y e' cambiato }
{ V ha mandato un nuovo valore per il suo $\min_W D^V(y,w)$ }
{ chiama questo valore "newval" }
per la singola destinazione y: $D^X(y,v) = c(x,v) + newval$

if hai un nuovo $\min_W D^X(y,w)$ per una qualunque destinazione y
manda il nuovo valore di $\min_W D^X(y,w)$ a tutti i vicini

forever

Distance Vector: esempio

node x table

cost to

from	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

cost to

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

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

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

node y table

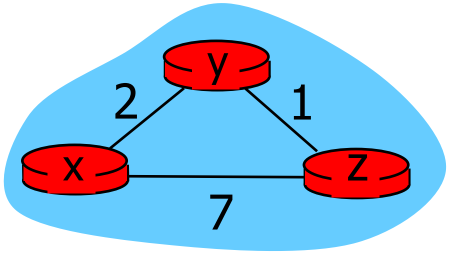
cost to

from	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

node z table

cost to

from	x	y	z
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	∞	∞	∞

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	3
from y	2	0	1
from z	3	1	0

node y table

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

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

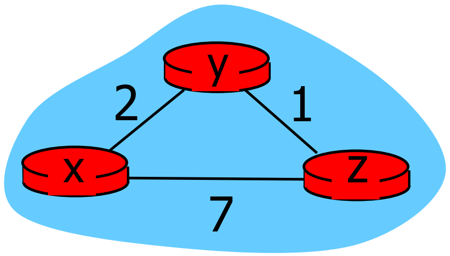
cost to	x	y	z
from x	0	2	3
from y	2	0	1
from z	3	1	0

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	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



time

Distance Vector: analisi

- Vantaggi:
  - facile da implementare
- Svantaggi
  - ogni messaggio contiene un'intera tabella di routing
  - lenta propagazione delle informazioni sui cammini:
    - converge alla velocità del router più lento
  - se lo stato della rete cambia velocemente, le rotte possono risultare inconsistenti
    - possono innescarsi dei loop a causa di particolari variazioni della topologia
  - difficile capirne e prevederne il comportamento su reti grandi
    - nessun nodo ha una mappa della rete!

Convergence Speed

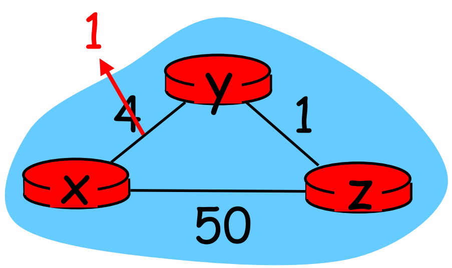
- How fast the routers learn about link-status change in the network

- With distance vector routing
  - Good news travels fast
  - Bad news travels slow

Distance Vector: link cost changes (1/2)

Link cost decreased:

- node detects local link cost change
- updates routing info, recalculates distance vector
- if DV changes, notify neighbors



At time t_0, *y* detects the link-cost change, updates its DV, and informs its neighbors.

"good news travels fast"

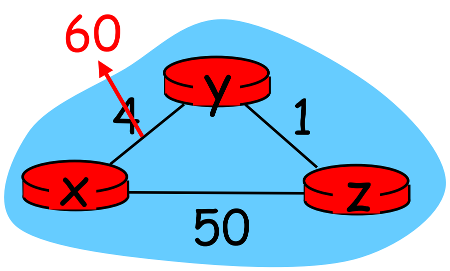
At time t_1, *z* receives the update from *y* and updates its table. It computes a new least cost to *x* and sends its neighbors its DV.

At time t_2, *y* receives *z*'s update and updates its distance table. *y*'s least costs do not change and hence *y* does *not* send any message to *z*.

Distance Vector: link cost changes (2/2)

Link cost <u>increased</u>:

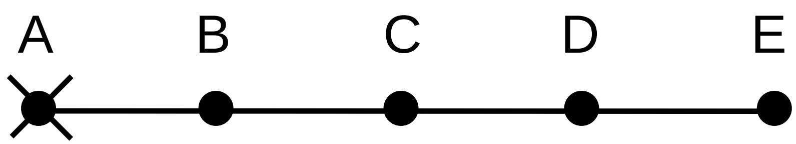
- t_0: y detects change, updates its cost to x to be **6**. Why?
  - Because z previously told y that "I can reach x with cost of 5"
  - **6 = min {60+0, 1+5}**
- Now we have a **routing loop!**
  - Pkts destined to x from y go back and forth between y and z forever (or until loop is broken)
- t_1: z gets the update from y. z updates its cost to x to be??
  - **7 = min {50+0, 1+6}**
- Algorithm will take several iterations to stabilize
- **Solutions?**



"Bad news travels slow"

Count-to-Infinity

1	2	3	4	Initially



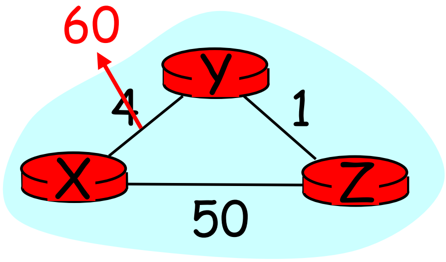
3	2	3	4	After 1 exchange
3	4	3	4	After 2 exchanges
5	4	5	4	After 3 exchanges
5	6	5	6	After 4 exchanges
7	6	7	6	After 5 exchanges

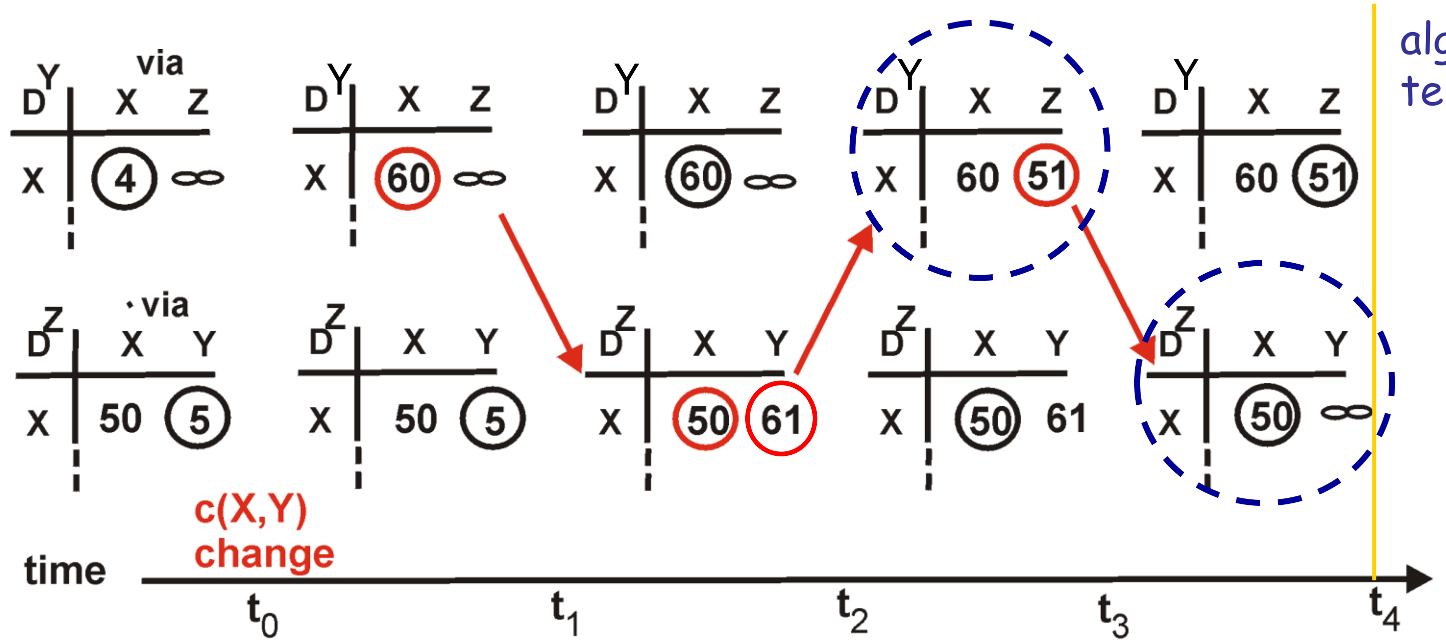
etc… to infinity

Distance Vector: *poisoned reverse*

Se z raggiunge x tramite y:

- z dice a y che la sua distanza per x è infinita (così y non andrà a x attraverso z)
- Viene risolto completamente il problema?

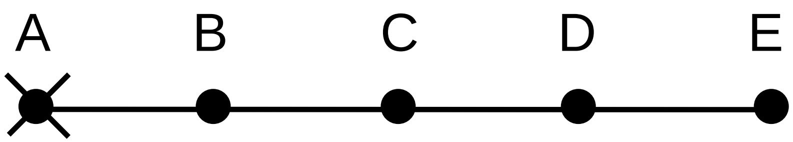




Poisoned Reverse

- 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)

Poisoned Reverse



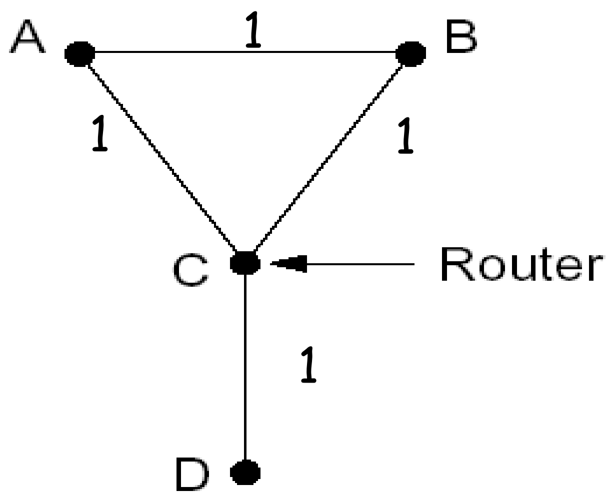
inf.	2	3	4	B learns A is dead
inf. $\xrightarrow{\text{inf.}}$	2	3	4	B reports to C that A's metric is inf.
inf.	inf.	3	4	After 1 exchange
inf.	inf.	inf.	4	After 2 exchanges
inf.	inf.	inf.	inf.	After 3 exchanges

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?

Un esempio in cui il Poisoned Reverse fallisce



- Quando il link tra C e D si interrompe, C “setterà” la sua distanza da D ad ∞
- Però, A userà B per andare a D e B userà A per andare a D.
- Dopo questi update, sia A che B riporteranno un nuovo percorso da C a D (diverso da ∞)