Corso di Laurea in Ingegneria Informatica

Corso di Reti di Calcolatori
(a.a. 2010/11)

Roberto Canonico (roberto.canonico@unina.it)
Giorgio Ventre (giorgio.ventre@unina.it)

Routing Distance Vector
Routing Link State

16 novembre 2010

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

Nota di Copyright

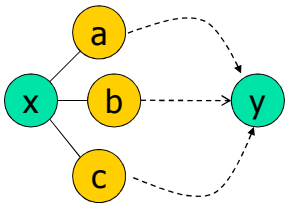
Equazione di Bellman-Ford

Definito

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

allora

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



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

Algoritmo Distance Vector (Bellman-Ford)

- Ogni nodo:
  - Invia ai nodi adiacenti un distance vector, costituito da:
    - insieme di coppie *(indirizzo,distanza)*, dove la distanza è espressa tramite metriche classiche, quali numero di hop e costo
  - Memorizza per ogni linea l'ultimo distance vector ricevuto.
  - Calcola le proprie tabelle di instradamento.
  - Se le tabelle risultano diverse da quelle precedenti:
    - invia ai nodi adiacenti un nuovo distance vector

4

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
- Vantaggi:
  - Molto semplice da implementare
- Svantaggi
  - Possono innescarsi dei loop a causa di particolari variazioni della topologia
  - Converge alla velocità del link più lento e del router più lento
  - Difficile capirne e prevederne il comportamento su reti grandi
    - nessun nodo ha una mappa della rete!

5

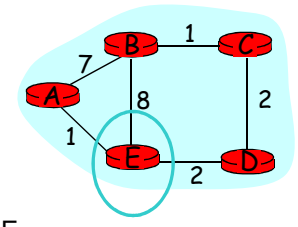
Distance Vector: caratteristiche

- Iterativo:
  - continua fino a quando non c'è più scambio di informazioni
  - *self-terminating*: non c'è un esplicito segnale di stop
- Asincrono
- Distribuito:
  - ogni nodo comunica con i diretti vicini
- Struttura Distance Table
  - ogni nodo ha la sua tabella delle distanze:
    - una riga per ogni destinazione
    - una colonna per ogni nodo adiacente
- notazione:

$$D^{X}(Y,Z) = \begin{array}{l}\text{distanza } \textit{da } X \textit{ a } Y,\\ \textit{via } Z \text{ (prossimo hop)}\end{array} = c(X,Z) + \min_{w}\{D^{Z}(Y,w)\}$$

6

Distance Vector: un esempio



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

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

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

costo per la destinazione via

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

destinazione

7

Distance Vector: *distance table* e *routing table*

costo a destinazione via

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

destinazione

Distance table di E

	link uscita da usare	costo
A	A	1
B	D	5
C	D	4
D	D	2

destinazione

Routing table di E

8

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

9

Distance Vector: l'algoritmo (1/2)

Ad ogni nodo, x:

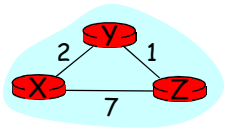
```
1  Inizializzazione:
2   per tutti i nodi adiacenti  v:
3      D^X(*,v) = infinito        {il simbolo  * significa "per ogni riga" }
4      D^X(v,v) = c(x,v)
5   per tutte le destinazioni, y
6      manda  min_W D(y,w) a ogni vicino
```

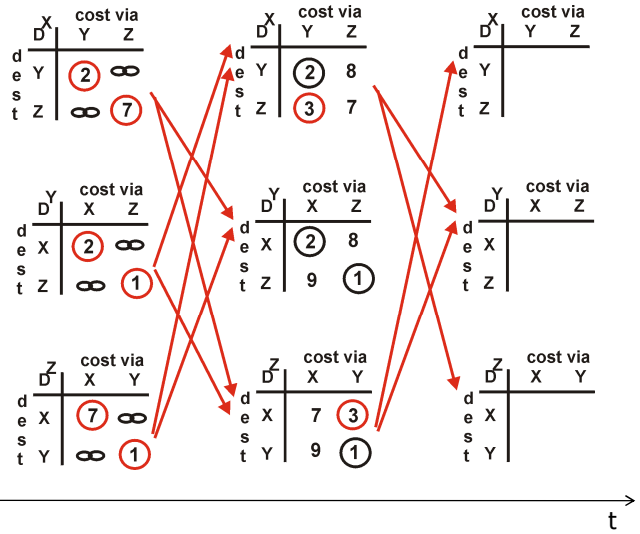
10

Distance Vector : algoritmo (2/2)

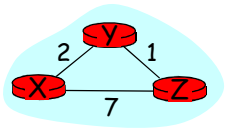
```
8  loop
9   aspetta  (fino a  quando vedo una modifica nel costo di un
10          collegamento  oppure ricevo un messaggio da un vicino v)
11
12  if (c(x,v) cambia di  d)
13   { cambia il costo a tutte le dest. via vicino v di d }
14   { nota: d puo' essere  positivo o  negativo  }
15   per tutte le  destinazioni  y:  D^X(y,v) =  D^X(y,v) + d
16
17  else if (ricevo mess. aggiornamento da v verso destinazione y)
18   { cammino minimo da v a y e' cambiato }
19   { V ha mandato un nuovo valore per il suo  min_W D^V(y,w) }
20   { chiama questo valore  "newval"  }
21   per la singola destinazione y: D^X(y,v) = c(x,v) + newval
22
23  if hai un nuovo  min_W D^X(y,w) per una qualunque destinazione  y
24    manda il nuovo valore di  min_W D^X(y,w) a tutti i vicini
25
26  forever
```


Distance Vector : esempio completo (1/2)





Distance Vector : esempio completo (2/2)



D^X

dest	via Y	via Z
Y	2	∞
Z	∞	7

D^X

dest	via Y	via Z
Y	2	8
Z	3	7

D^Y

dest	via X	via Z
X	2	∞
Z	∞	1

D^Z

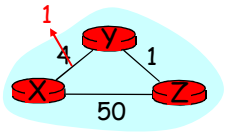
dest	via X	via Y
X	7	∞
Y	∞	1

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

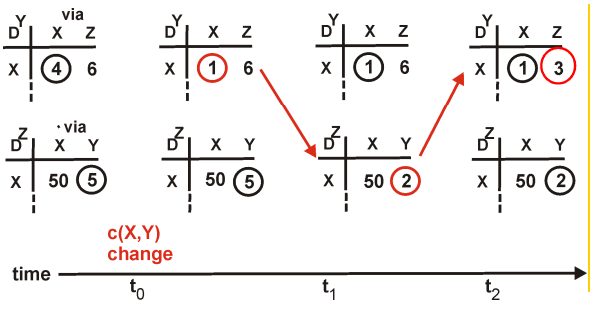
$$D^X(Z,Y) = c(X,Y) + \min_w\{D^Y(Z,w)\}$$
$$= 2+1 = 3$$

Distance Vector: modifica dei costi dei collegamenti (1/2)

- Un nodo si accorge di una modifica locale al costo di un link ad esso connesso
- Aggiorna la sua distance table (linea 15 algoritmo)
- Se cambia il costo di qualche path allora lo notifica ai vicini (linee 23,24 algoritmo)



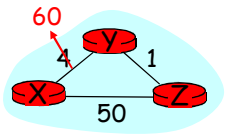
"le buone notizie" viaggiano veloci

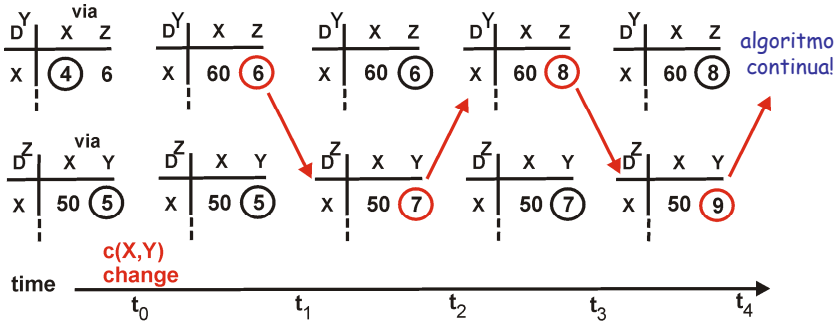


Distance Vector: modifica dei costi dei collegamenti (2/2)

Cambiamenti nei costi: le "buone notizie" viaggiano in fretta, le cattive lentamente…

Problema: "conteggio all'infinito"!



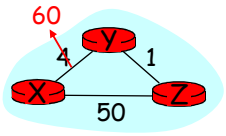


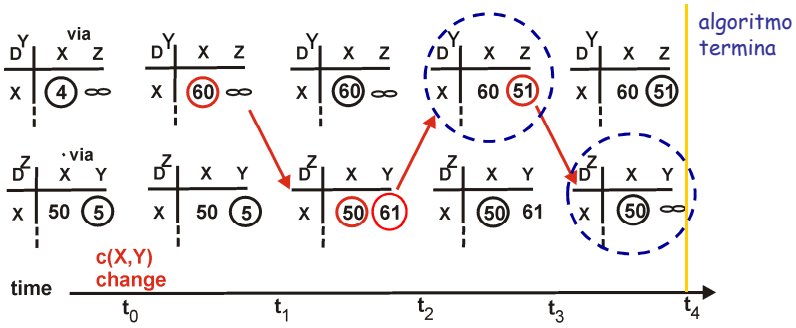
15

Distance Vector: *poisoned reverse*

Se z raggiunge x tramite y:

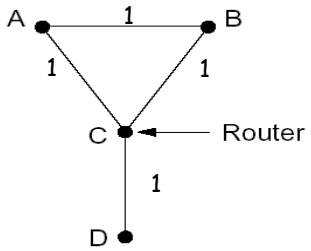
- z dice a y che la sua distanza per x è infinita (così y non andrà a x attraverso z)
- Viene risolto completamente il problema?





16

Un esempio in cui lo *split horizon* 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 ∞)

17

Link State

- Ogni router:
  - impara il suo ambito locale (linee e nodi adiacenti)
  - trasmette queste informazioni a tutti gli altri router della rete tramite un *Link State Packet* (LSP)
  - memorizza gli LSP trasmessi dagli altri router e costruisce una mappa della rete
  - Calcola, in maniera indipendente, le sue tabelle di instradamento applicando alla mappa della rete l'algoritmo di Dijkstra, noto come *Shortest Path First* (SPF)
- Tale approccio è utilizzato nello standard ISO 10589 (protocollo IS-IS) e nel protocollo OSPF (adottato in reti TCP/IP)

18

Il processo di *update*

- Ogni router genera un Link State Packet (LSP) contenente:
  - stato di ogni link connesso al router
  - identità di ogni vicino connesso all'altro estremo del link
  - costo del link
  - numero di sequenza per l'LSP
  - checksum
  - Lifetime:
    - la validità di ogni LSP è limitata nel tempo (e.g. un errore sul numero di sequenza potrebbe rendere un LSP valido per anni)

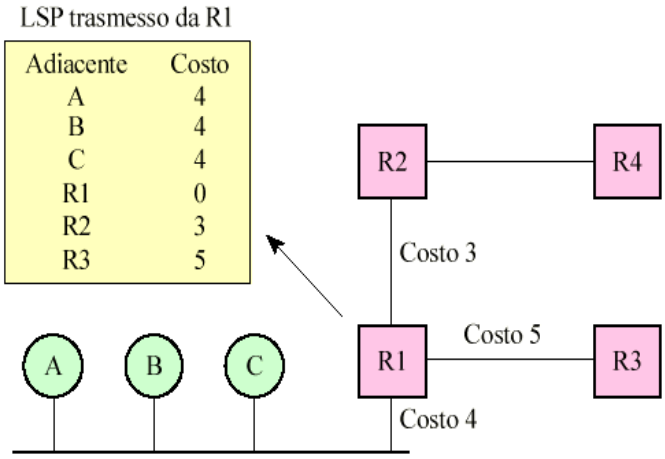
19

LSP flooding

- Un LSP viene generato periodicamente, oppure quando viene rilevata una variazione nella topologia locale (adiacenze), ossia :
  - Viene riconosciuto un nuovo vicino
  - Il costo verso un vicino e' cambiato
  - Si e' persa la connettività verso un vicino precedentemente raggiungibile
- Un LSP è trasmesso in flooding su tutti i link del router
- I pacchetti LSP memorizzati nei router formano una mappa completa e aggiornata della rete:
  - *Link State Database*

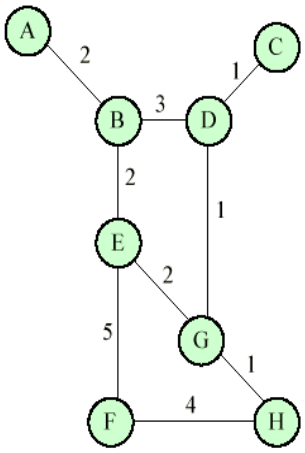
20

Esempio: trasmissione di un LSP



21

Esempio: grafo della rete e LSP-DB



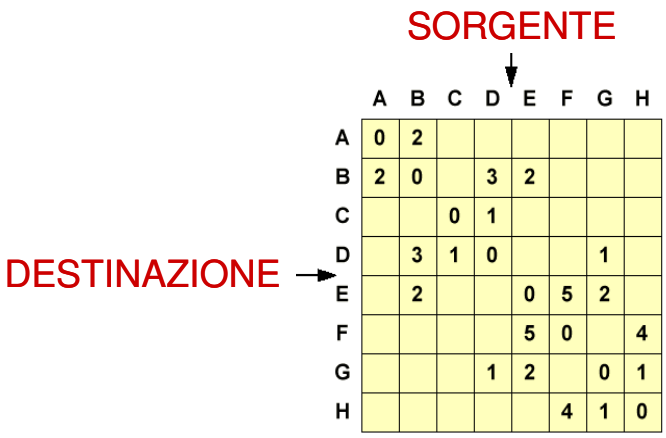
LSP Database

A	B/2		
B	A/2	D/3	E/2
C	D/1		
D	B/3	C/1	G/1
E	B/2	F/5	G/2
F	E/5	H/4	
G	D/1	E/2	H/1
H	F/4	G/1	

(replicato su ogni IS)

22

LSP database



Questa rappresentazione è quella più appropriata per applicare l'agoritmo di Dijkstra

23

Gestione degli LSP

- All'atto della ricezione di un LSP, il router compie le seguenti azioni:
  1. se non ha mai ricevuto LSP da quel router o se l'LSP è più recente di quello precedentemente memorizzato:
     - memorizza il pacchetto
     - lo ritrasmette in flooding su tutte le linee eccetto quella da cui l'ha ricevuto
  2. se l'LSP ha lo stesso numero di sequenza di quello posseduto:
     - non fa nulla
  3. Se l'LSP è più vecchio di quello posseduto:
     - trasmette al mittente il pacchetto più recente

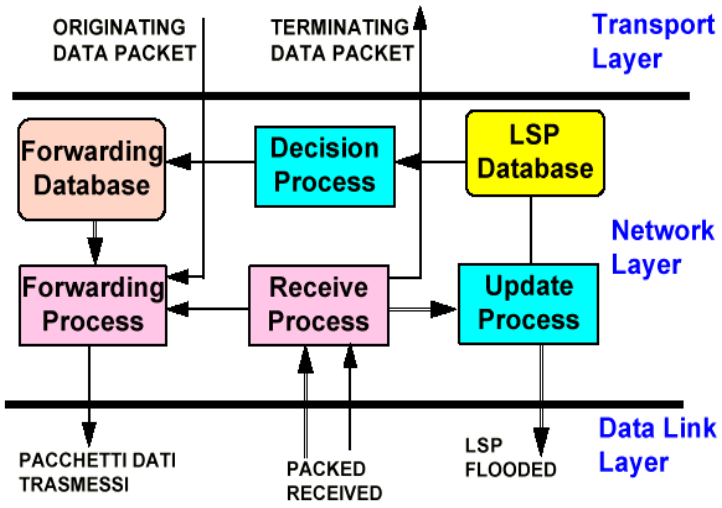
24

Routing: decisioni

- Il router elabora il *Link State Database* per produrre il Forwarding Database:
  - si pone come radice dello shortest-path tree
  - cerca lo shortest path per ogni nodo destinazione
  - memorizza il vicino (i vicini) che sono sullo shortest path verso ogni nodo destinazione
- Il *Forwarding Database* contiene, per ogni nodo destinazione:
  - l'insieme delle coppie {path, vicino}
  - la dimensione di tale insieme

25

Architettura di un router Link State



26

Link State: caratteristiche

- Vantaggi:
  - può gestire reti di grandi dimensioni
  - ha una convergenza rapida
  - difficilmente genera loop, e comunque è in grado di identificarli ed interromperli facilmente
  - facile da capire: ogni nodo ha la mappa della rete
- Svantaggi:
  - Molto complesso da realizzare:
    - Es: la prima implementazione ha richiesto alla Digital 5 anni

27

Esempio: tabelle di instradamento

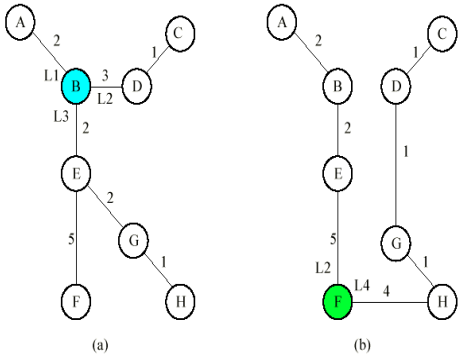


Tabella di B

A	L1
C	L2
D	L2
E	L3
F	L3
G	L3
H	L3

Tabella di F

A	L2
B	L2
C	L4
D	L4
E	L2
G	L4
H	L4

28