



Corso di Laurea in Ingegneria Informatica

Corso di Reti di Calcolatori

Docente: Simon Pietro Romano
spromano@unina.it

Routing

—

Parte seconda: algoritmi Distance Vector e Link State



Algoritmo Distance Vector

- Algoritmo di *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



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!**

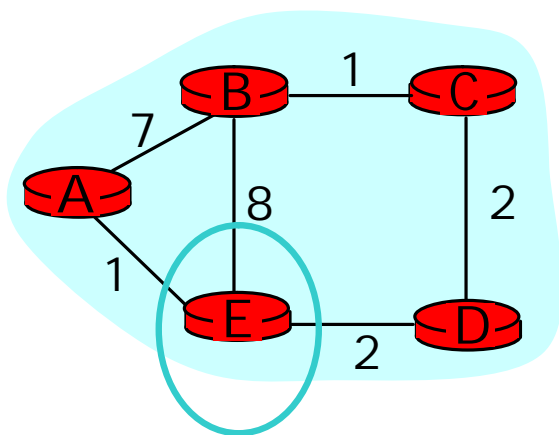


Distance Vector: caratteristiche

- **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 adoperata:
$$D^X(Y,Z) = \text{distanza da } X \text{ a } Y, \text{ via } Z \text{ (prossimo hop)}$$
$$= c(X,Z) + \min_w \{D^Z(Y,w)\}$$



Distance Vector: un esempio



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

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

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

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



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

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

destinazione

Distance table \longrightarrow Routing table



Distance Vector: ricapitolando...

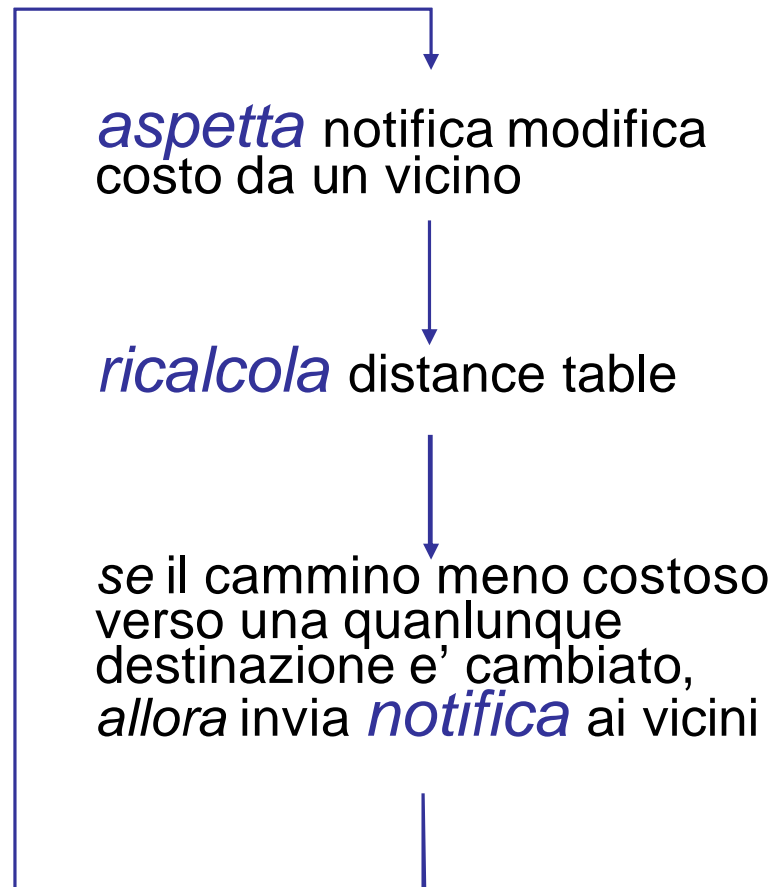
Iterativo, asincrono: ogni iterazione locale è causata da:

- 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:





Distance Vector: l' algoritmo (1/2)

Ad ogni nodo, x :

1 *Inizializzazione:*

2 *per tutti i nodi adiacenti v :*

3 $D^x(*,v) = \text{infinito}$ {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*

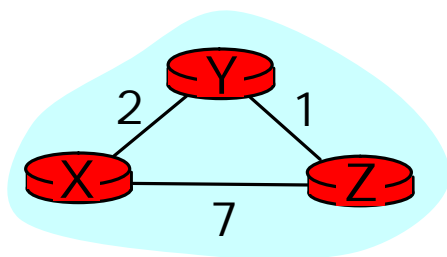


Distance Vector : algoritmo (2/2)

```
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:  $D^X(y,v) = c(x,v) + \text{newval}$ 
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)



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

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

		cost via	
		Y	Z
dest	D ^X		
	Y		
	Z		

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

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

		cost via	
		X	Z
dest	D ^Y		
	X		
	Z		

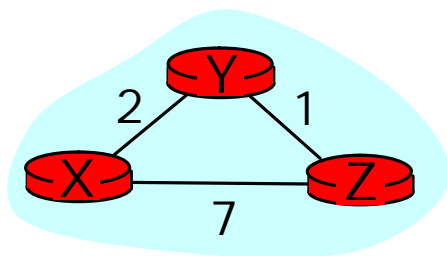
		cost via	
		X	Y
dest	D ^Z		
	X	7	∞
	Y	∞	1

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

		cost via	
		X	Y
dest	D ^Z		
	X		
	Y		



Distance Vector : esempio completo (2/2)



		cost via	
		Y	Z
d e s t	D ^X	2	∞
	Z	∞	7

		cost via	
		X	Z
d e s t	D ^Y	2	∞
	Z	∞	1

		cost via	
		X	Y
d e s t	D ^Z	7	∞
	Y	∞	1

		cost via	
		Y	Z
d e s t	D ^X	2	8
	Z	3	7

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

$$= 7 + 1 = 8$$

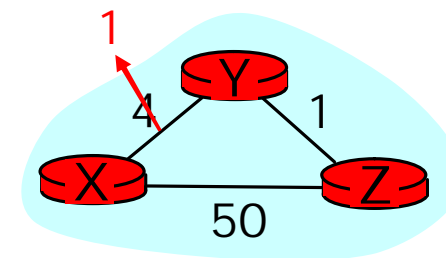
$$D^X(Z,Y) = c(X,Y) + \min_w \{D^Y(Z,w)\}$$

$$= 2 + 1 = 3$$

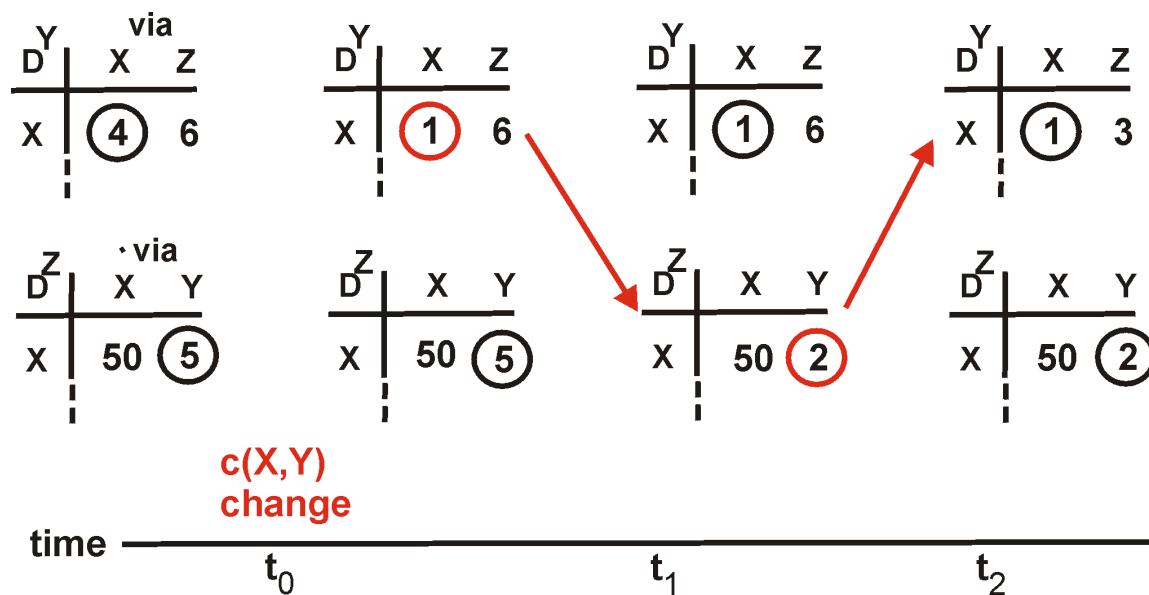


Distance Vector : modifica dei costi dei collegamenti (1/2)

- 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

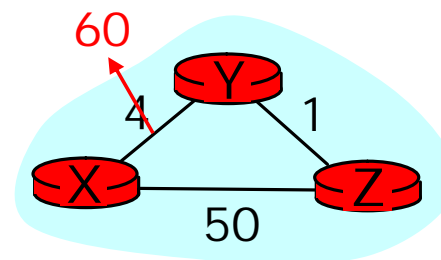


algoritmo
termina

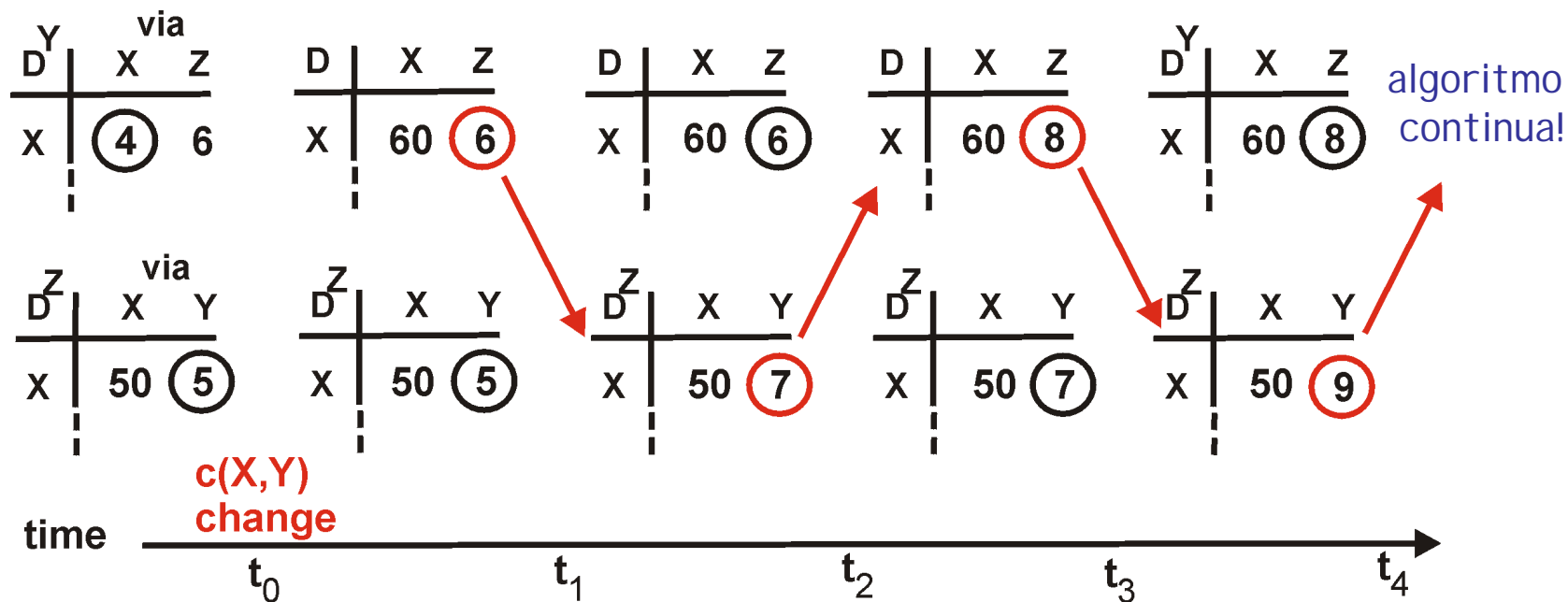


Distance Vector: modifica dei costi dei collegamenti (2/2)

Cambiamenti nei costi: le “buone notizie” viaggiano in fretta, le cattive lentamente



Problema: “conteggio all’infinito”!

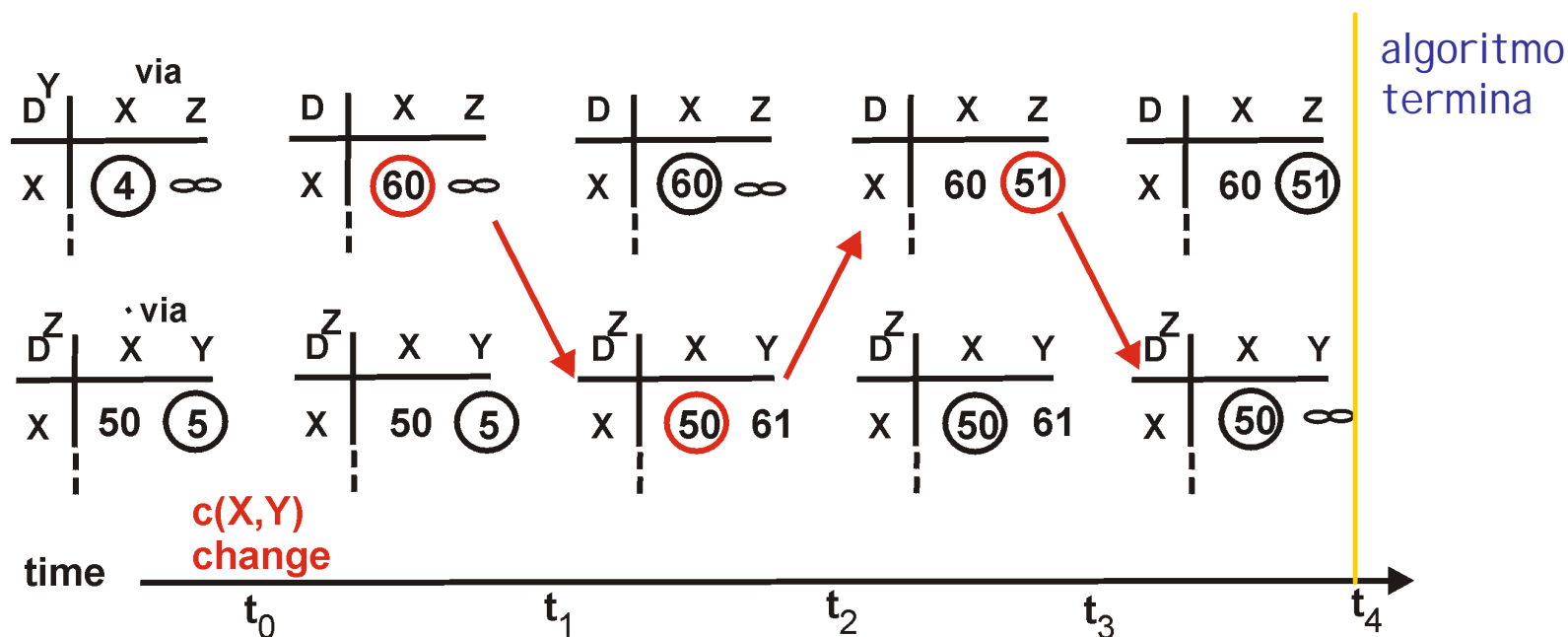
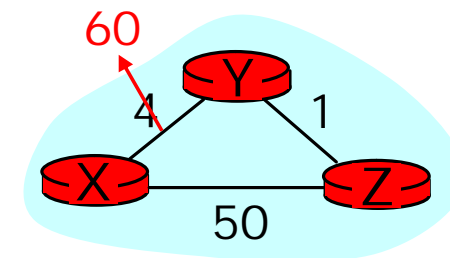




Distance Vector: algoritmo modificato

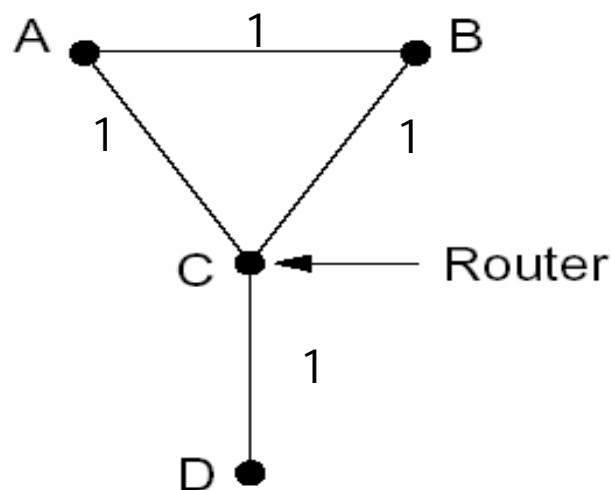
Se z raggiunge x tramite y:

- 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?**





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 ∞)



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)



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



LSP flooding

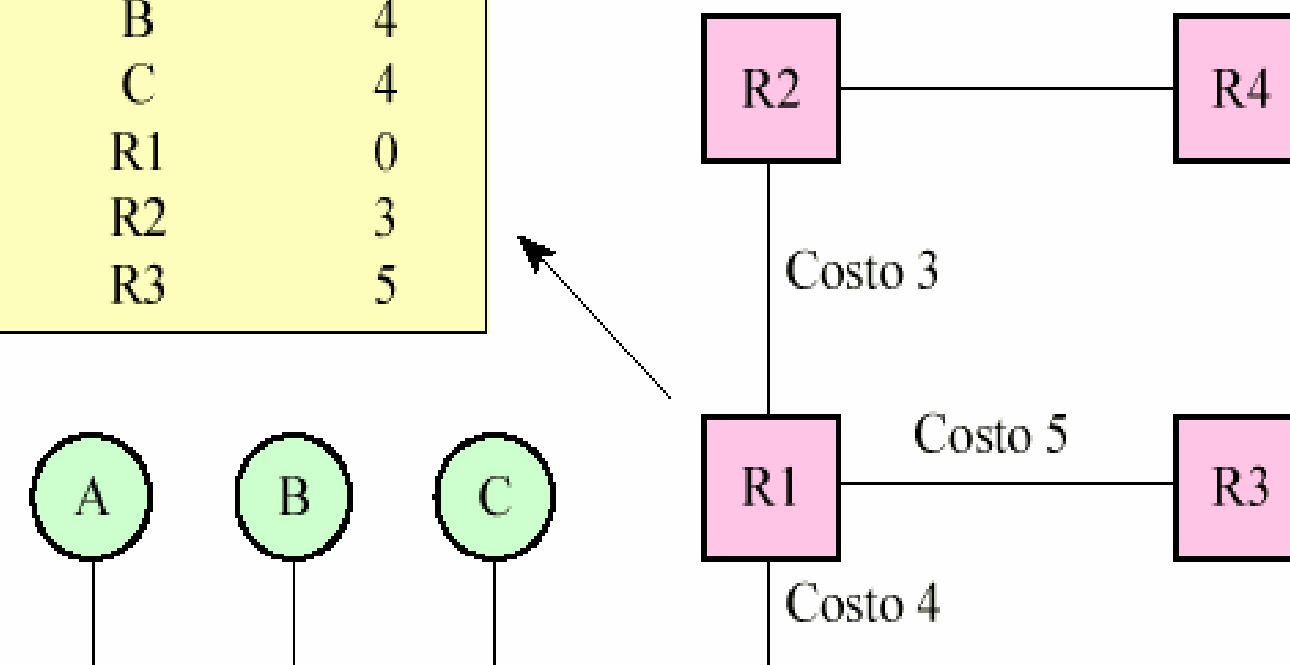
- Un LSP è trasmesso in flooding su tutti i link del router
- I pacchetti LSP memorizzati nei router formano una mappa completa e aggiornata della rete:
 - *Link State Database*



Esempio: trasmissione di un LSP

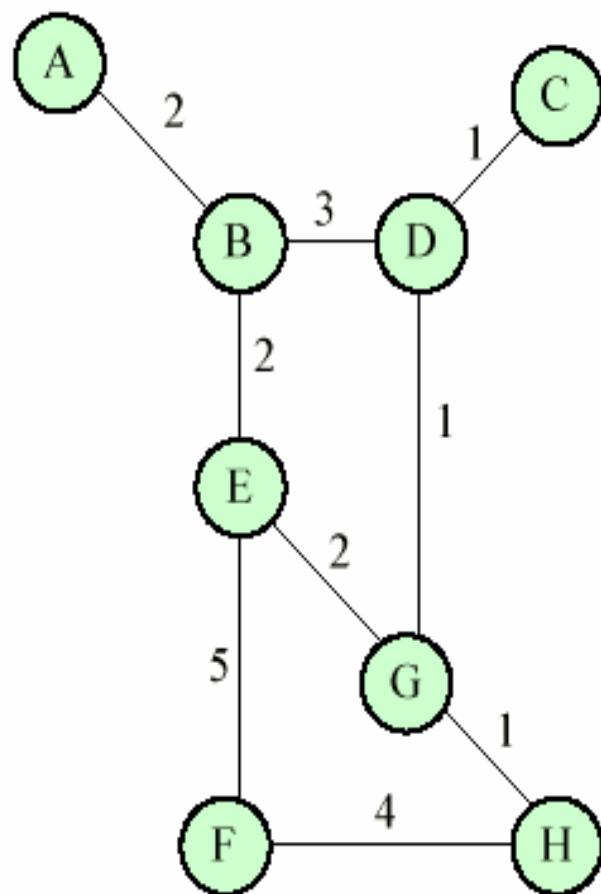
LSP trasmesso da R1

Adiacente	Costo
A	4
B	4
C	4
R1	0
R2	3
R3	5





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)



LSP database

SORGENTE



A B C D E F G H

A	0	2						
B	2	0		3	2			
C			0	1				
D		3	1	0			1	
E		2			0	5	2	
F					5	0		4
G				1	2		0	1
H						4	1	0

DESTINAZIONE



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



Gestione degli LSP

- All'atto della ricezione di un LSP, il router compie le seguenti azioni:
 - 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
 - se l'LSP ha lo stesso numero di sequenza di quello posseduto:
 - non fa nulla
 - Se l'LSP è più vecchio di quello posseduto:
 - trasmette al mittente il pacchetto più recente

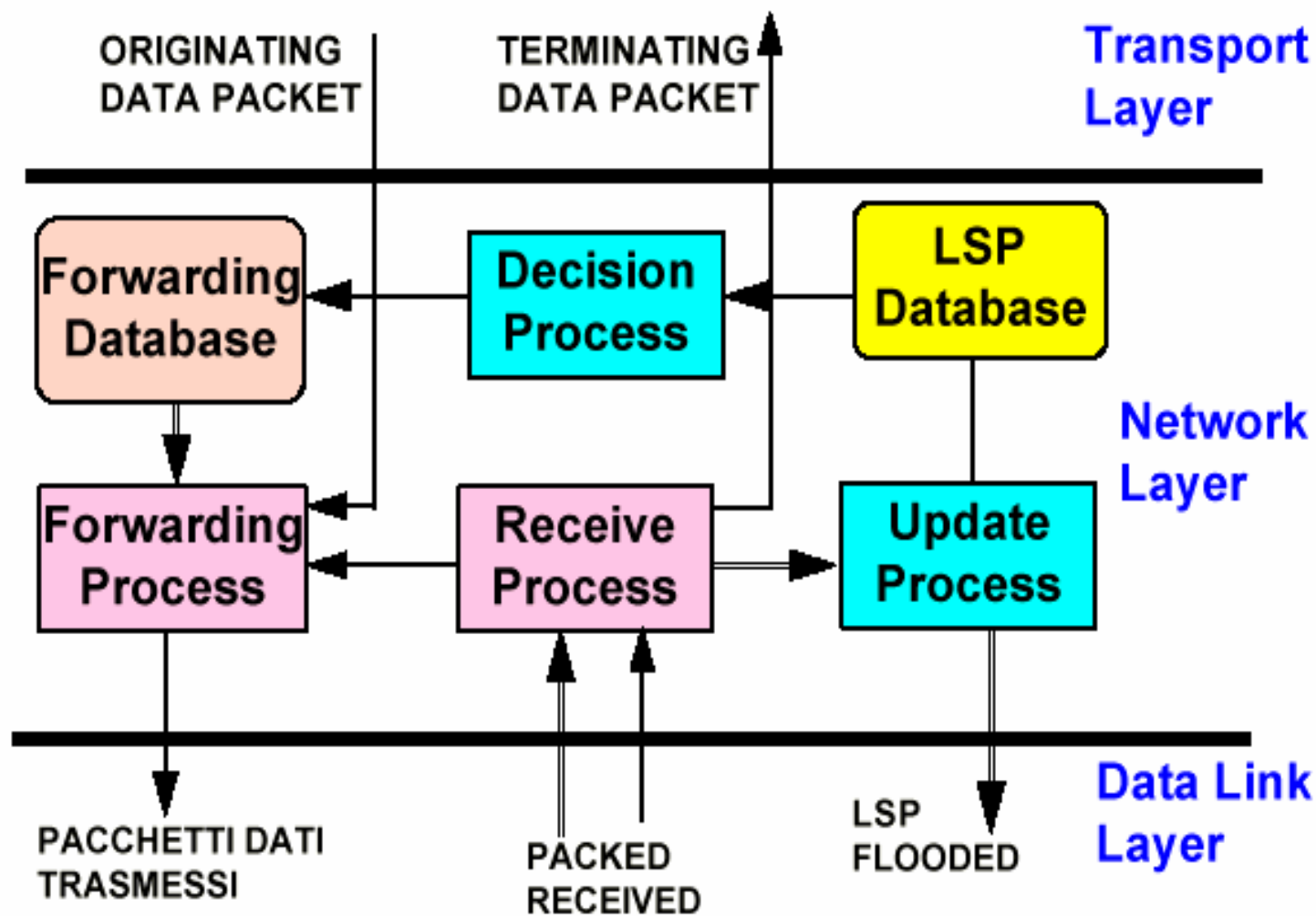


Routing: decisioni

- 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



Architettura di un router Link State



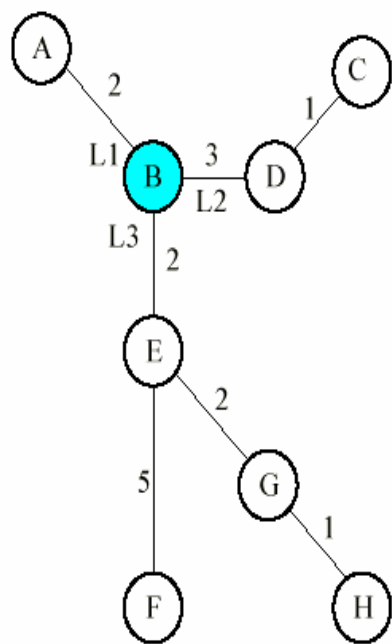


Link State: caratteristiche

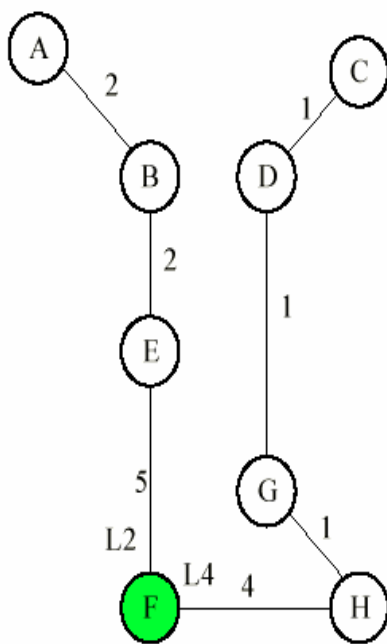
- Vantaggi:
 - può gestire reti di grandi dimensioni
 - ha una convergenza rapida
 - difficilmente genera loop, e comunque è in grado di identificarli ed interromperli facilmente
 - facile da capire: ogni nodo ha la mappa della rete
- Svantaggi:
 - Molto complesso da realizzare:
 - Es: la prima implementazione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



Esempio: tabelle di instradamento



(a)



(b)

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



Algoritmo di Dijkstra

- Ogni nodo ha a disposizione il grafo della rete:
 - i nodi sono i router
 - gli archi sono le linee di collegamento tra router:
 - agli archi è associato un costo
- Ogni nodo usa l'algoritmo di Dijkstra per costruire lo *Spanning Tree* del grafo, ovvero l'albero dei cammini di costo minimo
- Ad ogni nodo si assegna un'etichetta che rappresenta il costo massimo per raggiungere quel nodo
- L'algoritmo modifica le etichette cercando di minimizzarne il valore e di renderle permanenti



Algoritmo di Dijkstra: formalizzazione

- La Topologia della rete è nota a tutti i nodi:
 - la diffusione è realizzata via “link state broadcast”
 - tutti i nodi hanno la stessa informazione
 - Si calcola il percorso minimo da un nodo a tutti gli altri:
 - l'algoritmo fornisce la **tavola di routing** per quel nodo
 - **Iterativo:** un nodo, dopo k iterazioni, conosce i cammini meno costosi verso k destinazioni
- Notazione:**
- **$c(i,j)$** : costo collegamento da i a j:
 - infinito se non c'è collegamento
 - per semplicità, **$c(i,j) = c(j,i)$**
 - **$D(v)$** : costo corrente del percorso, dalla sorgente al nodo v
 - **$p(v)$** : predecessore (collegato a v) lungo il cammino dalla sorgente a v
 - **N** : insieme di nodi per cui la distanza è stata trovata



Algoritmo di Dijkstra (cont.)

1 **Inizializzazione:**

2 $N = \{A\}$

3 per tutti i nodi v

4 if (v e' adiacente a A)

5 then $D(v) = c(A,v)$

6 else $D(v) = \text{infty}$

7

8 **Loop**

9 sia w non in N tale che $D(w)$ è minimo

10 aggiungi w a N

11 aggiorna $D(v)$ per ogni v adiacente a w e non in N :

12 $D(v) = \min(D(v), D(w) + c(w,v))$

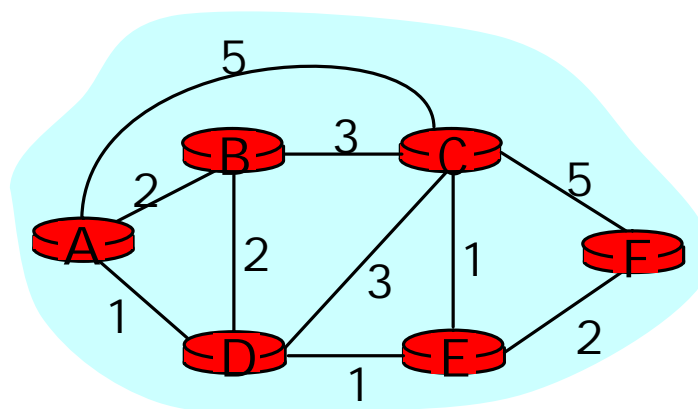
13 {il nuovo costo fino a v è o il vecchio costo, oppure il costo del cammino più breve fino a w più il costo da w a v }

15 **fino a quando tutti i nodi sono in N**



Algoritmo di Dijkstra: interpretazione

- L'algoritmo consiste in un passo di inizializzazione, più un ciclo di durata pari al numero di nodi della rete. Al termine avremo i percorsi più brevi dal nodo sorgente a tutti gli altri nodi
- **Esempio.** Calcoliamo sulla rete data i percorsi di costo minimo da A a tutte le possibili destinazioni. Ciascuna riga della tabella della slide seguente fornisce i valori delle variabili dell'algoritmo alla fine di ciascuna iterazione

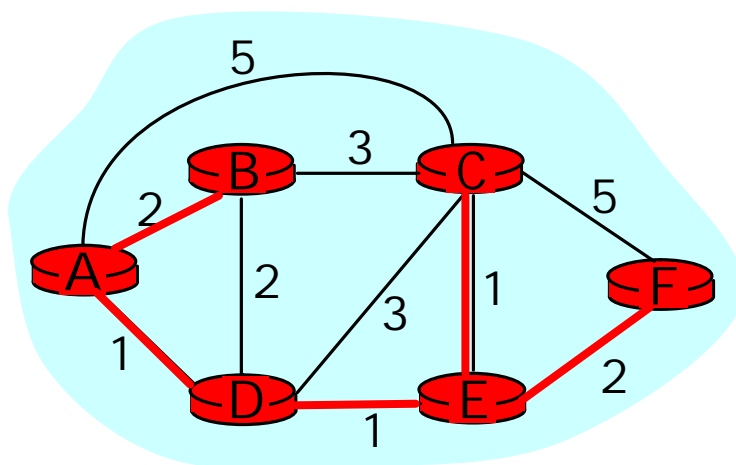




Algoritmo di Dijkstra: esempio

↓

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	infinity	infinity
→ 1	AD	2,A	4,D	2,D	infinity	infinity
→ 2	ADE	2,A	3,E	3,E	4,E	infinity
→ 3	ADEB	2,A	3,E	3,E	4,E	4,E
→ 4	ADEBC	2,A	3,E	3,E	4,E	4,E
5	ADEBCF	2,A	3,E	3,E	4,E	4,E



Notazione:

- $c(i,j)$: costo collegamento da i a j (infinito se non c'è collegamento e per semplicità $c(i,j) = c(j,i)$)
- $D(v)$: costo corrente del percorso, dalla sorgente al nodo v
- $p(v)$: predecessore (collegato a v) lungo il cammino dalla sorgente a v
- N : insieme di nodi per cui la distanza è stata trovata



Algoritmo di Dijkstra: discussione

Se il costo di un link è proporzionale al traffico su quel link, allora sono possibili oscillazioni

