



Fondamenti di Internet e Reti

Antonio Capone, Matteo Cesana,
Ilario Filippini, Guido Maier



4 – Rete (parte 4)

Antonio Capone, Matteo Cesana,
Ilario Filippini, Guido Maier

Agenda

- Instradamento in rete
 - Caratteristiche
 - Algoritmi su grafi
 - Algoritmo di Bellman-Ford
 - Algoritmo di Dijkstra
 - Algoritmi d'instradamento a costo minimo
 - Instradamento Distance Vector
 - Instradamento Link State



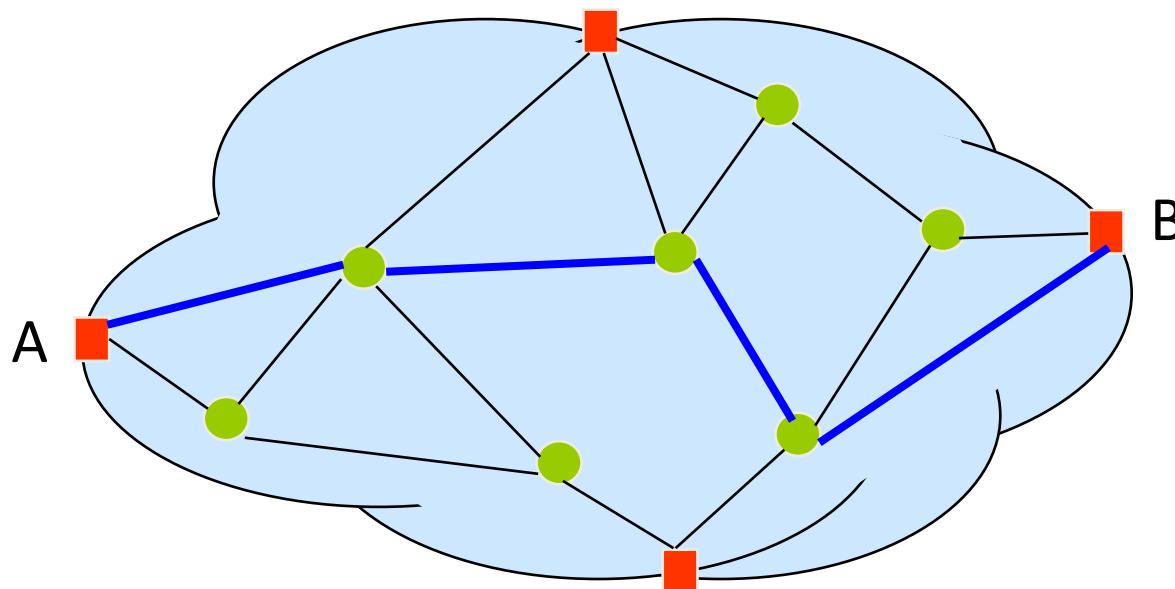
Agenda

- Instradamento in rete
 - Caratteristiche
 - Algoritmi su grafi
 - Algoritmo di Bellman-Ford
 - Algoritmo di Dijkstra
 - Algoritmi d'instradamento a costo minimo
 - Instradamento Distance Vector
 - Instradamento Link State



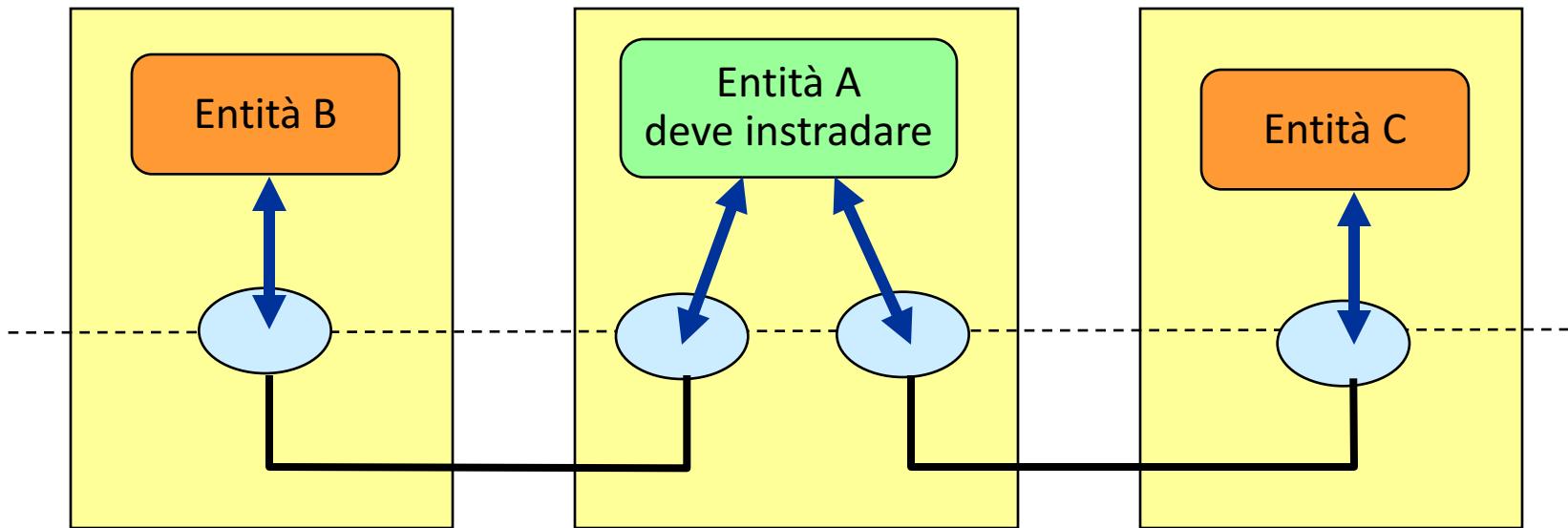
Routing Unicast

- L'instradamento (o *routing*) è alla base della funzionalità di rete implementata dalle entità di livello 3 (OSI) dei nodi
- Consente a due nodi A e B, non collegati direttamente, di comunicare tra loro mediante la collaborazione di altri nodi



Routing Unicast

- Le entità di livello 3 sul cammino basano la commutazione (forwarding) verso il SAP d'uscita sulla base di un indirizzo o di una etichetta posta sul pacchetto
- La corrispondenza tra indirizzo e SAP d'uscita è mantenuta dal nodo in una **tabella di routing**

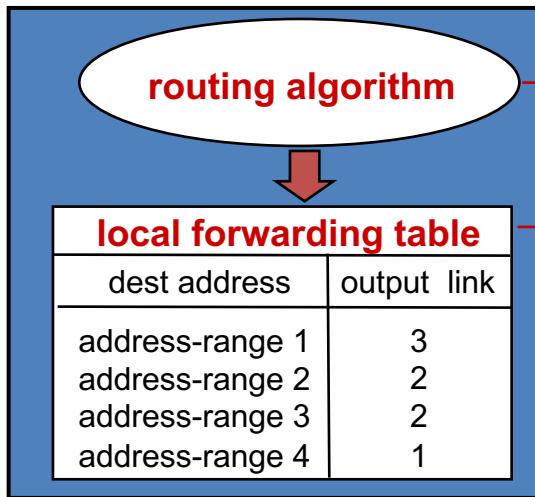


Politiche di Routing

- La **politica di routing** (o *algoritmo di routing*) è quella che definisce i criteri di scelta del cammino nella rete per i pacchetti che viaggiano tra un nodo di ingresso ed uno di uscita
- e dunque quella che costruisce le tabelle di *routing* che vengono usate dai nodi per effettuare il *forwarding*
- Il tipo di rete (*datagram*, circuito virtuale) determina il tipo di tabelle da utilizzare e i gradi di libertà della politica di *routing* nella scelta dei cammini



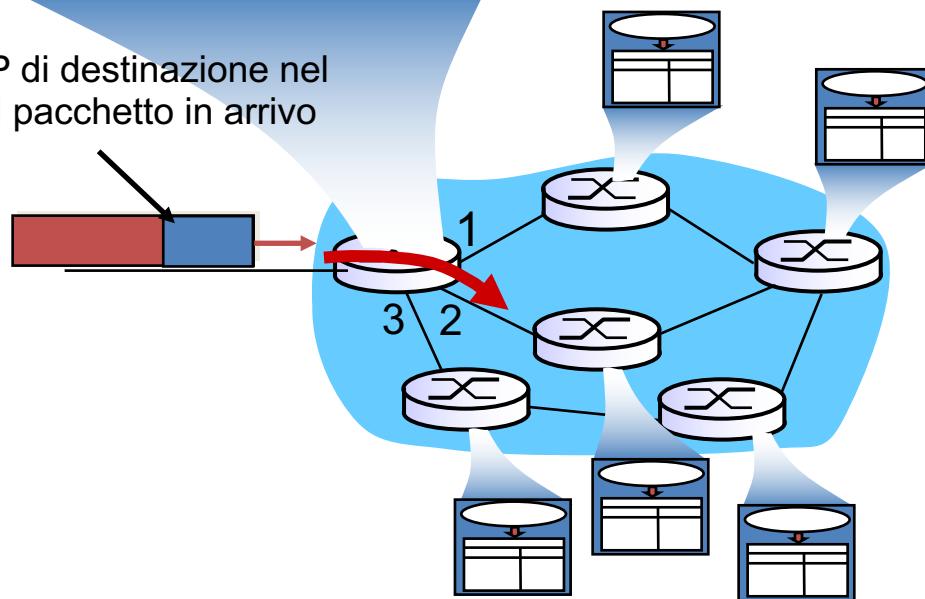
Interazione tra routing e forwarding



L'algoritmo di routing determina il percorso end-end-path attraverso la rete

La tabella di forwarding determina l'inoltro locale a questo router

Indirizzo IP di destinazione nel header del pacchetto in arrivo



Protocolli di Routing

- Con questo nome si indicano in genere due diverse funzionalità, anche se legate fra loro
 - lo scambio fra i *router* di informazioni di raggiungibilità (info sulla topologia di rete, sul traffico etc.)
 - la costruzione delle tabelle di *routing* (scelta del “percorso migliore”)
- Formalmente il protocollo è solo la parte che descrive lo scambio di messaggi tra i *router*
- In realtà questo scambio è poi strettamente legato al modo con cui sono calcolate le tabelle di *routing*



Routing e capacità

- Nelle reti broadcast (come WiFi) non vi sono nodi che effettuano instradamento ed il mezzo condiviso può essere usato a turno
- Il risultato è che il traffico massimo che può essere smaltito dalla rete (**capacità**) è al più pari alla capacità del canale
- Nelle reti magliate la trasmissione di un pacchetto non occupa tutte le risorse di rete e più canali e cammini possono essere usati in parallelo
- E' facile comprendere come in questo caso la politica di instradamento abbia un forte impatto sul traffico smaltibile dalla rete

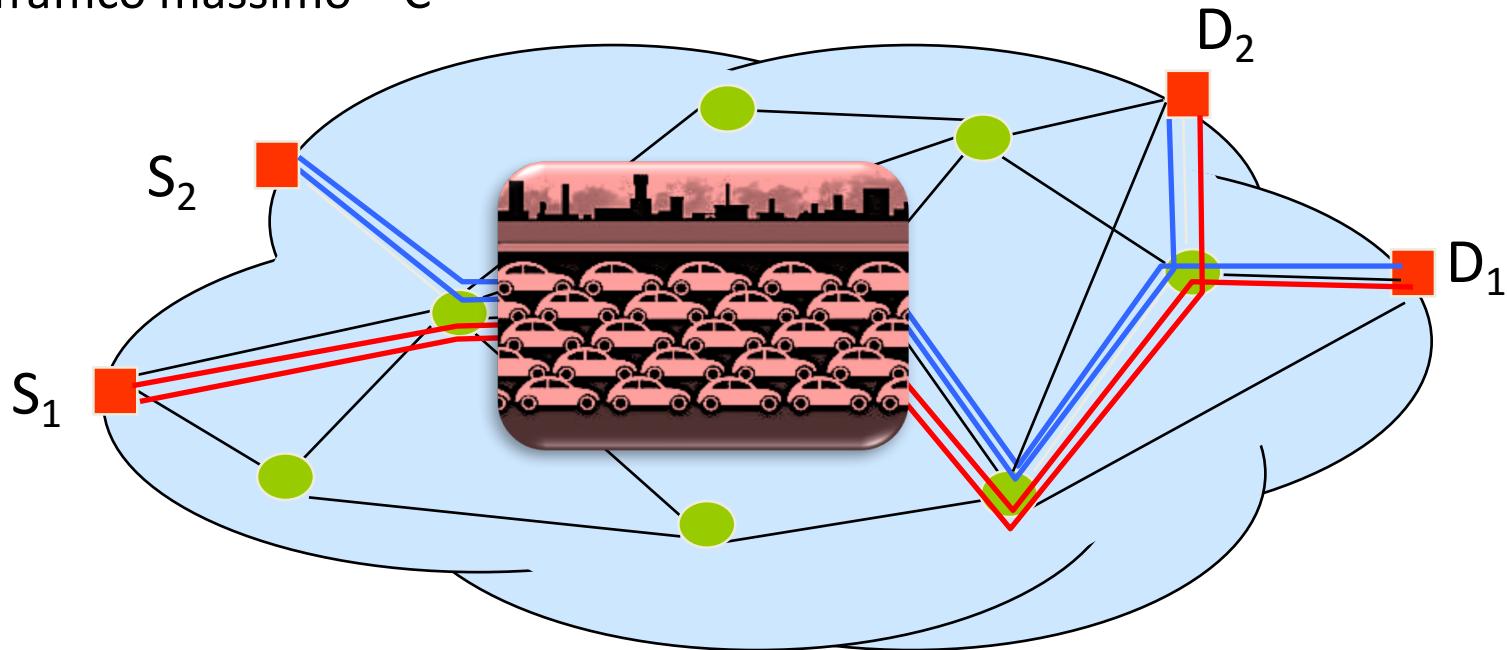


Routing e capacità: esempio (1)

- Ad esempio:
 - Se il traffico viene fatto passare da pochi cammini nella rete il traffico massimo sarà basso

Capacità di tutti i link = C

Traffico massimo = C

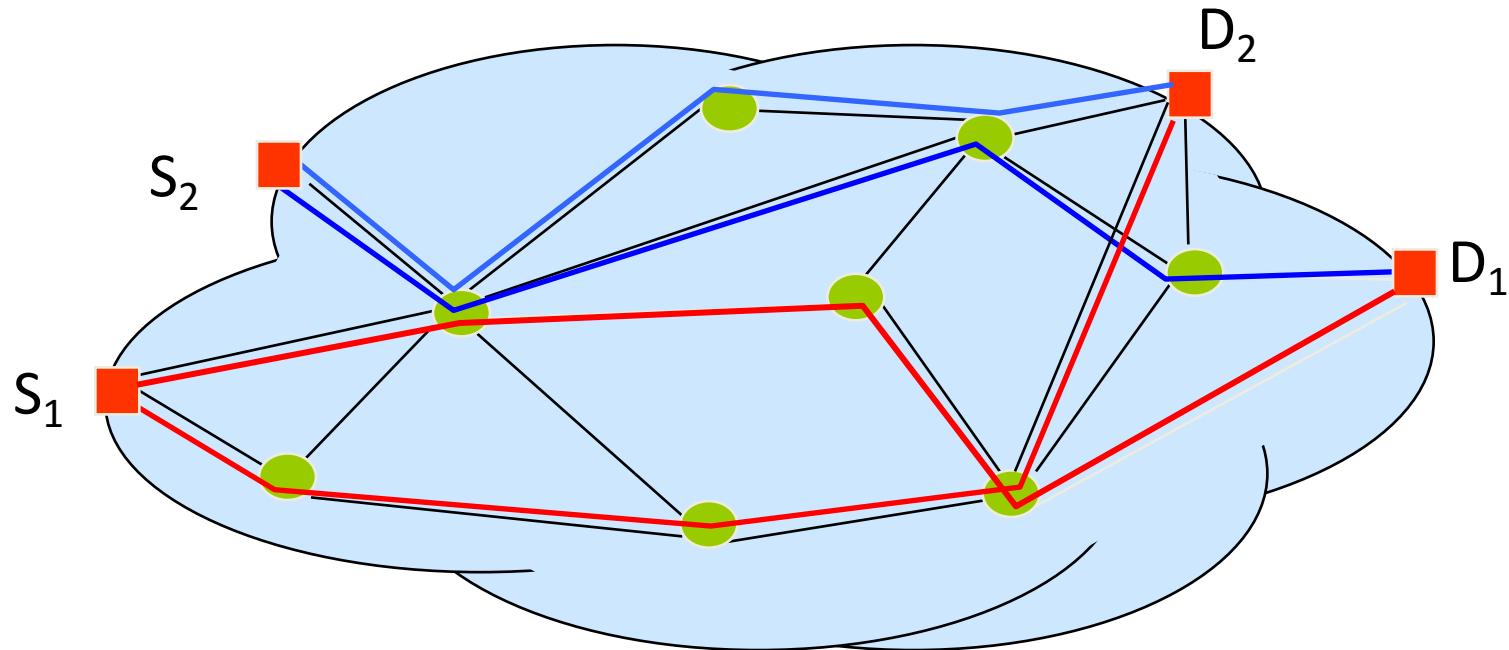


Routing e capacità: esempio (2)

- Se invece si usano molti cammini ripartendo il carico il massimo traffico sarà elevato

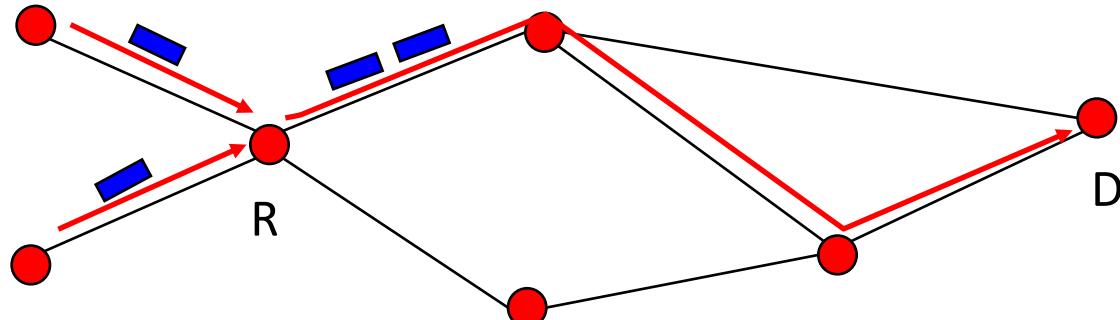
Capacità di tutti i link = C

Traffico massimo = 3C



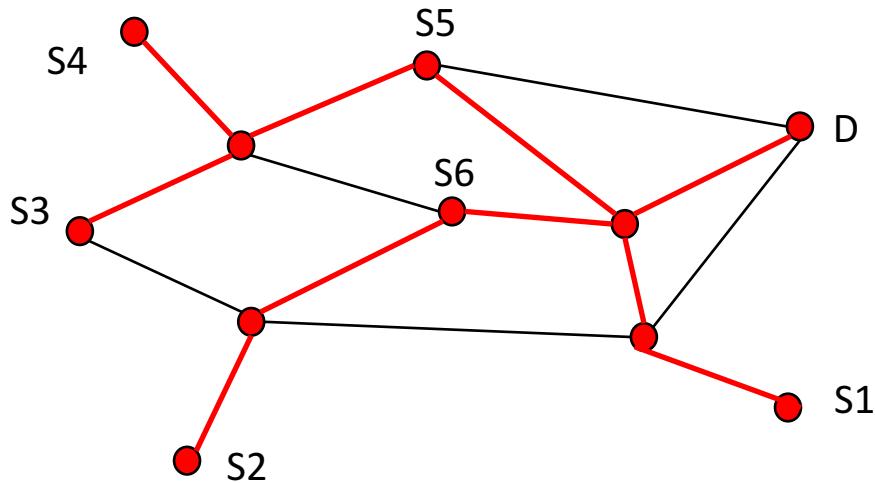
Politiche di routing per Internet

- Il tipo di inoltro (*forwarding*) utilizzato dalle reti IP condiziona la scelta delle politiche di routing
- Ricordiamo che il *forwarding* IP è
 - *Basato sull'indirizzo di destinazione (destination-based)*
 - *Con inoltro al nodo successivo (next-hop routing)*
- Come conseguenza:
 - *I pacchetti diretti ad una stessa destinazione D che giungono in un router R seguono lo stesso percorso da R verso D indipendentemente dal link di ingresso in R*



Politiche di routing per Internet

- Quindi il vincolo che ogni politica di *routing* deve soddisfare è che
 - *l'insieme dei cammini da ogni sorgente verso una destinazione D sia un albero, per ogni possibile destinazione D*

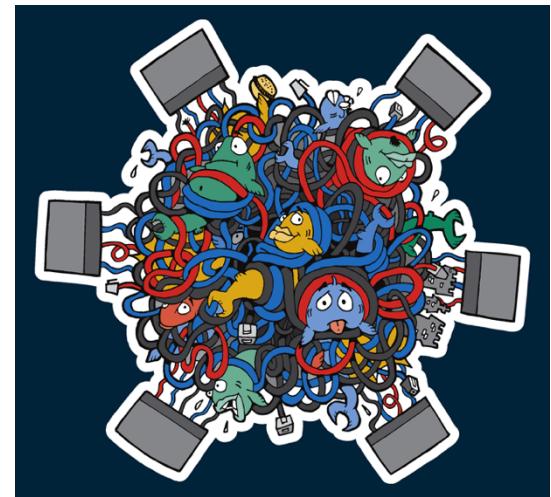


- *Non è dunque possibile instradare in modo indipendente ogni relazione di traffico (coppia sorgente-destinazione)*



Routing IP

- Il principio su cui si basa il *routing* IP è molto semplice
 - Inviare i pacchetti sul *cammino minimo* verso la destinazione
 - La metrica su cui si calcolano i cammini minimi è generale
 - Il calcolo avviene in modo distribuito dai *router* mediante uno scambio di informazioni con gli altri *router*
 - Nella tabella viene indicato solo il primo *router* sul cammino grazie alla proprietà secondo la quale anche i sotto-cammini di un cammino minimo sono minimi



Agenda

- Instradamento in rete
 - Caratteristiche
 - Algoritmi su grafi
 - Algoritmo di Bellman-Ford
 - Algoritmo di Dijkstra
 - Algoritmi d'instradamento a costo minimo
 - Instradamento Distance Vector
 - Instradamento Link State

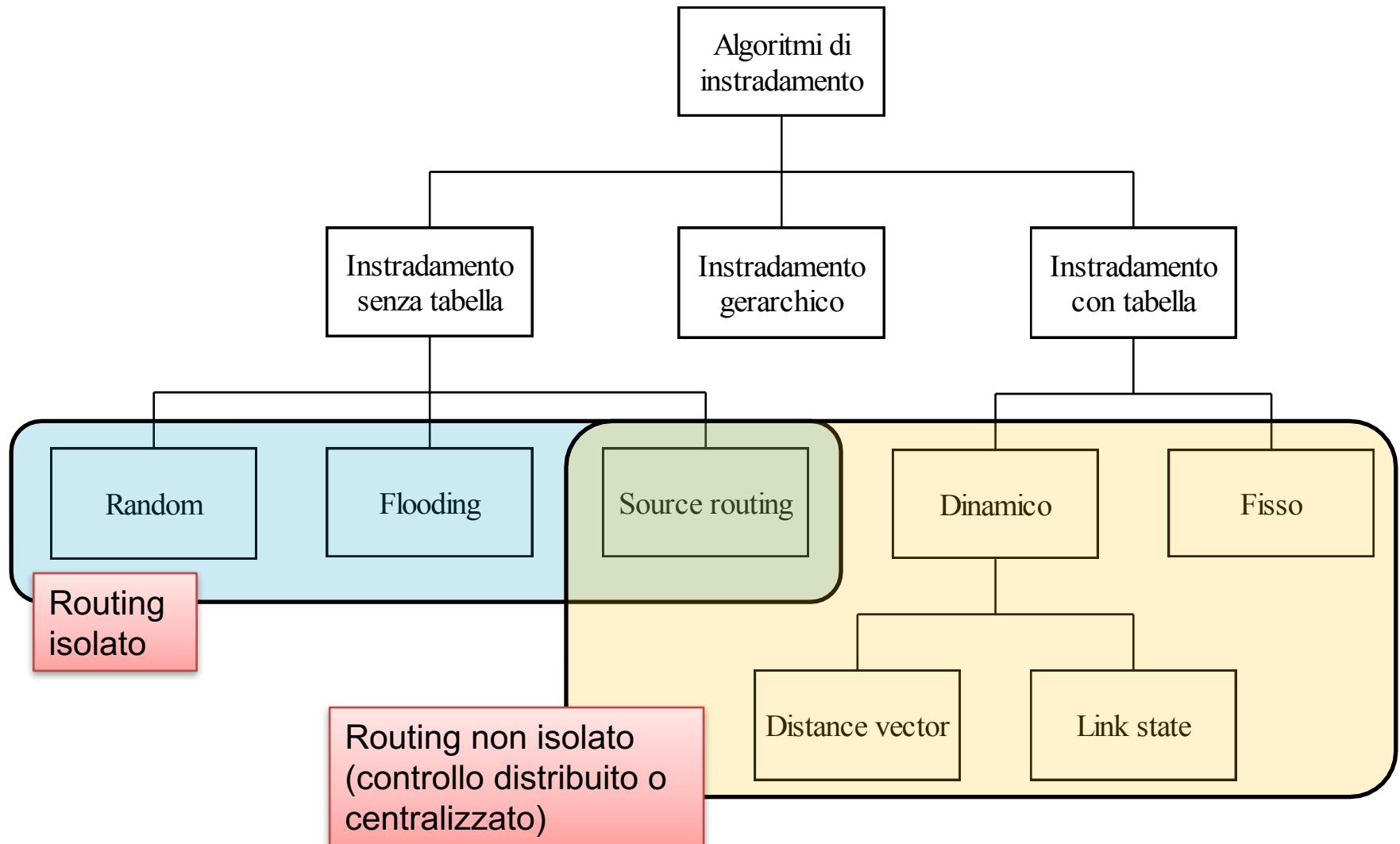


Algoritmi di instradamento: caratteristiche

- Requisiti di un algoritmo di instradamento
 - Semplicità
 - Robustezza
 - Stabilità
 - Ottimalità
- Pacchetti che devono essere instradati
 - Pacchetti di segnalazione in servizi a circuito virtuale
 - Pacchetti dati in servizi datagram
- Localizzazione della decisione di instradamento
 - Algoritmi centralizzati: un unico centro di controllo prende tutte le decisioni
 - Algoritmi distribuiti: tutti i nodi cooperano per determinare il migliore instradamento in ogni nodo
 - Algoritmi isolati: il nodo sorgente prende le proprie decisioni eventualmente anche in base a informazioni chieste ad altri nodi



Algoritmi di instradamento: classificazione generale



Algoritmi di instradamento con tabella

- Algoritmo di instradamento a distanza minima (o costo minimo) secondo un'opportuna metrica
- Richiede la definizione di una metrica
 - Numero di salti
 - Capacità dei link sulla via
 - Ritardo medio sulla via
 - Numero totale di pacchetti in coda sulla via
 - ecc.
- Definisce la tabella di instradamento che indica per ogni destinazione di rete il nodo successivo verso cui instradare il pacchetto



Instradamento statico (fisso)

- Un centro di controllo costruisce le tabelle di instradamento che devono essere applicate da ogni singolo nodo e le comunica ai nodi stessi
- Le tabelle vengono cambiate solo a seguito di aggiornamento della topologia, su azione del centro di controllo
- Poco flessibile, in quanto non reagisce a sovraccariche e guasti aleatori
- Consente un'accurata pianificazione di rete (traffic engineering)
- Due fasi:
 - Fase di progetto
 - Le rotte vengono decise scegliendo per ciascun router l'insieme dei next-hop router secondo un opportuno criterio
 - Fase di configurazione
 - Le tabelle dei router vengono configurate con opportune procedure (si vedrà a laboratorio)



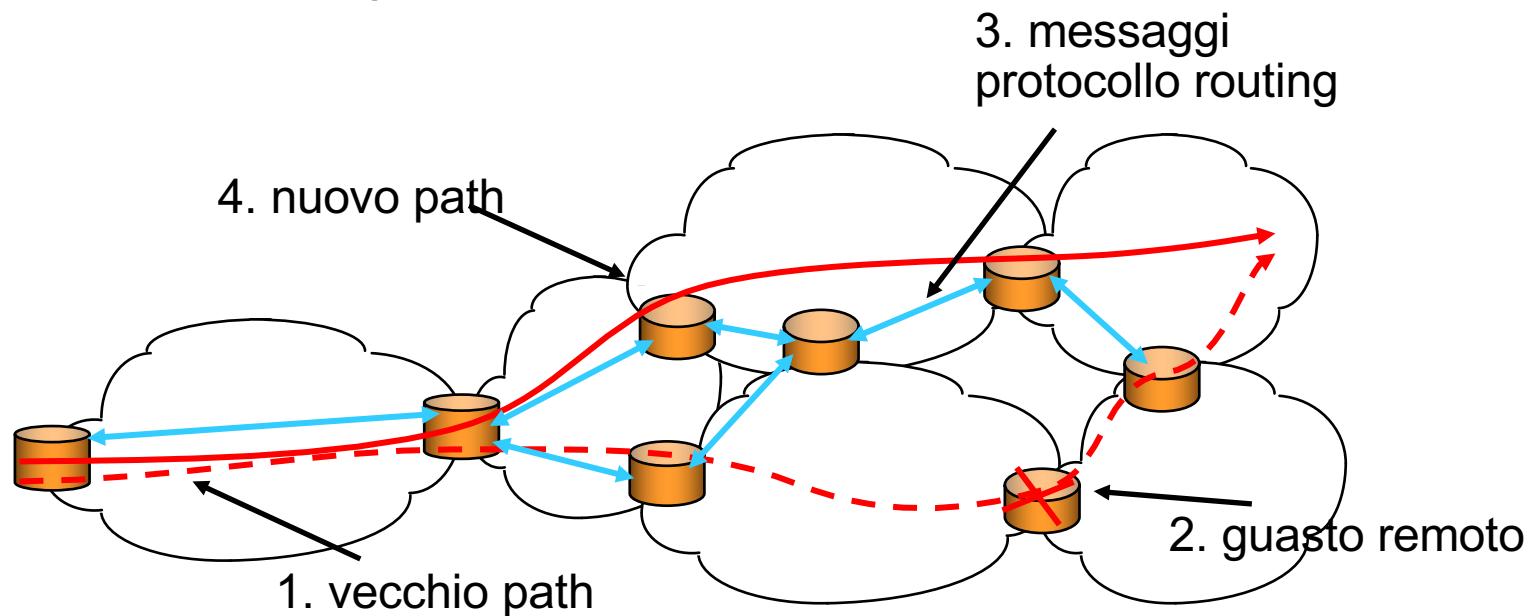
Instradamento dinamico (adattativo)

- La tabella di routing di ciascun router varia nel tempo in base alle indicazioni che il router riceve dagli altri router grazie al protocollo di routing
- In un protocollo di routing si definiscono
 - Una metrica per valutare il costo degli elementi di rete (link e a volte anche nodi)
 - Possibili metriche: ritardo, numero di hop, capacità disponibile o utilizzata, affidabilità, ecc.
 - L'algoritmo per scegliere i percorsi migliori (a minimo costo)
 - La modalità con cui vengono scambiati i messaggi di aggiornamento tra i router
 - Tipo, contenuto, frequenza
- Il routing che si utilizza in Internet è dinamico



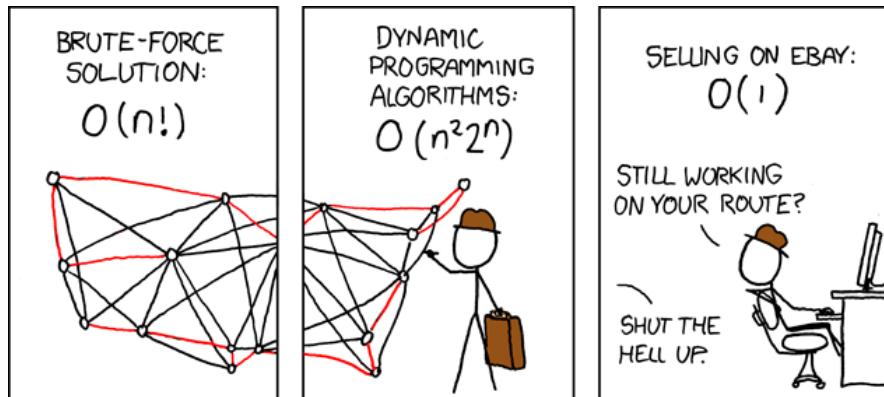
Instradamento dinamico (adattativo)

- Perché si modificano le tabelle di routing?
 - Cambiamenti nella topologia della rete
 - Quando viene attivato un nuovo link, usalo!
 - Carico del traffico e congestione di rete
 - Se c'è un link meno carico di un altro, usalo!
 - Guasti di link e nodi
 - Se un link “è giù”, trova un percorso alternativo!



Routing a cammini minimi

- La politica di routing utilizzata sin dall'introduzione delle reti TCP/IP è basata sul calcolo dei cammini minimi
- Il calcolo è effettuato sul **grafo** che rappresenta la rete nel quale ad ogni arco è associato un peso opportunamente scelto (metrica)
- I motivi di questa scelta sono fondamentalmente:
 - i cammini minimi verso una destinazione formano un albero (l'albero dei cammini minimi);
 - esistono degli algoritmi di calcolo dei cammini minimi che possono essere eseguiti in modo distribuito nei nodi della rete.
- Dato un grafo e dei pesi associati agli archi il calcolo del cammino minimo si può ottenere con algoritmi di complessità polinomiale nel numero di nodi (sinonimo di **algoritmo veloce**).



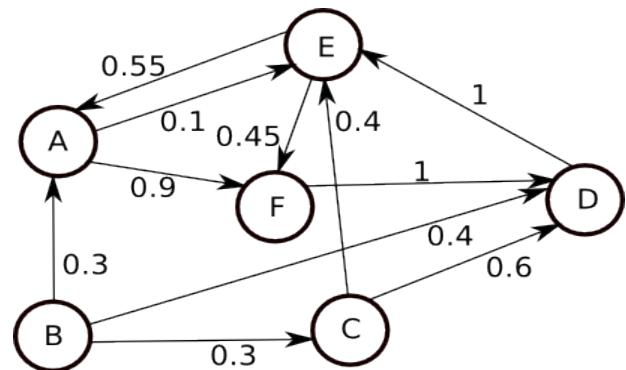
Agenda

- Instradamento in rete
 - Caratteristiche
 - Algoritmi su grafi
 - Algoritmo di Bellman-Ford
 - Algoritmo di Dijkstra
 - Algoritmi d'instradamento a costo minimo
 - Instradamento Distance Vector
 - Instradamento Link State



Richiami sui grafi

- **digrafo $G(N,A)$**
 - N nodi
 - $A=\{(i,j), i\in N, j\in N\}$ archi (coppia ordinata di nodi)
- **percorso:** (n_1, n_2, \dots, n_l) insieme di nodi con $(n_i, n_{i+1}) \in A$
- **cammino:** percorso senza nodi ripetuti
- **ciclo:** percorso con $n_1 = n_l$
- **digrafo connesso:** per ogni coppia i e j esiste almeno un cammino da i a j
- **digrafo pesato:** d_{ij} peso associato all'arco $(i,j) \in A$
- **lunghezza di un cammino** (n_1, n_2, \dots, n_l) :
$$d_{n1, n2} + d_{n2, n3} + \dots + d_{n(l-1), nl}$$



Problema del cammino minimo

- Il problema è di complessità polinomiale

Dato un digrafo $G(N,A)$ e due nodi i e j , trovare il cammino di lunghezza minima tra tutti quelli che consentono di andare i a j



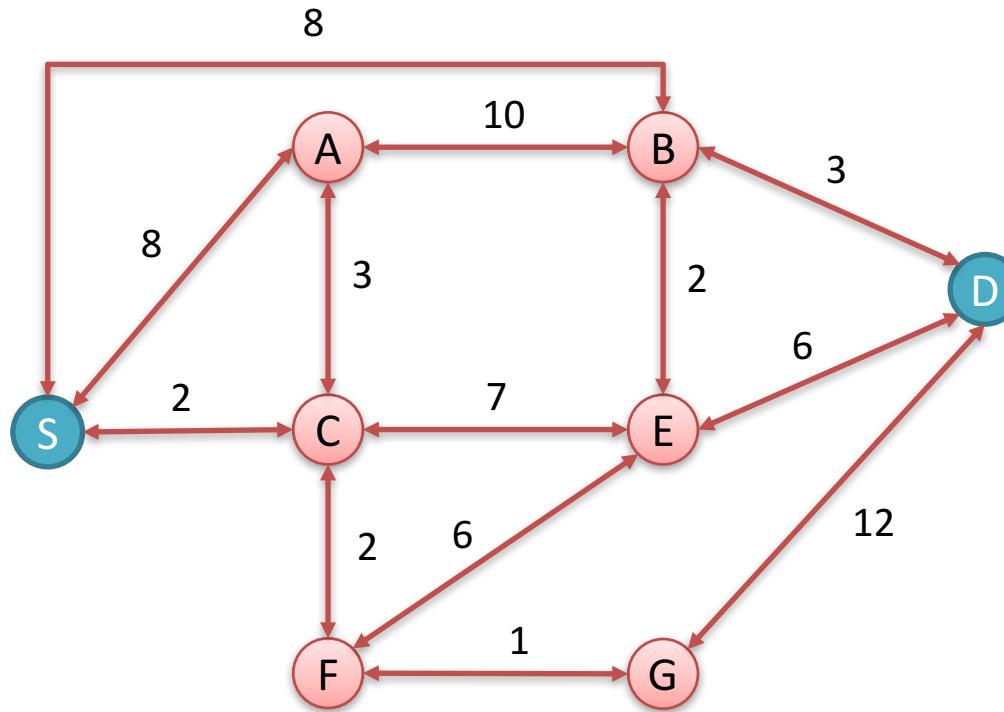
Proprietà:

Se il nodo k è attraversato dal cammino minimo da i a j , il sotto-cammino fino a k è anch'esso minimo



Esempio

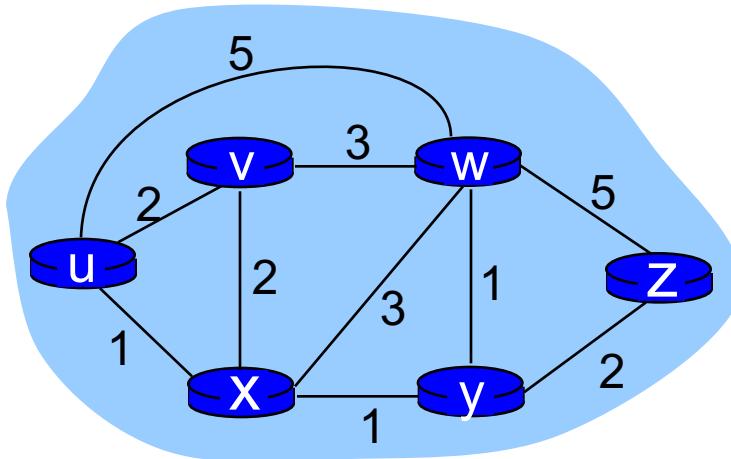
- Quale è il cammino minimo da S a D?



- Come è stato trovato? Elencando tutti i possibili percorsi?
- Come convincere il professore che è quello minimo e non ce ne sono altri migliori?



Esempio di grafo



graph: $G = (N, E)$

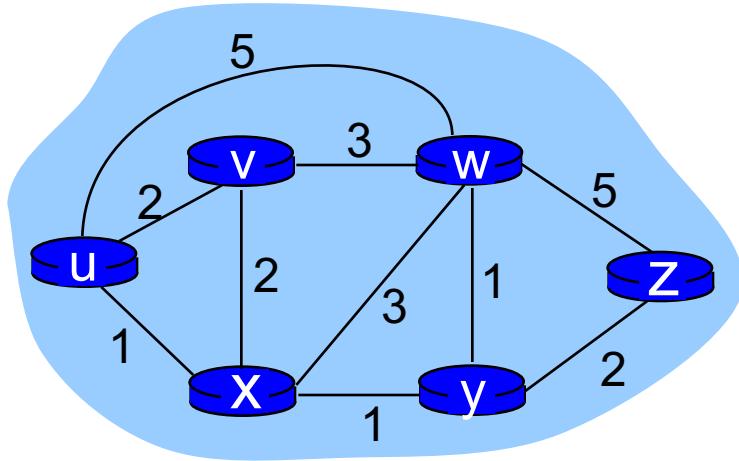
$N = \text{insieme di router} = \{ u, v, w, x, y, z \}$

$E = \text{insieme di link} = \{ (u,v), (u,x), (v,w), (v,x), (v,y), (w,z), (w,y), (x,y) \}$

- L'astrazione mediante grafo è utile anche in altri contesti di rete, come ad esempio il peer-2-peer (N insieme dei peers, E insieme di connessioni TCP)



Esempio di grafo: costi



$c(x,x')$ = costo del link (x,x')
e.g., $c(w,z) = 5$

Il costo può essere fisso, o inversamente proporzionale alla banda disponibile o inversamente proporzionale alla congestione, ecc.

Costo del percorso $(x_1, x_2, x_3, \dots, x_p)$ = $c(x_1, x_2) + c(x_2, x_3) + \dots + c(x_{p-1}, x_p)$

- Domanda fondamentale: qual è il percorso a costo minimo tra u e z ?
- L'algoritmo di routing troverà tale percorso



Agenda

- Instradamento in rete
 - Caratteristiche
 - Algoritmi su grafi
 - Algoritmo di Bellman-Ford
 - Algoritmo di Dijkstra
 - Algoritmi d'instradamento a costo minimo
 - Instradamento Distance Vector
 - Instradamento Link State



Algoritmo di Bellman-Ford

- Ipotesi:
 - pesi sia positivi che negativi
 - non esiste alcun ciclo di lunghezza negativa
- Scopo:
 - trovare i cammini minimi tra un nodo (sorgente s) e tutti gli altri nodi, *oppure*
 - trovare i cammini minimi da tutti i nodi ad un nodo (destinazione d)



Algoritmo di Bellman-Ford

- Variabili aggiornate nelle iterazioni:
 - $D_i^{(h)}$ lunghezza del cammino minimo tra il nodo i e il nodo 1 composto da un numero di archi $\leq h$
- Valori iniziali:

$$D_s^{(h)} = 0 \quad \forall h$$

$$D_i^{(0)} = \infty \quad \forall i \neq 1$$

- Iterazioni:

$$D_i^{(h+1)} = \min \left[D_i^{(h)}, \min_j (D_j^{(h)} + d_{ji}) \right]$$

- L'algoritmo termina in $N-1$ passi



Pseudo-codice dell'algoritmo di Bellman-Ford

d_{ij} = costo del collegamento diretto da i e j
 $(=\infty$ in assenza del collegamento)

D_j^h = costo del percorso a minimo costo
da s a j con max h hop

$h=0;$

$D_s^h = 0;$

$D_j^h = \infty \quad \forall j \neq s;$

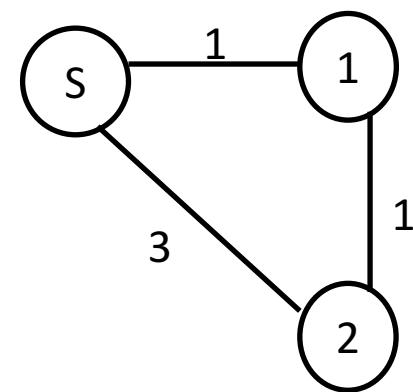
repeat

$h = h + 1;$

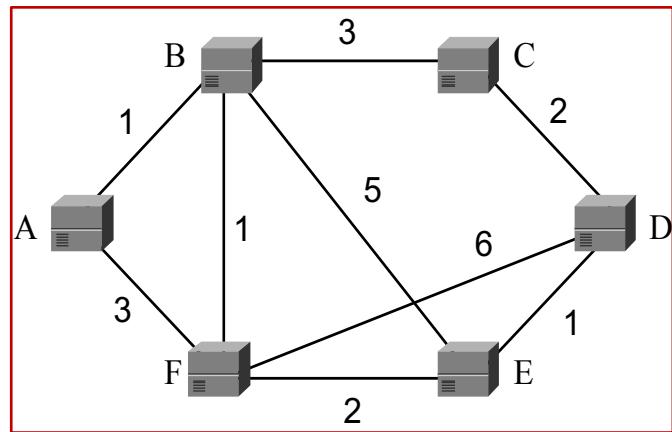
$D_j^h = \min_i \{D_i^{h-1} + d_{ij}, D_j^{h-1}\};$

until $D_j^h = D_j^{h-1} \quad \forall j \neq s$

- Initializzazione
 - $D_s^h=0$
 - $D_1^0=\text{inf}$
 - $D_2^0=\text{inf}$
- Prima iterazione
 - $D_1^1=\min(D_1^0, D_s^0+1)=1$, NH:S
 - $D_2^1=\min(D_2^0, D_s^0+3)=3$, NH:S
- Seconda iterazione
 - $D_1^2=\min(D_1^1, D_2^1+1)=1$, NH:S
 - $D_2^2=\min(D_2^1, D_1^1+1)=2$, NH:1

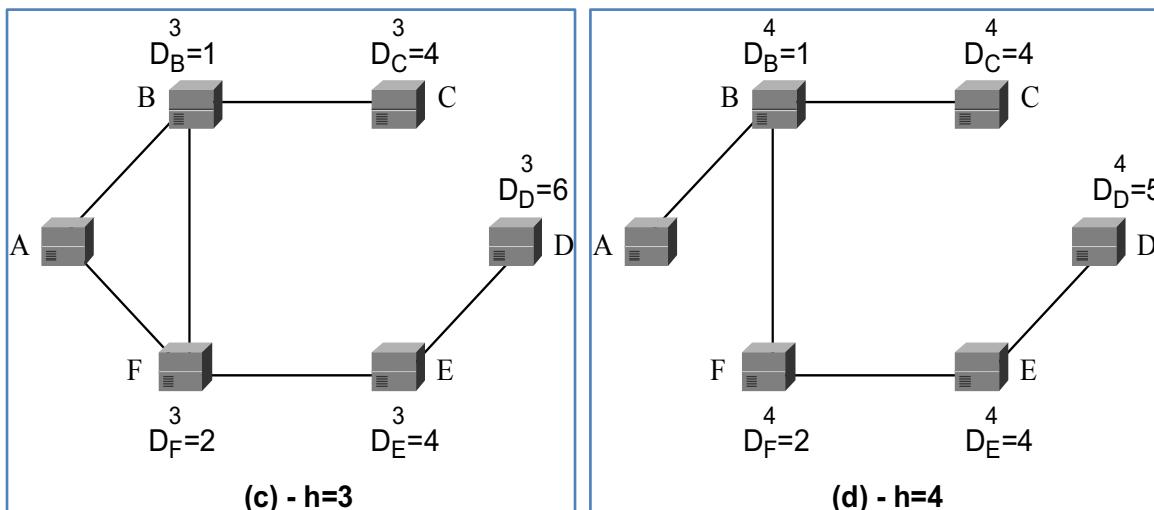
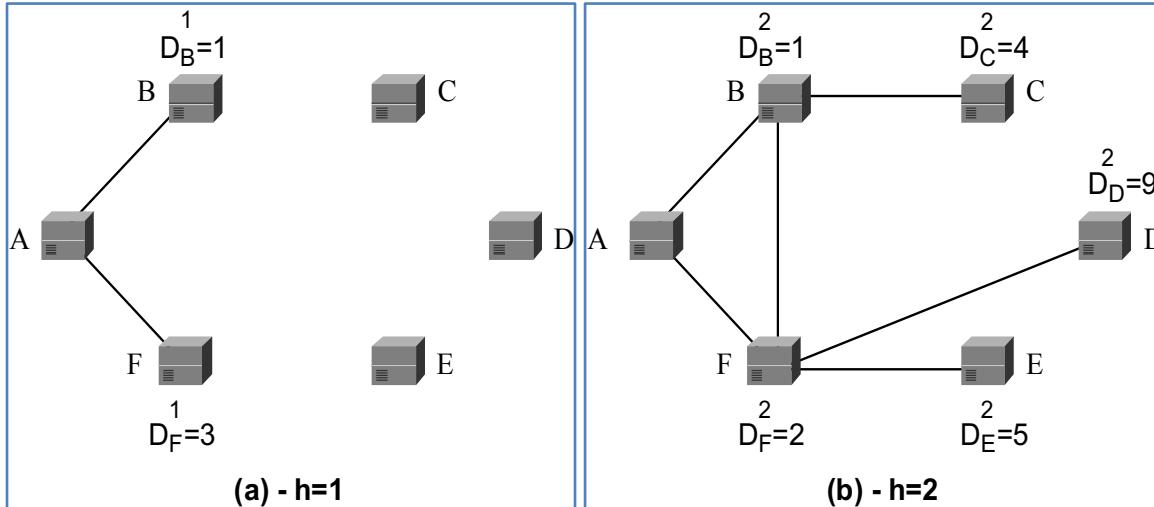


Algoritmo di Bellman-Ford: esempio



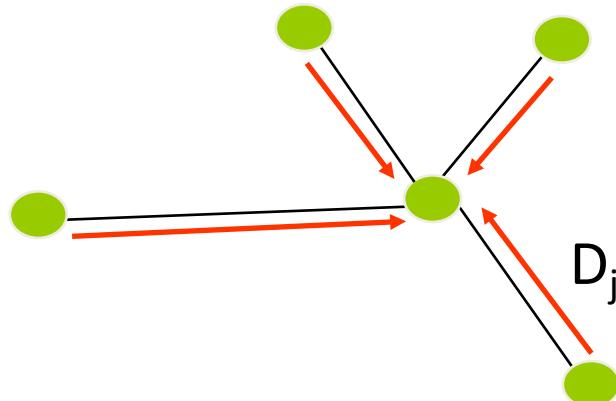
$$D_j^h = \min_i \{ D_i^{h-1} + d_{ij}, D_j^{h-1} \}$$

Src	h=1		h=2		h=3		h=4	
	Cost	NH	Cost	NH	Cost	NH	Cost	NH
A	-		-		-		-	
B	1	A	1	A	1	A	1	A
C	infty		4	B	4	B	4	B
D	infty		9	F	6	E	5	E
E	infty		5	F	4	F	4	F
F	3	A	2	B	2	B	2	B



Algoritmo di Bellman-Ford in forma distribuita

- Si dimostra che l'algoritmo converge in un numero finito di passi anche nel caso in cui viene implementato in modo distribuito
- Periodicamente i nodi inviano l'ultima stima del cammino minimo ai vicini e aggiornano la propria stima secondo il criterio delle iterazioni
- L'ordine di aggiornamento delle etichette è dunque ininfluente



$$D_i := \min \left[D_i, \min_j (D_j + d_{ji}) \right]$$

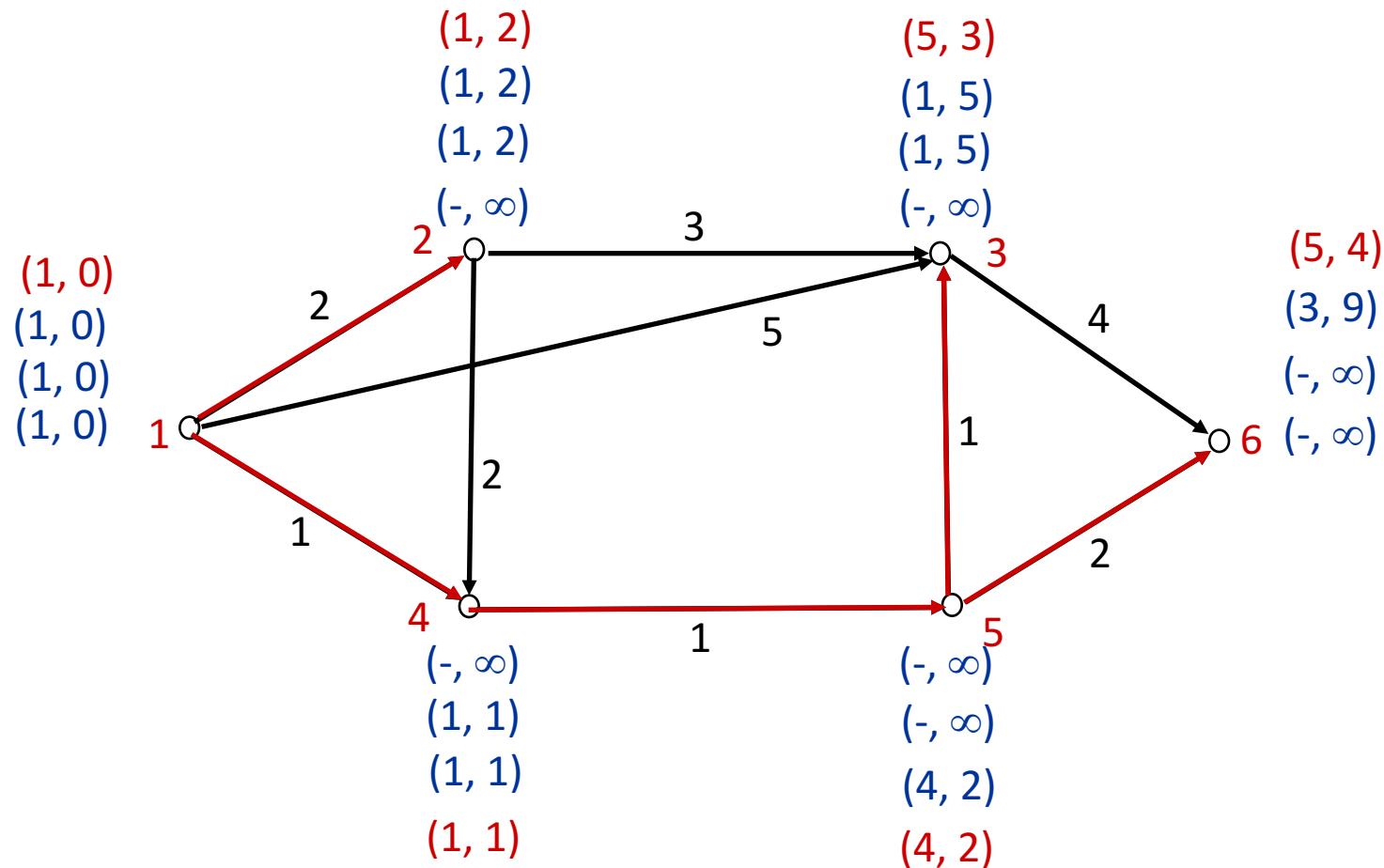


Algoritmo di Bellman-Ford in pratica

- Per poter applicare praticamente l'algoritmo si può procedere in questo modo:
- Si usano delle etichette per i nodi (n, L) dove n indica il primo nodo sul cammino minimo ed L la sua lunghezza
- Le etichette vengono aggiornate guardando le etichette dei vicini (l'ordine non conta grazie alla proprietà dell'algoritmo distribuito)
- Quando le etichette non cambiano più si ricostruisce l'albero dei cammini minimi ripercorrendo le etichette



Algoritmo di Bellman-Ford in pratica



Agenda

- Instradamento in rete
 - Caratteristiche
 - Algoritmi su grafi
 - Algoritmo di Bellman-Ford
 - Algoritmo di Dijkstra
 - Algoritmi d'instradamento a costo minimo
 - Instradamento Distance Vector
 - Instradamento Link State

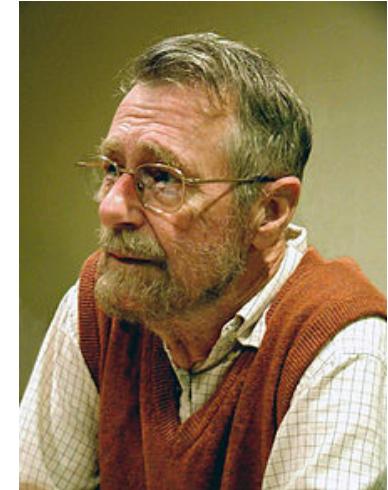


Algoritmi di *Dijkstra*

- Ipotesi:
 - archi con pesi positivi
- Scopo:
 - trovare il cammini da tutti i nodi ad un nodo 1
- Valori iniziali:

$$P = \{1\},$$

$$D_1 = 0, \quad D_j^{(0)} = d_{1j} \quad \forall j \neq 1$$



- si assume $d_{ij} = \infty$ se l'arco tra i e j non esiste



Algoritmo di Dijkstra

Repeat

1. Scegliere $i \in (N \setminus P)$ tale che

$$D_i = \min_{j \in (N - P)} D_j$$

modificare

$$P := P \cup \{ i \}. \text{ Se } P = N, \text{ allora STOP.}$$

2. Per ogni $j \in (N \setminus P)$ vicino ad almeno un nodo in P:

$$D_j = \min \left[D_j, \min_k (D_k + d_{kj}) \right]$$

endrepeat

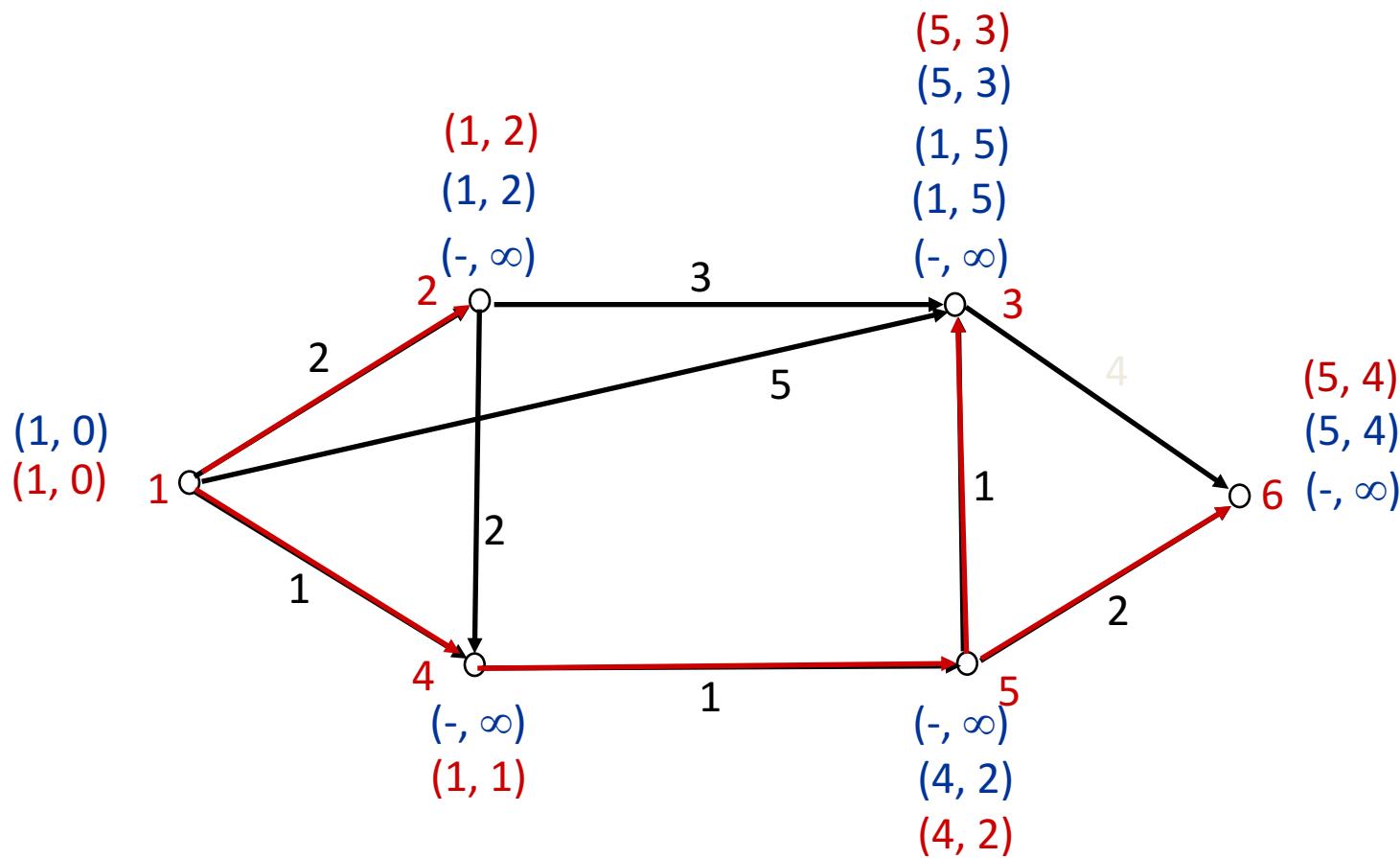


Algoritmi di Dijkstra in pratica

- Si applica lo stesso criterio di Bellman-Ford
- L'unica differenza consiste nella distinzione tra etichette temporanee e permanenti
 - all'inizio l'unica etichetta permanente è quella della sorgente
 - ad ogni iterazione l'etichetta temporanea con la lunghezza più corta diventa permanente

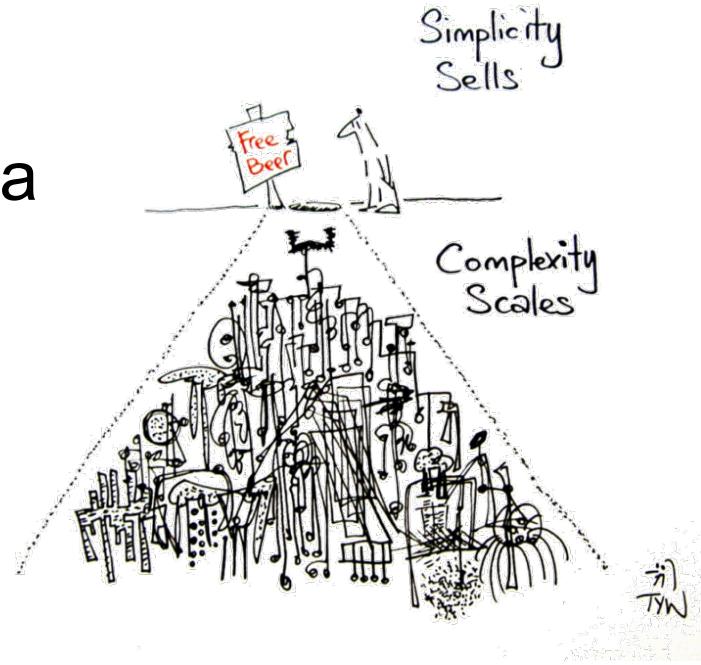


Esempio: Dijkstra



Algoritmi: complessità

- L'algoritmo di Bellman-Ford ha una complessità:
 - N-1 iterazioni
 - N-1 nodi per iterazione
 - N-1 confronti per nodo
 - Complessità: $O(N^3)$
- L'algoritmo di Dijkstra ha una complessità:
 - N-1 iterazioni
 - in media N operazioni per iterazioni
 - Complessità: $O(N^2)$
- L'algoritmo di Dijkstra è in generale più conveniente



Instradamento: informazione globale / decentralizzata

- I protocolli di routing si possono classificare in base al tipo di informazione disponibile a ciascun router
 - Globale
 - Tutti i router vedono la topologia completa della rete e hanno informazione di costo per ciascun link
 - Tipico degli algoritmi detti “link state”
 - Decentralizzata
 - I router conoscono i vicini direttamente connessi e il costo dei link verso i vicini
 - Riescono a determinare la tabella di routing scambiando informazioni esclusivamente con i vicini e applicando un processo di calcolo iterativo
 - Tipico degli algoritmi detti “distance vector”



Agenda

- Instradamento in rete
 - Caratteristiche
 - Algoritmi su grafi
 - Algoritmo di Bellman-Ford
 - Algoritmo di Dijkstra
 - **Algoritmi d'instradamento a costo minimo**
 - Instradamento Distance Vector
 - Instradamento Link State



Algoritmi d'instradamento a costo minimo

- Algoritmo distance vector
 - Usato in ARPANET fino al 1979 con periodo di aggiornamento di 125 ms
 - Basato sulla lunghezza delle code (non su velocità dei link)
 - Anche con metriche diverse era troppo lento a convergere in un contesto dinamico
- Successivo algoritmo per ARPANET (1979): link state
 - Ogni nodo misura la distanza (secondo la metrica ritardo) a tutti i suoi vicini
 - Questa distanza è comunicata a tutti altri nodi con flooding
 - Ogni nodo può così costruirsi i percorsi a minima distanza ad ogni altro nodo
 - Periodo di aggiornamento nell'ordine dei 10 s
- Algoritmo path vector
 - I router si scambiano percorsi, cioè sequenze di AS da attraversare fino a destinazione
 - Ibrido tra distance vector e link state



Agenda

- Instradamento in rete
 - Caratteristiche
 - Algoritmi su grafi
 - Algoritmo di Bellman-Ford
 - Algoritmo di Dijkstra
 - Algoritmi d'instradamento a costo minimo
 - Instradamento Distance Vector
 - Instradamento Link State



Instradamento Distance vector

- Output
 - Tabella di instradamento in ogni nodo che specifica la minima distanza ad ogni altro nodo e quale nodo a valle deve essere utilizzato
- Può essere implementato in forma centralizzata o distribuita
- Molto più diffusa l'implementazione distribuita
 - Il nodo riceve la stima delle distanze dai suoi vicini (vettore delle distanze), somma la sua distanza al vicino e scopre la distanza minima verso ogni altro nodo e il nodo a valle relativo



Algoritmo Distance Vector in forma distribuita

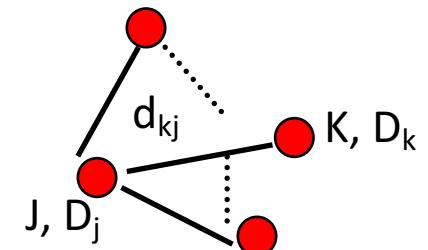
- L'informazione sulla connettività scambiata dai nodi è costituita dal Distance Vector (DV):
 - [indirizzo destinatario, distanza]
- Il DV è inviato ai soli nodi adiacenti
- Il DV è inviato periodicamente o a seguito di un cambiamento nella topologia di rete
- La stima delle distanze avviene tramite Bellman-Ford distribuito



Algoritmo Distance Vector in forma distribuita

- Ogni nodo invia il DV
 - periodicamente
 - se il risultato di un ricalcolo differisce dal precedente
- Ogni nodo esegue il ricalcolo delle distanze se
 - riceve un DV diverso da quello memorizzato in precedenza
 - cade/nasce una linea attiva a cui è connesso

$$\text{Ricalcolo: } D_j' = \min_k [D_k + d_{kj}]$$



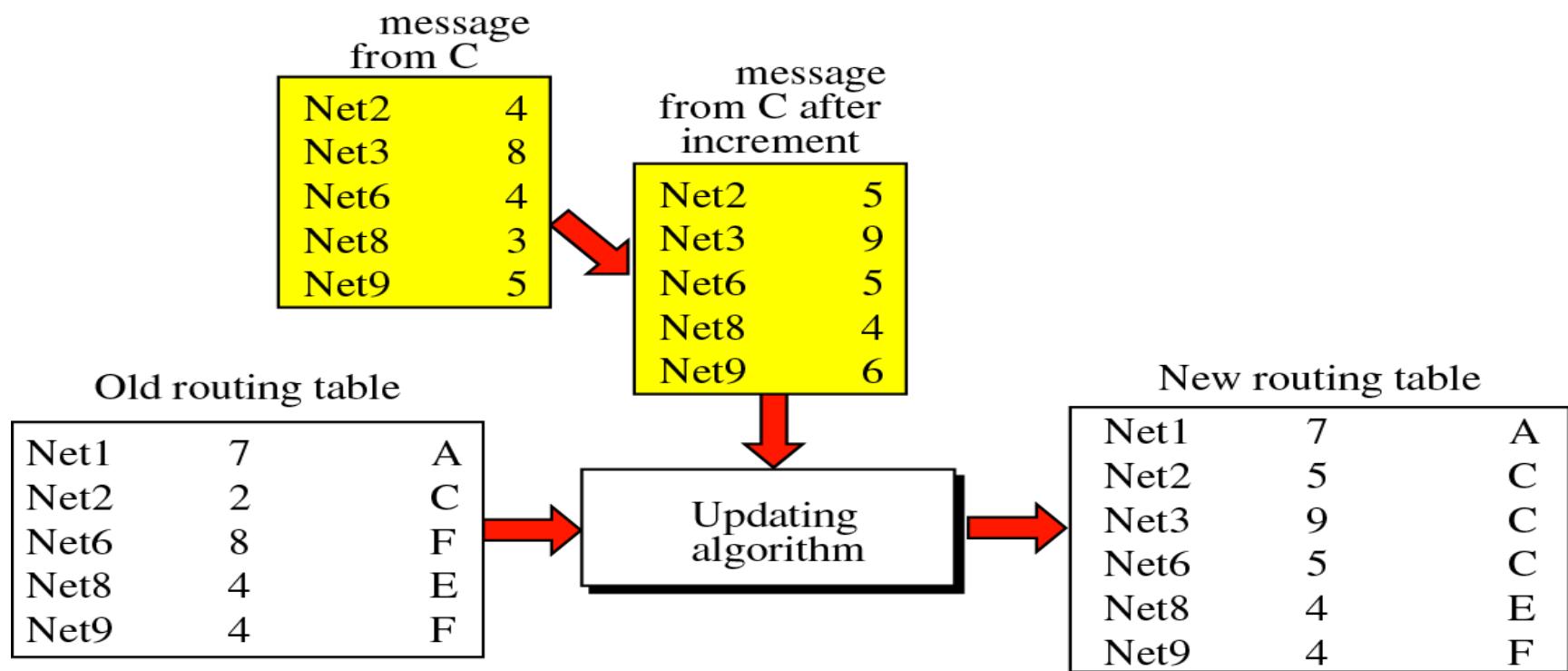
Algoritmo Distance Vector distribuito

Ricezione DV da un vicino

1. Incrementa la distanza dalle destinazioni specificate del costo del link in ingresso
2. Ripeti per ogni destinazione specificata nel DV
 - Se la destinazione non è nella tabella di routing
 - Aggiungi la destinazione/distanza
 - Altrimenti
 - Se il next hop nella tabella di routing corrisponde al mittente del DV
 - Sostituisci l'informazione della tabella di routing con quella nuova
 - Altrimenti
 - Se la distanza indicata nel DV è minore di quella scritta nella tabella di routing
 - » Sostituisci l'informazione della tabella di routing con quella nuova
3. Termina



Modifica delle tabelle di routing



Rules

Net1: No news, don't change

Net2: Same next hop, replace

Net3: A new router, add

Net6: Different next hop, new hop count smaller, replace

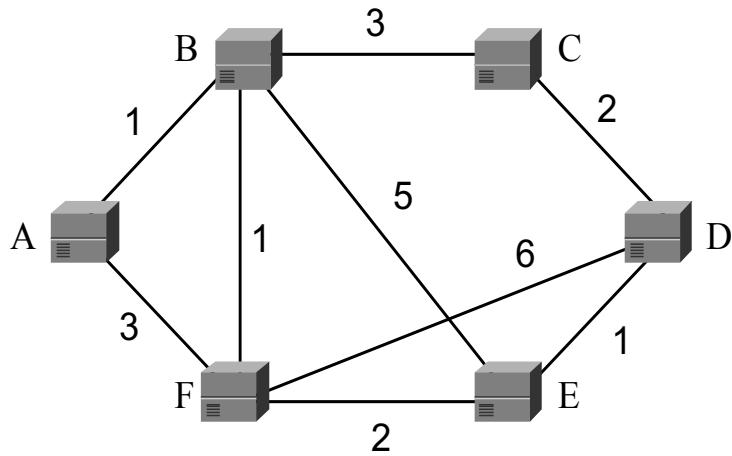
Net8: Different next hop, new hop count the same, don't change

Net9: Different next hop, new hop count larger, don't change



Distance vector: esempio

- Calcolo della tabella di instradamento nel nodo A
 - Aggiornamento ogni T s
 - In $[0, T)$ A vede solo i suoi vicini
 - Primo DV ricevuto in A da B e F a $t = T$



Dest.	Cost	Route
A	-	
B	1	B
C		
D		
E		
F	3	F

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	9	F
E	5	F
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	6	F
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

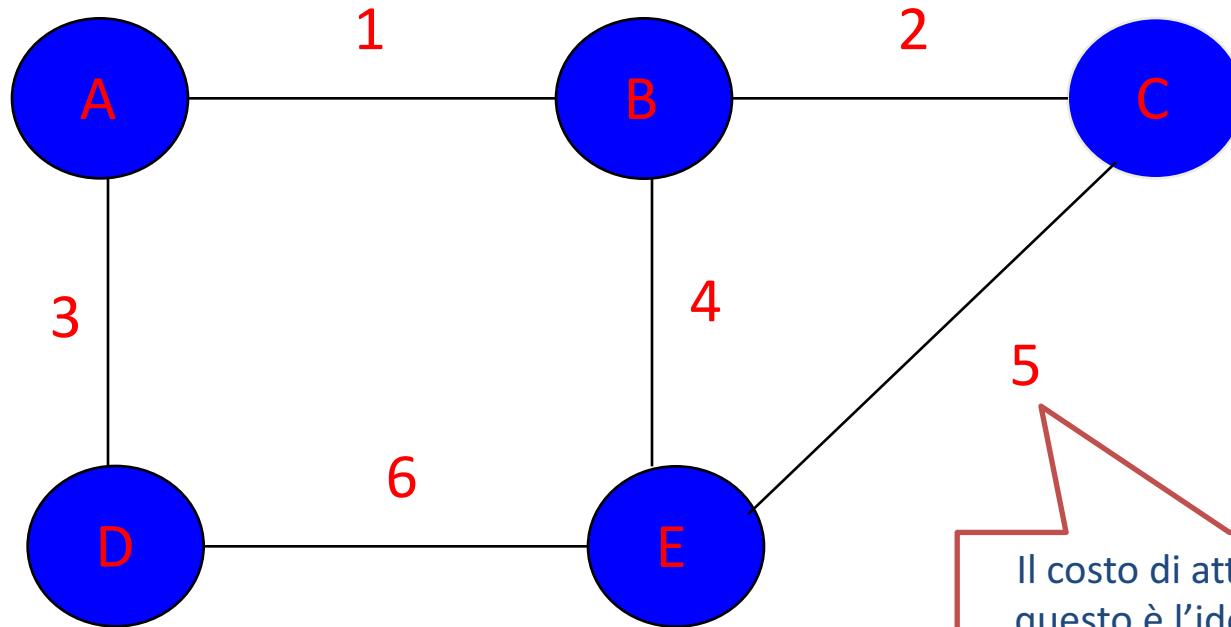
Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route
A	-	
B	1	B
C	4	B
D	5	B
E	4	B
F	2	B

Dest.	Cost	Route

Esempio: Distance Vector (1)



- Consideriamo la rete soprastante
 - Ogni nodo (non differenziamo tra host e router) è identificato da un suo indirizzo (A, B, C, D o E)
 - Supponiamo che ogni link abbia costo 1



Esempio: Distance Vector (2)

- Tutti i nodi si attivano contemporaneamente
👉 procedura *cold start*
- Ogni nodo ha delle informazioni iniziali permanenti (*conoscenze locali*), in particolare conosce il suo indirizzo e a quali *link* è direttamente connesso, non conosce gli altri nodi nella rete
- Inizialmente le tabelle di *routing* contengono solo la *entry* del nodo, per esempio il nodo A

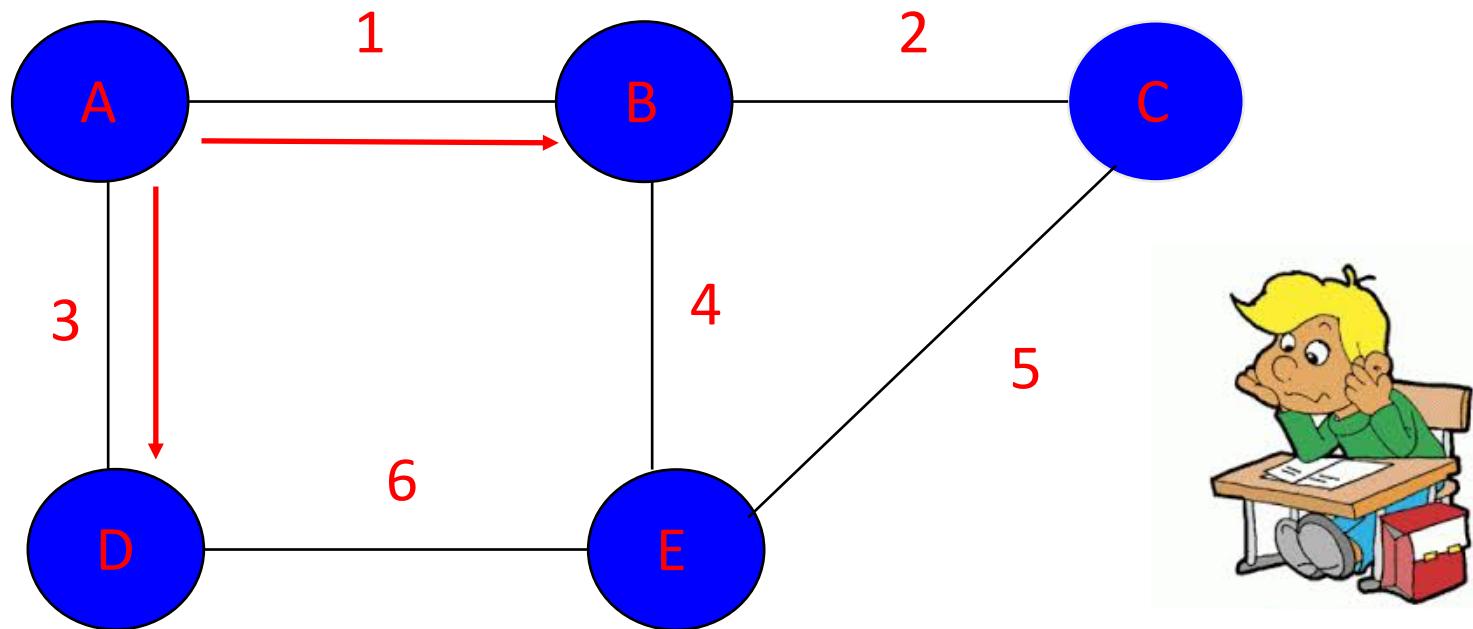
Da A verso	Link	Costo
A	locale	0

Identificativo del link adiacente da attraversare per raggiungere la destinazione



Esempio: Distance Vector (3)

- Da questa tabella il nodo A estrae il *Distance Vector* $A=0$ e lo trasmette a tutti i vicini, cioè su tutti i *link* locali
- B e D ricevono l'informazione e allargano le loro conoscenze locali,



Esempio: Distance Vector (4)

- Il nodo B, dopo aver ricevuto il *Distance Vector*, aggiorna la distanza aggiungendo il costo del *link* locale trasformando il messaggio in $A=1$, lo confronta con le informazioni nella sua tabella di *routing* e vede che il nodo A non è conosciuto

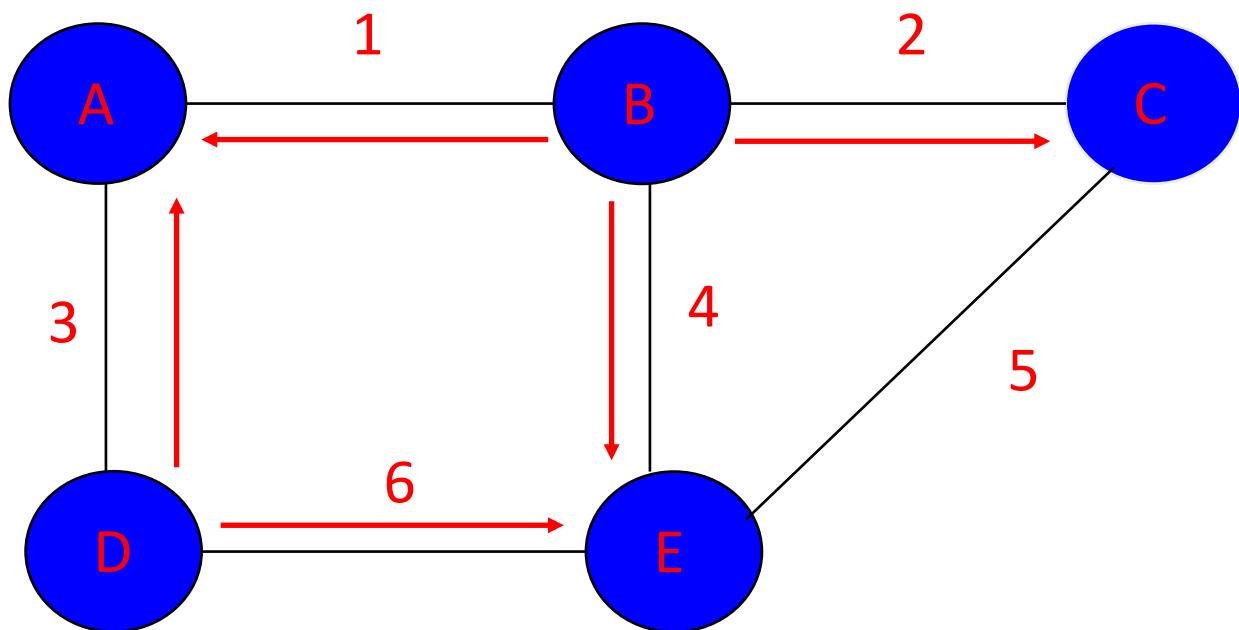
Da B verso	Link	Costo
B	locale	0
A	1	1

- Anche il nodo D aggiorna in modo analogo la sua tabella dopo aver ricevuto il DV da A



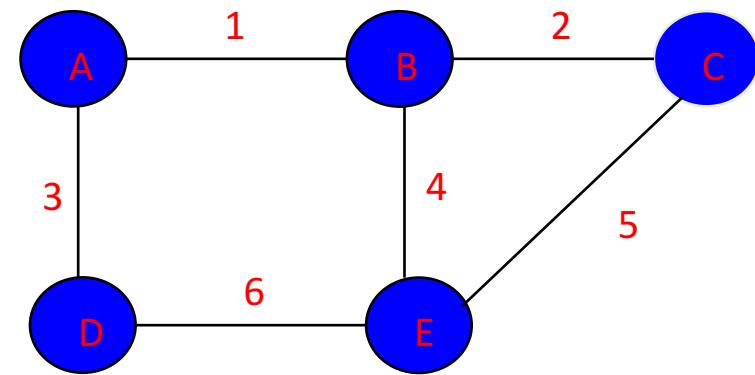
Esempio: Distance Vector (5)

- Il nodo B prepara il proprio DV
 $B=0, A=1$
e lo trasmette su tutti i *link* locali
- Anche il nodo D prepara il DV e lo invia:
 $D=0, A=1$



Esempio: Distance Vector (6)

- Il messaggio da B viene ricevuto da A,C ed E mentre quello da D è ricevuto da A ed E,
 - A riceve i due DV
 - Da B: B=0, A=1
 - Da D: D=0, A=1
- ... e aggiorna la sua tabella



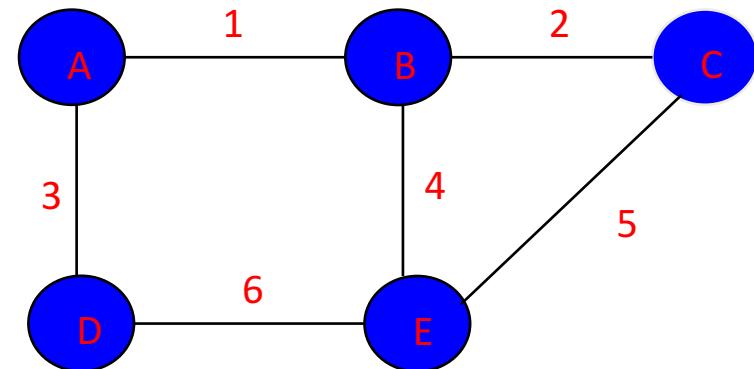
Da A verso	Link	Costo
A	local	0
B	1	1
D	3	1



Esempio: Distance Vector (7)

- Il nodo C riceve da B sul *link 2* il DV
 $B=0, A=1$
... e aggiorna la sua tabella:

Da C verso	Link	Costo
C	local	0
B	2	1
A	2	2



Esempio: Distance Vector (8)

- il nodo E riceve da B sul link 4 il DV

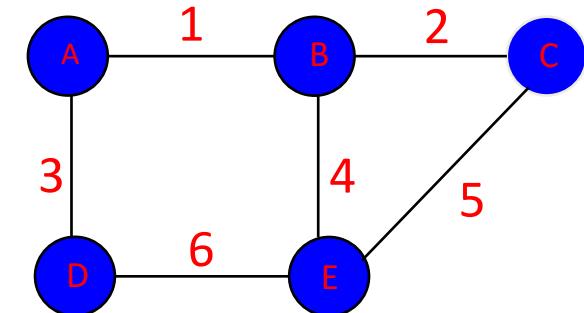
B=0, A=1

- e da D sul link 6 il DV

D=0, A=1

... e aggiorna la sua tabella di routing

- la distanza verso il nodo A utilizzando i link 4 e 6 è la stessa

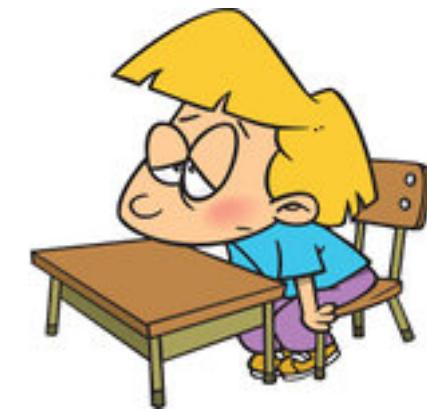
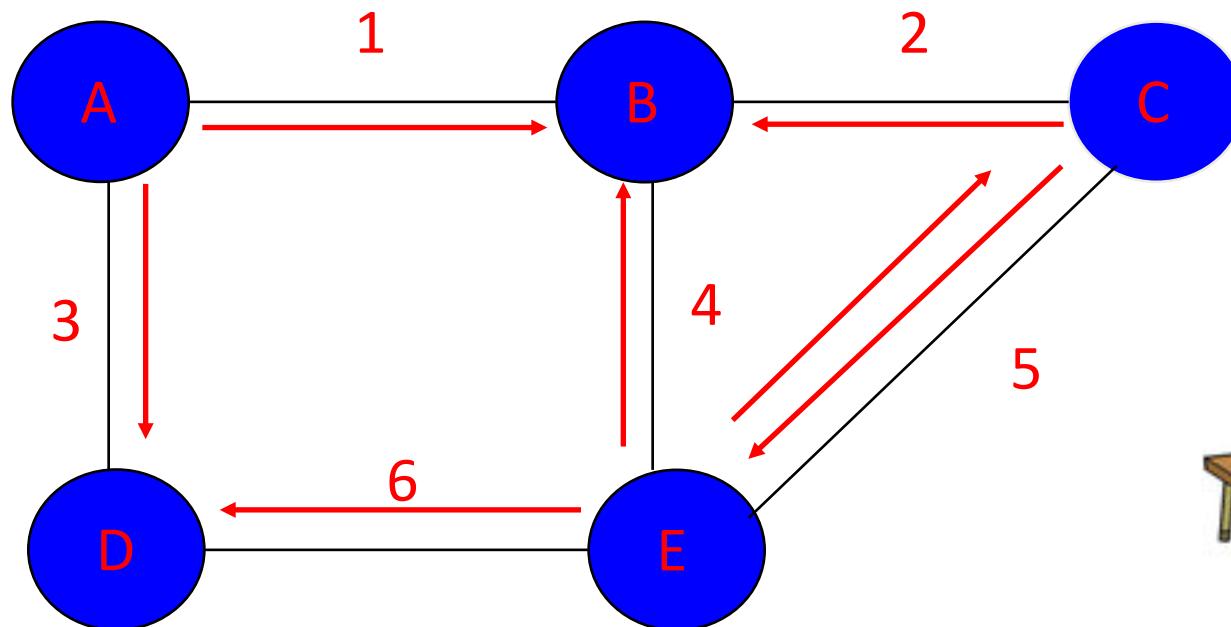


Da E verso	Link	Costo
E	local	0
B	4	1
A	4	2
D	6	1

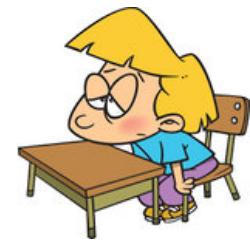


Esempio: Distance Vector (9)

- I nodi A,C ed E hanno aggiornato le proprie tabelle di *routing* e trasmettono sui *link* locali i DV aggiornati
nodo A: A=0, B=1, D=1
nodo C: C=0, B=1, A=2
nodo E: E=0, B=1, A=2, D=1



Esempio: Distance Vector (10)



Nodo B:

B	local	0
A	1	1

- A: A=0, B=1, D=1
C: C=0, B=1, A=2
E: E=0, B=1, A=2, D=1

Da B verso	Link	Costo
B	local	0
A	1	1
D	1	2
C	2	1
E	4	1

Nodo D:

D	local	0
A	3	1

- A: A=0, B=1, D=1
E: E=0, B=1, A=2, D=1

Da D verso	Link	Costo
D	local	0
A	3	1
B	3	2
E	6	1

Nodo E

E	Local	0
B	4	1
A	4	2
D	6	1

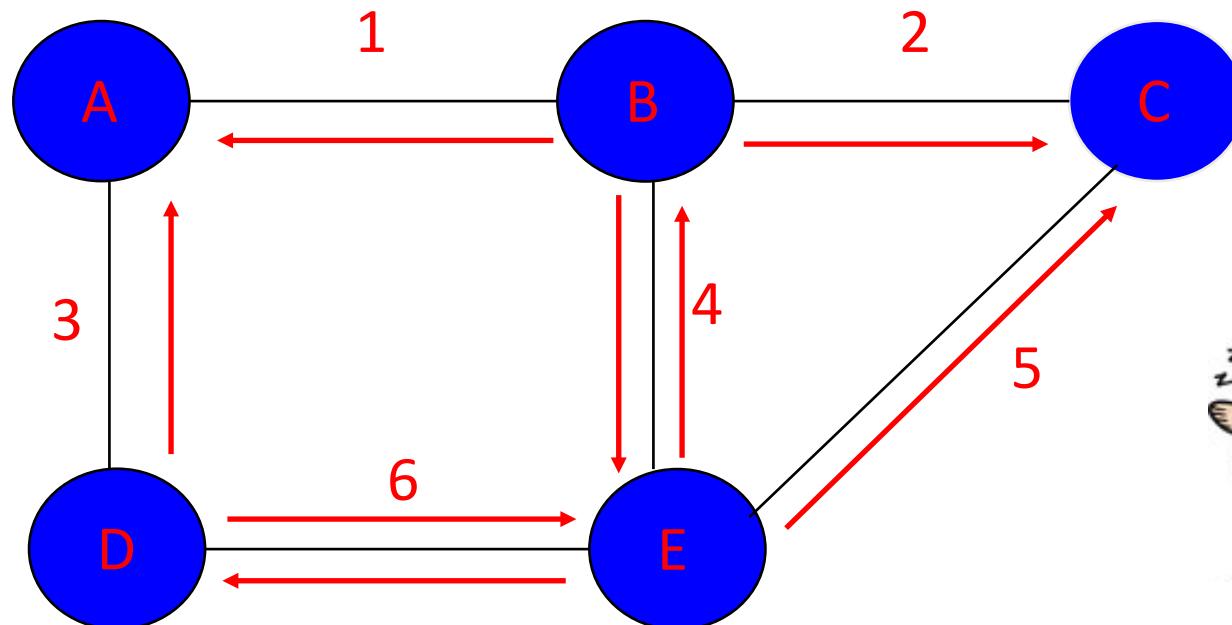
- C: C=0, B=1, A=2

Da E verso	Link	Costo
E	local	0
B	4	1
A	4	2
D	6	1
C	5	1



Esempio: Distance Vector (11)

- I nodi B,D ed E trasmettono i nuovi DV sui *link* locali
 - nodo B: B=0, A=1, D=2, C=1, E=1
 - nodo D: D=0, A=1, B=2, E=1
 - nodo E: E=0, B=1, A=2, D=1, C=1



Esempio: Distance Vector (12)



Nodo A:

A	local	0
B	1	1
D	3	1

B=0, A=1, D=2, C=1, E=1

D: D=0, A=1, B=2, E=1

Da A verso	Link	Costo
A	local	0
B		1
D		1
C		2
E		2

Nodo C:

C	local	0
B	2	1
A	2	2

B=0, A=1, D=2, C=1, E=1

D=0, A=1, B=2, E=1

E=0, B=1, A=2, D=1, C=1

Da C verso	Link	Costo
C	local	0
B		1
A		2
E		1
D		2

Nodo D

D	Local	0
A	3	1
B	3	2
E	6	1

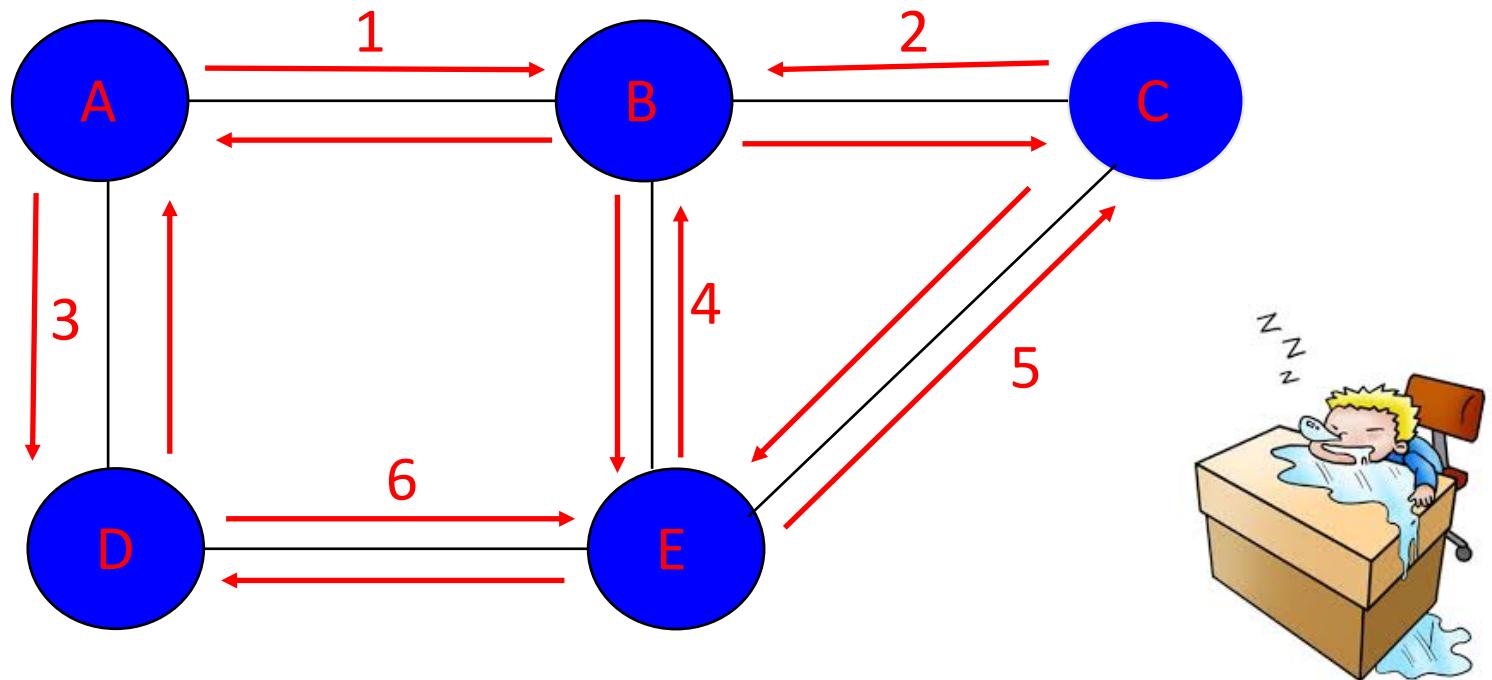
E=0, B=1, A=2, D=1, C=1

Da D verso	Link	Costo
D	local	0
A		1
B		2
E		1
C		2



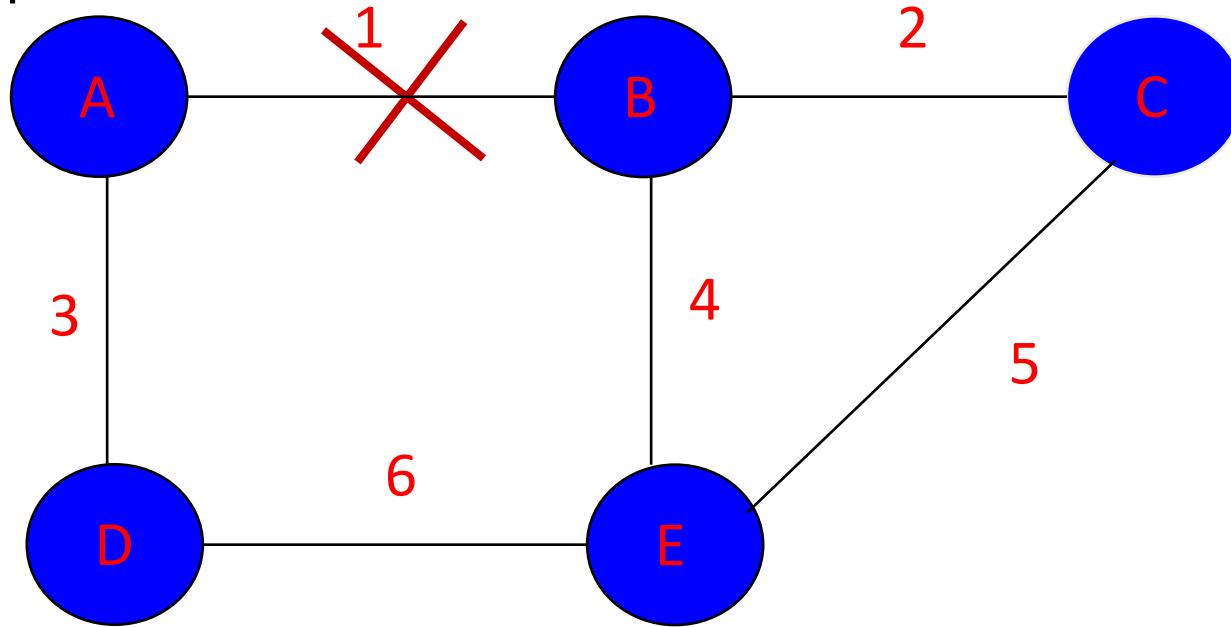
Esempio: Distance Vector (13)

- L'algoritmo è arrivato a convergenza, i nodi trasmettono i nuovi DV sui *link* che però non provocano aggiornamenti nelle tabelle di *routing* degli altri nodi



Distance Vector: rottura del link 1

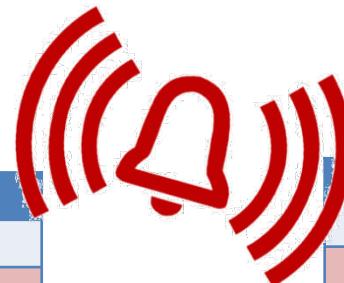
- Vediamo come le tabelle di routing si aggiornano quando si rompe il link 1



- I nodi A e B agli estremi del link 1 monitorano e riscontrano la rottura del link
- I nodi A e B aggiornano le proprie tabelle di routing assegnando costo infinito al link 1



Distance Vector: rottura del link 1



Da A verso	Link	Costo
A	local	0
B	1	1->inf
D	3	1
C	1	2->inf
E	1	2->inf

Da B verso	Link	Costo
B	local	0
A	1	1->inf
D	1	2->inf
C	2	1
E	4	1

□ Trasmettono i nuovi DV

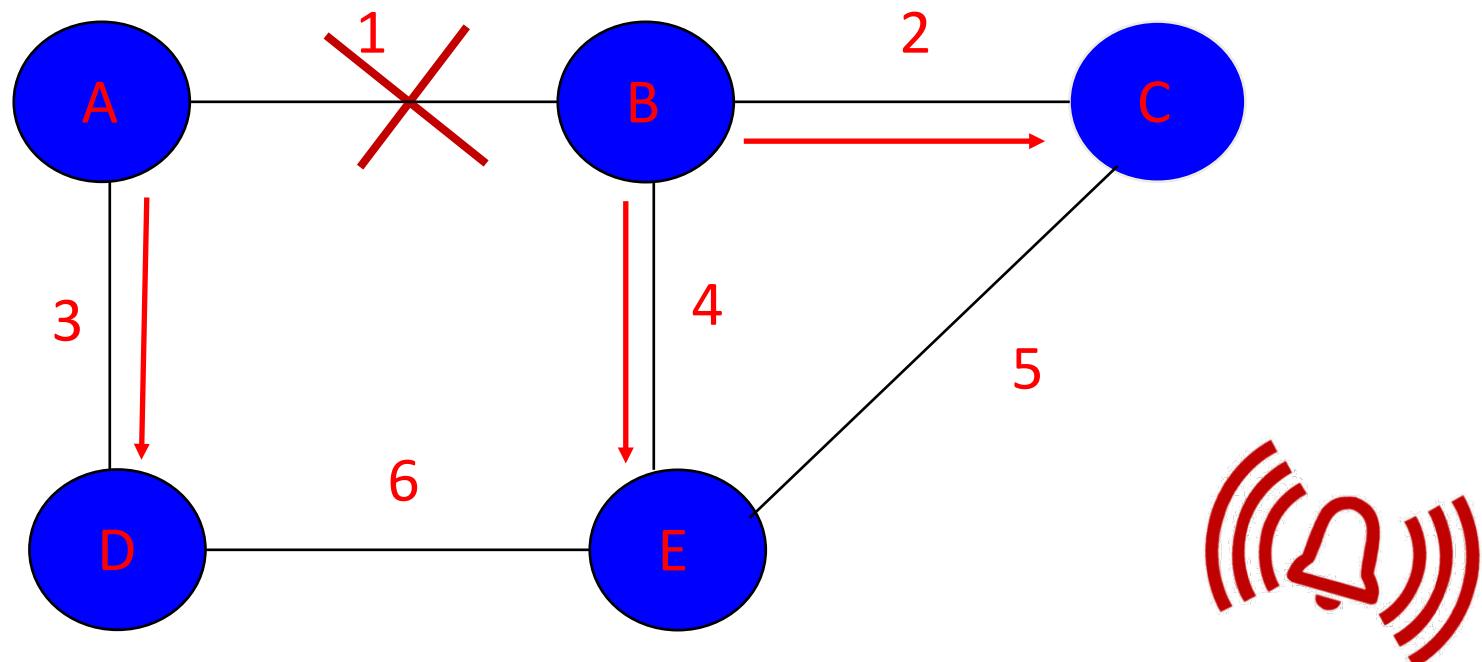
nodo A: A=0, B=inf, D=1, C=inf, E=inf

nodo B: B=0, A=inf, D=inf, C=1, E=1



Distance Vector: rottura link 1

- Il messaggio trasmesso da A viene ricevuto da D che confronta gli elementi con quelli presenti nella sua tabella di *routing*
- Tutti i costi sono maggiori o uguali a quelli presenti nella tabella, ma siccome il link da cui riceve il messaggio (link 3) è quello che utilizza per raggiungere il nodo B aggiorna la tabella



Distance Vector: rottura link 1

- Anche i nodi C ed E aggiornano le tabelle



Da D verso	Link	Costo
D	local	0
A	3	1
B	3	2->inf
E	6	1
C	6	2

Da C verso	Link	Costo
C	local	0
B	2	1
A	2	2->inf
E	5	1
D	5	2

Da E verso	Link	Costo
E	local	0
B	4	1
A	4	2->inf
D	6	1
C	5	1



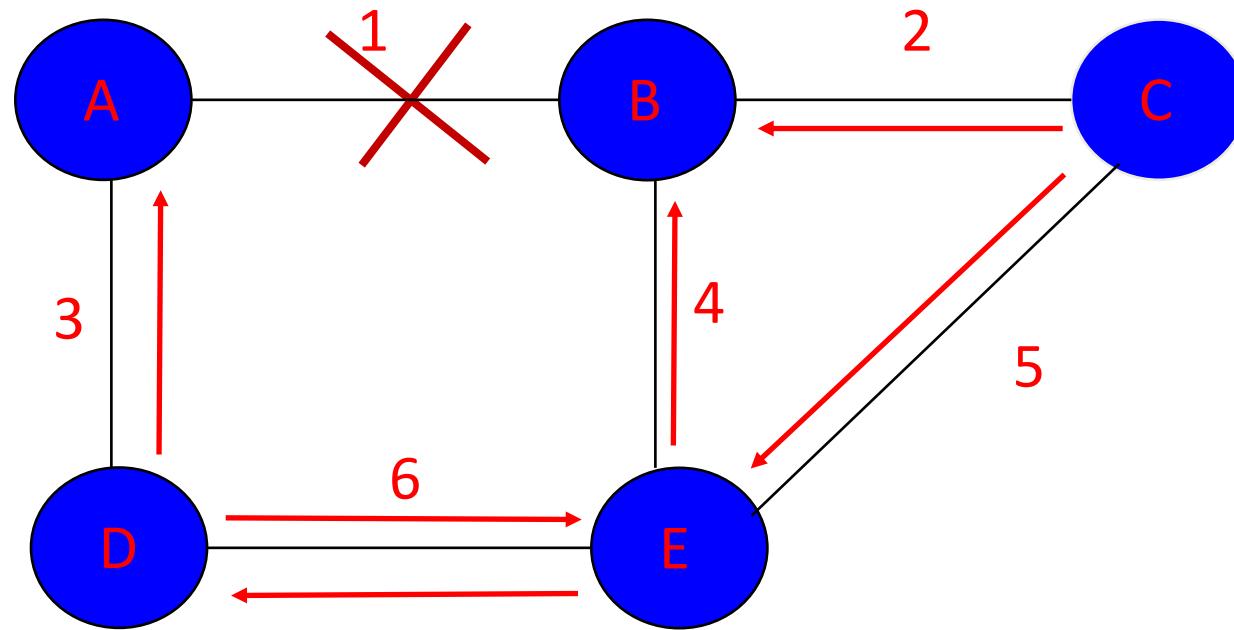
Distance Vector: rottura link 1

- I nodi D, C ed E trasmettono il loro DV

nodo D: $D=0, A=1, B=\text{inf}, E=1, C=2$

nodo C: $C=0, B=1, A=\text{inf}, E=1, D=2$

nodo E: $E=0, B=1, A=\text{inf}, D=1, C=1$



Distance Vector: rottura link 1



- questi messaggi aggiornano le tabelle di routing dei nodi A,B,D ed E

Da A verso	Link	Costo
A	local	0
B	1	inf
D	3	1
C	1->3	inf->3
E	1->3	inf->2

Da B verso	Link	Costo
B	local	0
A	1	inf
D	1->4	inf->2
C	2	1
E	4	1

Da D verso	Link	Costo
D	local	0
A	3	1
B	3->6	inf->2
E	6	1
C	6	1

Da E verso	Link	Costo
E	local	0
B	4	1
A	4->6	inf->2
D	6	1
C	5	1



Distance Vector: rottura link 1

- I nodi A,B,D ed E trasmettono i nuovi DV
 - nodo A: A=0, B=inf, D=1, C=3, E=2
 - nodo B: B=0, A=inf, D=2, C=1, D=1
 - nodo D: D=0, A=1, B=2, E=1, C=2
 - nodo E: E=0, B=1, A=2, D=1, C=1
- A, B e C si aggiornano

Da A verso	Link	Costo
A	local	0
B	1->3	inf->3
D	3	1
C	3	3
E	3	2

Da B verso	Link	Costo
B	local	0
A	inf->4	inf->3
D	4	2
C	2	1
E	4	1

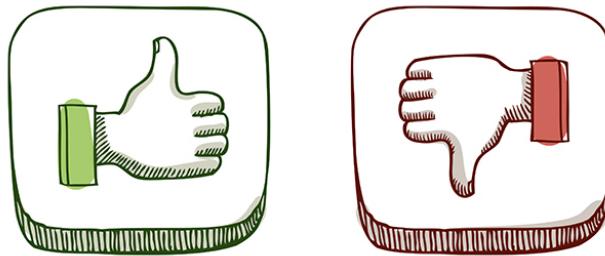
Da C verso	Link	Costo
C	local	0
B	2	1
A	2->5	inf->3
E	5	1
D	5	2

L'algoritmo è
arrivato a
convergenza !!!



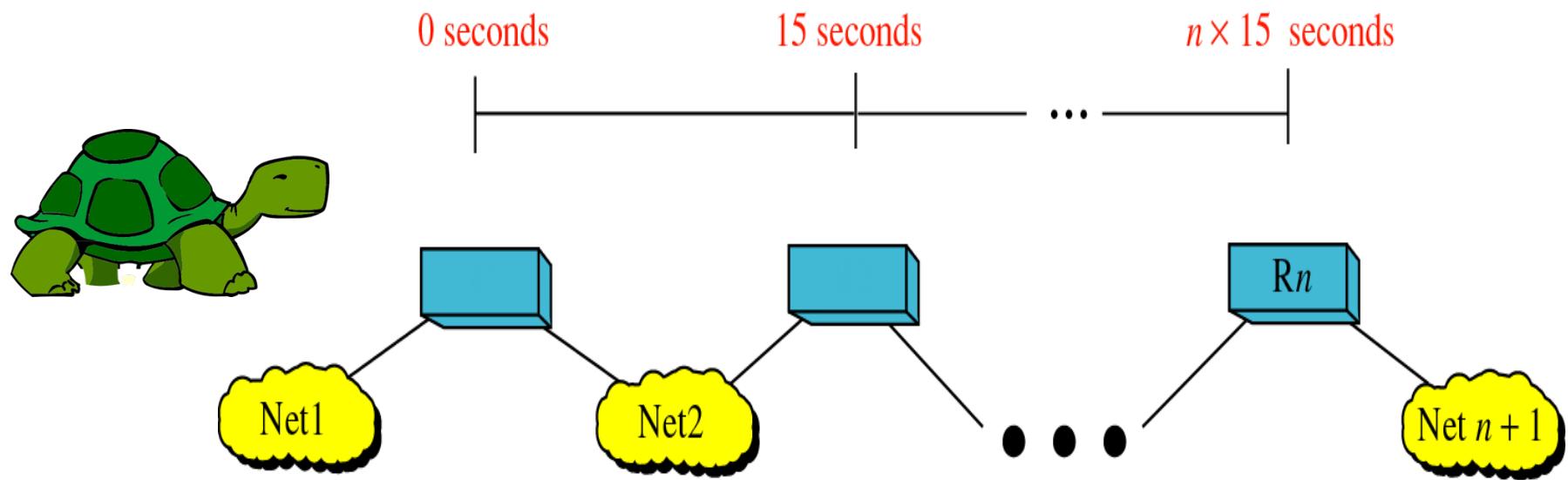
Distance Vector: caratteristiche

- **Vantaggi:**
 - molto facile da implementare
- **Svantaggi:**
 - problema della velocità di convergenza
 - limitato dal nodo più lento
 - dopo un cambiamento possono sussistere dei cicli (*loop*) per un tempo anche lungo
 - difficile mantenere comportamento stabile su reti grandi
 - problema della stabilità (*counting to infinity*)



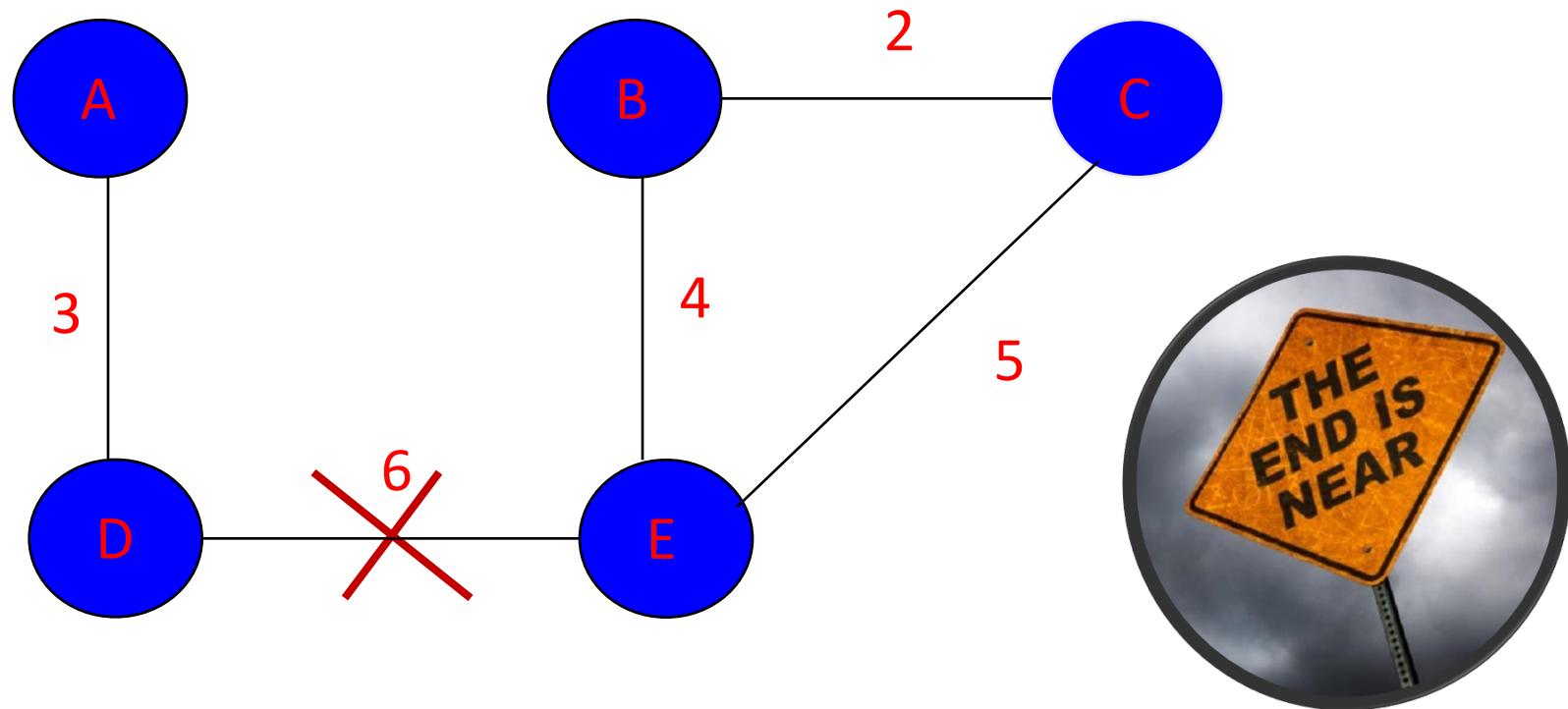
Velocità di convergenza

- Il tempo di convergenza cresce proporzionalmente con il numero di nodi



Distance Vector: *counting to infinity*

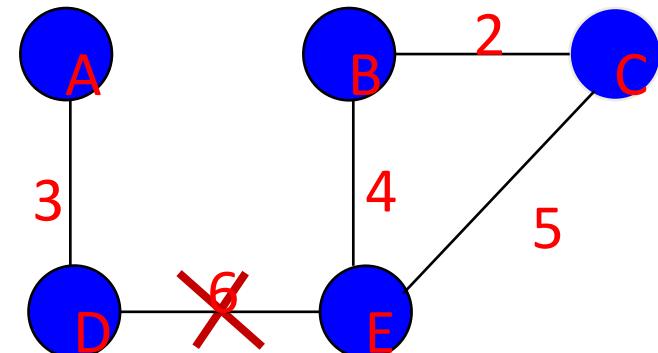
- Supponiamo che si rompa anche il *link 6*



Distance Vector: counting to infinity

- Il nodo D si accorge della rottura del *link* 6 e aggiorna la propria tabella di *routing*

Da D verso	Link	Costo
D	local	0
A	3	1
B	6	2->inf
E	6	1->inf
C	6	2->inf



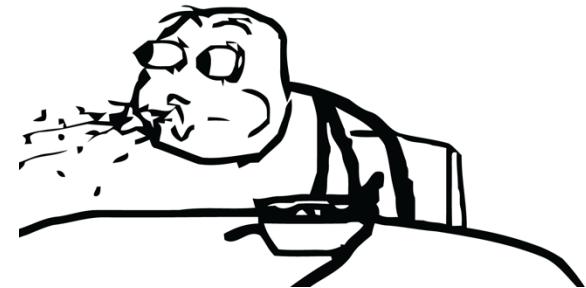
- Se il nodo D ha l'opportunità di trasmettere immediatamente il nuovo DV, il nodo A aggiorna immediatamente la sua tabella di routing e riconosce che l'unico nodo raggiungibile è D



Distance Vector: counting to infinity

- Se invece il nodo A trasmette il suo DV
nodo A: A=0, B=3, D=1, C=3, E=2
il nodo D aggiorna la sua tabella

Da D verso	Link	Costo
D	local	0
A	3	1
B	6->3	inf->4
E	6->3	inf->3
C	6->3	inf->4



- Si installa un *loop* tra i nodo A e D e non c'è modo di convergere naturalmente in uno stato stabile
- Ad ogni *step* le distanze verso i nodi B,C ed E si incrementano di 2 ➡ **counting to infinity**



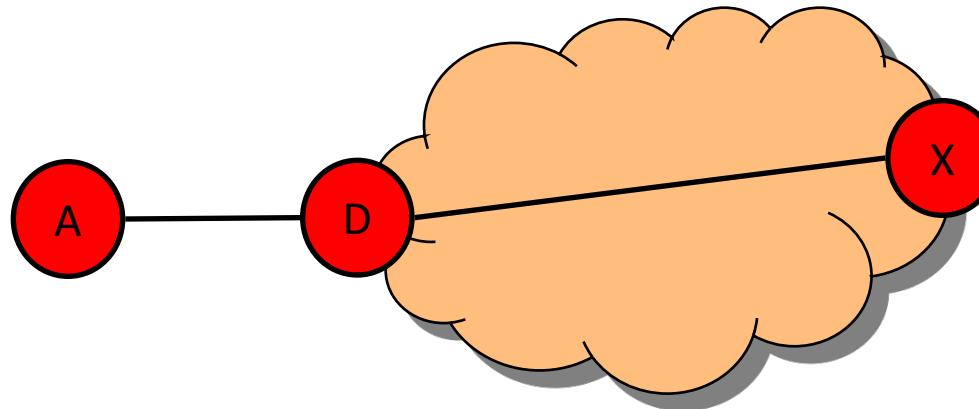
Counting to infinity: rimedi

- Hop Count Limit:
 - Il *counting to infinity* termina se si utilizza la convenzione di rappresentare l'infinito mediante un valore finito
 - Il valore deve essere maggiore del percorso più lungo nella rete
 - Quando la distanza raggiunge tale valore viene posta ad infinito e il nodo non è raggiungibile
 - Durante il periodo di *counting to infinity* la rete si trova in uno stato intermedio in cui:
 - I pacchetti sono in *loop*
 - Il *link* diventa congestionato
 - Alcuni pacchetti, compresi i messaggi di *routing* possono essere persi a causa della congestione
- 👉 La convergenza verso uno stato stabile è lenta



Counting to infinity: rimedi

- **Split-Horizon:**
 - Se il nodo A manda a D i pacchetti destinati al nodo X, non ha senso che A annuncia a D la raggiungibilità di X nel suo Distance Vector



- Il nodo A non annuncia a D con quale costo raggiunge X



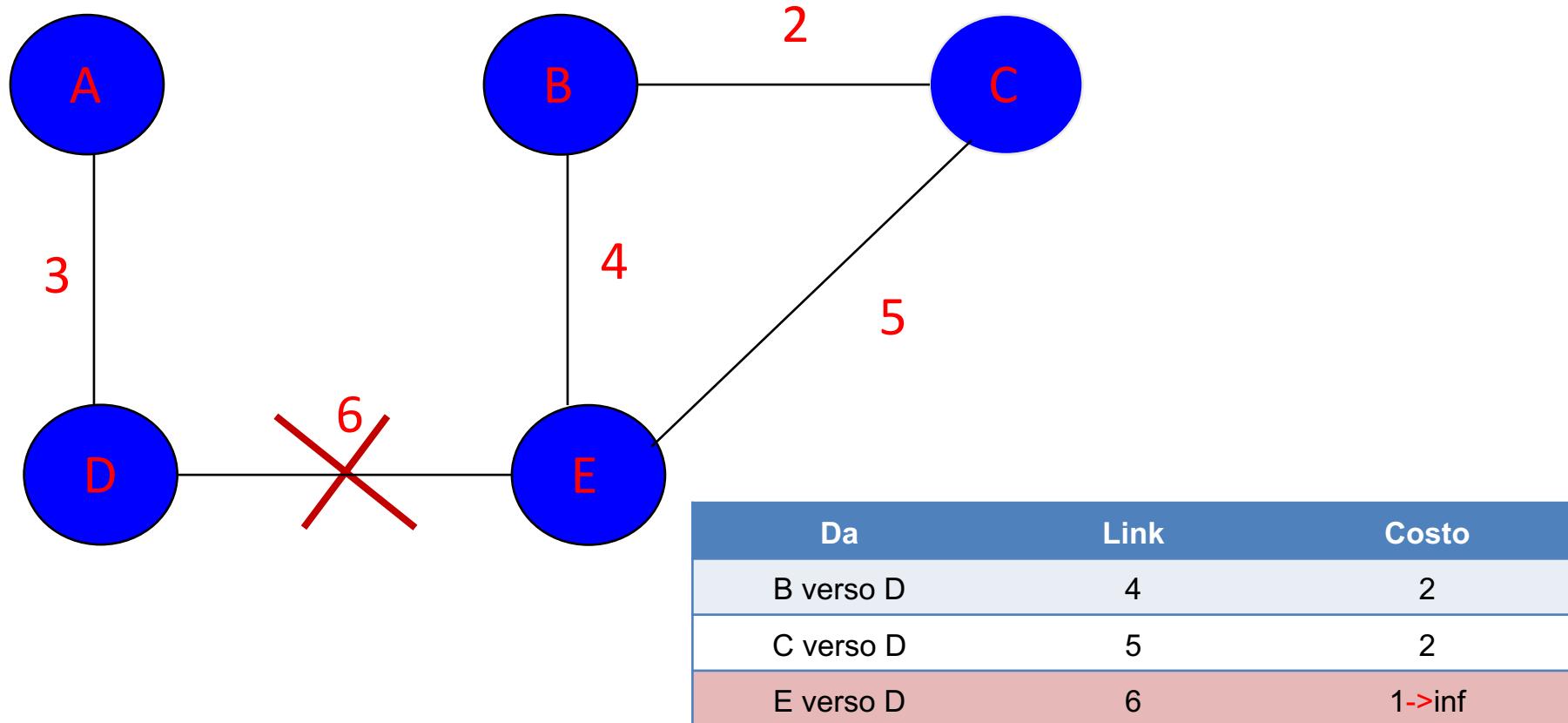
Distance Vector: Split Horizon

- Quindi il nodo A manda messaggi di *routing* diversi sui *link* locali
- *Split Horizon* esiste in due versioni:
 - **forma semplice**: il nodo omette nel messaggio ogni informazione sulle destinazioni che raggiunge tramite quel *link*
 - **con *Poisonous Reverse***: il nodo include nel messaggio tutte le destinazioni ma pone a distanza infinita quelle raggiungibili tramite quel *link*
- Non funziona con certe topologie



Distance Vector: Split Horizon

- quando il *link* 6 si rompe le tabelle di *routing* dei nodi B,C ed D contengono



Distance Vector: Split Horizon

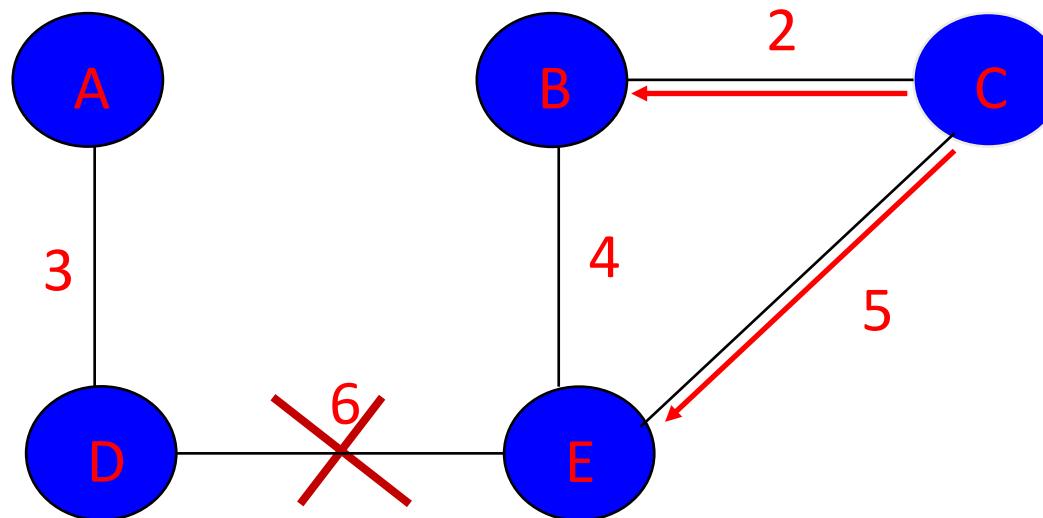
- Il nodo E comunica sui *link* 4 e 5 che la distanza da D è ora infinita
- Supponiamo che il messaggio sia ricevuto da B mentre a causa di un errore non sia ricevuto da C

Da	Link	Costo
B verso D	4	2->inf
C verso D	5	2
E verso D	6	inf



Distance Vector: Split Horizon

- Il nodo C trasmette il DV, utilizzando lo *Split Horizon* con *Poisonous Reverse* tratterà
 - al nodo E: C=0, B=1, **A=inf**, E=inf, D=inf
 - sul link 5 che vede il nodo D con costo infinito
 - al nodo B: C=0, B=inf, **A=3**, E=1, **D=2**
 - sul link 2 che vede il nodo D con costo 2



Distance Vector: Split Horizon

- Il nodo B aggiorna la sua tabella di routing e utilizzando lo *Split Horizon Poisonous Reverse* trasmette:
 - sul *link 2* che vede il nodo D con costo infinito
 - sul *link 4* che vede il nodo D con costo 3
- Nei nodi B,C ed E ora avremo

Da	Link	Costo
B verso D	4->2	inf->3
C verso D	5	2
E verso D	6->4	inf->4

- Si forma un *loop* tra i nodi B,C ed E fino a quando i valori di costo superano la soglia e sono posti ad infinito
- Si ripresenta il fenomeno di **counting to infinity**



Counting to infinity: rimedi

- Utilizzo dei contatori (Hold down)
 - Un router “bolla” una rotta come non attiva (hold down):
 - riceve un DV con distanza infinita per la rotta stessa
 - Quando non riceve il DV che segnala la rotta dal nodo del primo hop per un tempo $T_{invalid}$
 - Le rotte in hold down
 - non vengono annunciate nei DV (o annunciata con costo infinito)
 - non vengono considerati validi per essa i DV ricevuti da altri nodi con metrica peggiore rispetto a quella corrente
 - Vengono considerati validi DV migliorativi da altri router (uscita da hold down)
 - Dopo un tempo T_{flush} la rotta è cancellata



Counting to infinity: rimedi

- Il tempo tra $T_{invalid}$ e T_{flush} deve essere tarato in modo che l'informazione relativa ad un cambiamento (guasto) si propaghi nella rete
- *Triggered Update*
 - I cambiamenti di topologia sono annunciati immediatamente e distinti dagli altri
 - Aumenta la velocità di convergenza e fa scoprire prima i guasti



Agenda

- Instradamento in rete
 - Caratteristiche
 - Algoritmi su grafi
 - Algoritmo di Bellman-Ford
 - Algoritmo di Dijkstra
 - Algoritmi d'instradamento a costo minimo
 - Instradamento Distance Vector
 - Instradamento Link State

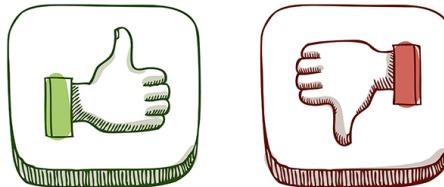


Protocolli di routing Link State

- Ogni nodo impara a conoscere i nodi e le destinazioni sue adiacenti, e le relative distanze per raggiungerle
- Ogni nodo invia a tutti gli altri nodi della rete (*flooding*) queste informazioni mediante dei *Link State Packet* (*LSP*)
- Tutti i nodi si costruiscono un database di LSP e una mappa completa della topologia della rete
- Sulla base di questa informazione vengono calcolati i cammini minimi verso tutte le destinazioni (ad esempio con Dijkstra)



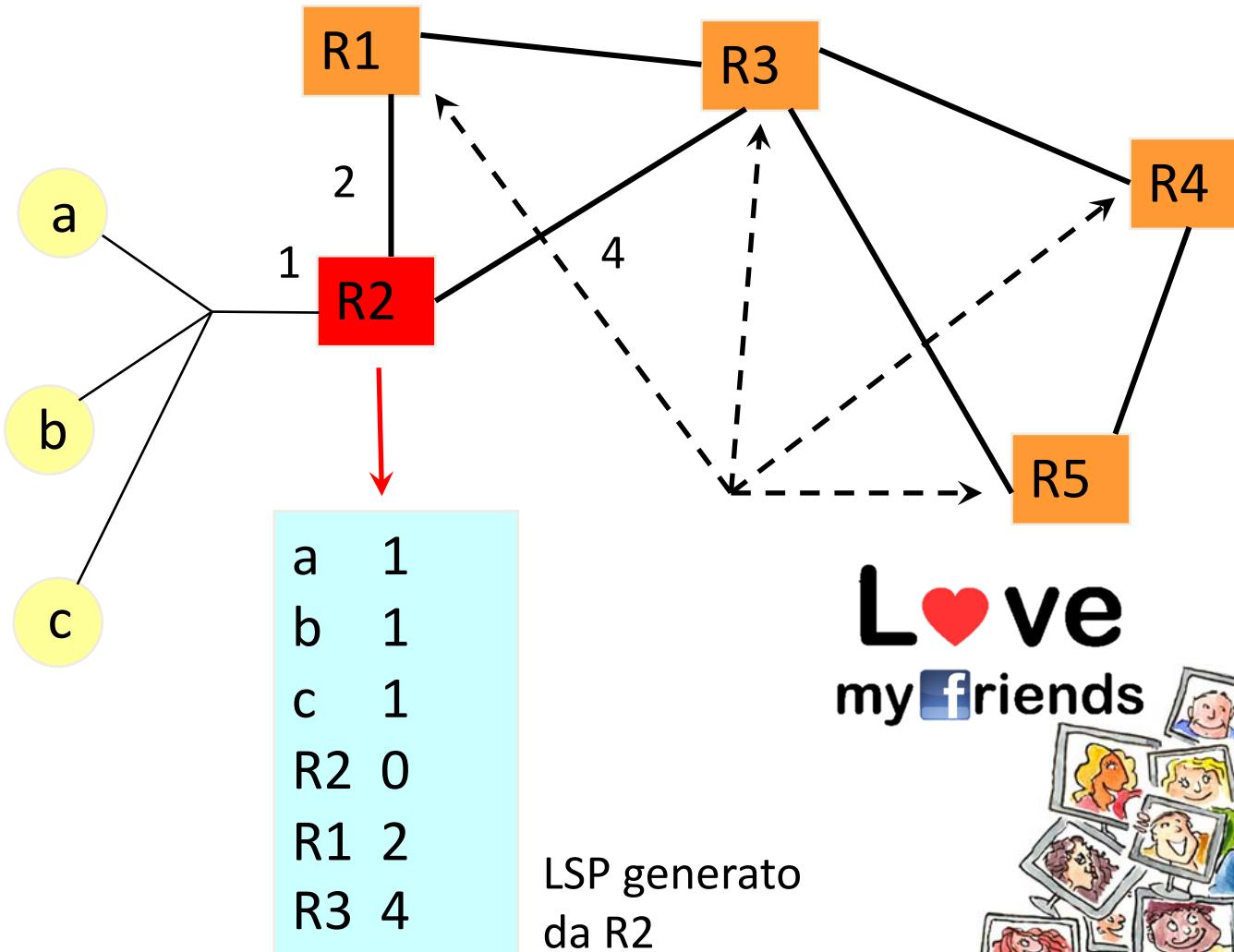
Vantaggi/Svantaggi protocolli Link State



- **Vantaggi**
 - Più flessibile in quanto ogni nodo ha una mappa completa della rete (routing ottimale)
 - Non è necessario inviare l'informazione (LSP) periodicamente ma solo dopo un cambiamento
 - Tutti i nodi vengono subito informati dei cambiamenti (in particolare topologici)
- **Svantaggi**
 - E' necessario un protocollo dedicato a mantenere l'informazione sui vicini (*Hello*)
 - E' necessario l'utilizzo del *flooding*
 - E' necessario un riscontro dei pacchetti di *routing* inviati
 - Difficile da implementare



Link State: esempio



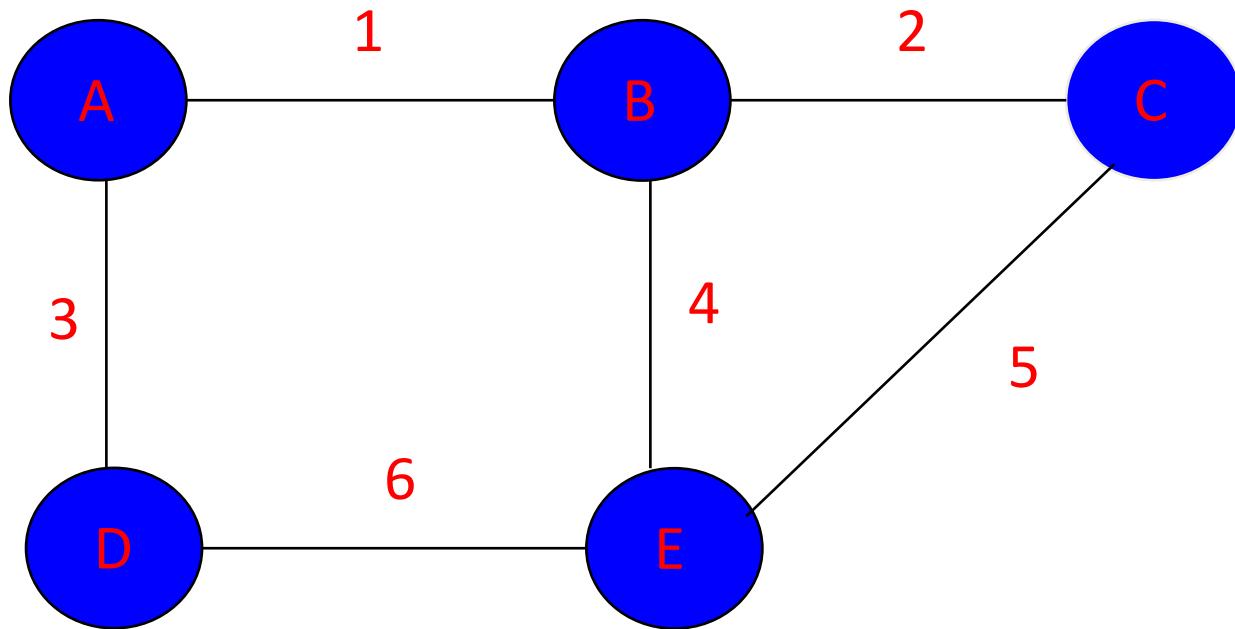
Flooding

- Ogni pacchetto in arrivo viene ritrasmesso su tutte le uscite eccetto quella da cui è stato ricevuto
- Occorre prevenire i *loop* e la conseguente generazione incontrollata di traffico
- Rimedi
 - Numero di sequenza (SN) + database degli SN ricevuti in ogni nodo: i pacchetti non vengono ritrasmessi una seconda volta
 - Contatore di *hop* (come TTL di IP)



Esempio: Link State

- Ogni nodo ha un database (archivio degli LSP) in cui è descritta una mappa della rete



Esempio: Link State

- La rete è rappresentata dal database

Da	Verso	Link	Costo	Sequence Number
A	B	1	1	1
A	D	3	1	1
B	A	1	1	1
B	C	2	1	1
B	E	4	1	1
C	B	2	1	1
C	E	5	1	1
D	A	3	1	1
D	E	6	1	1
E	B	4	1	1
E	C	5	1	1
E	D	6	1	1



- Ogni nodo può calcolare il percorso più breve verso tutti gli altri nodi



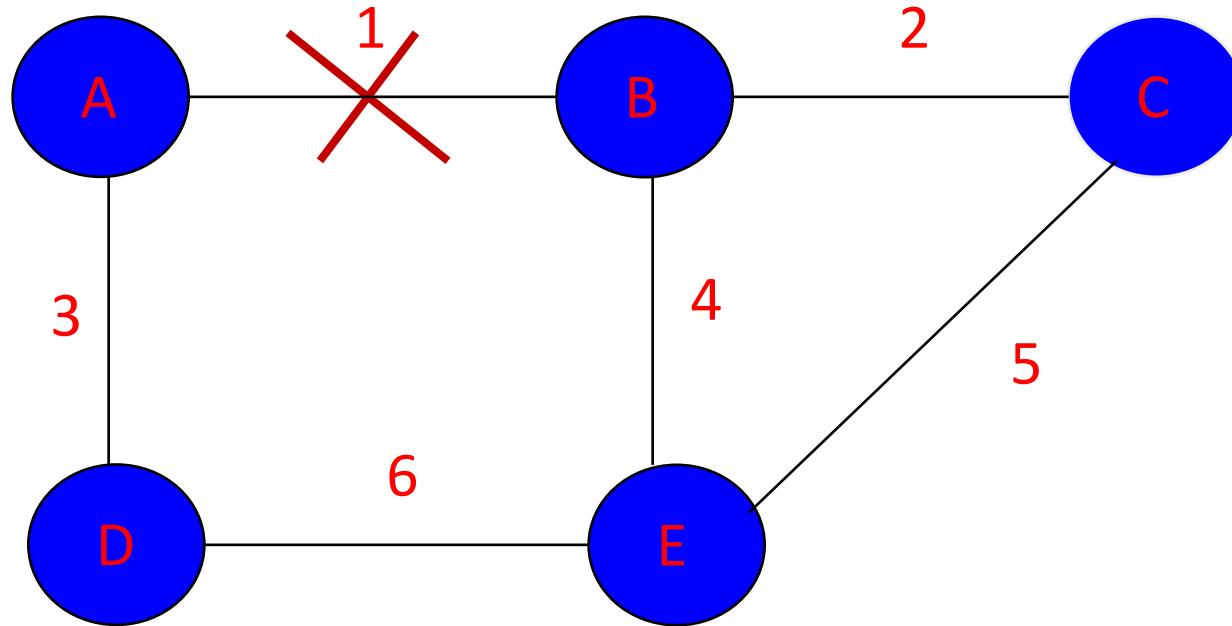
All'arrivo di un LSP

- Se il LSP non è mai stato ricevuto, o il SN è superiore a quello memorizzato precedentemente:
 - memorizza il LSP
 - lo ritrasmette in *flooding* sulle uscite
- Se il LSP ha lo stesso SN di quello memorizzato
 - non fa nulla
- Se il LSP è più vecchio di quello memorizzato
 - trasmette quello più recente al mittente



Esempio: Link State

- Il protocollo di *routing* deve aggiornare il database quando la rete cambia



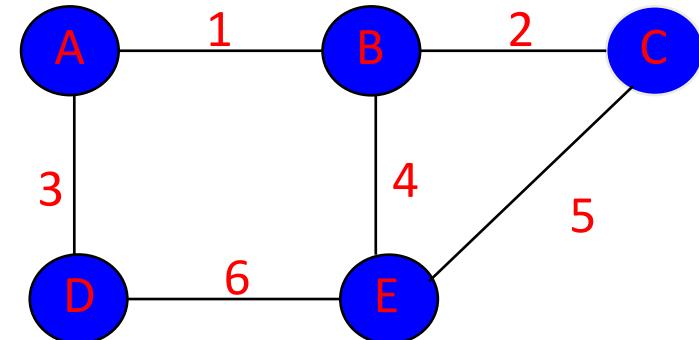
- La rottura del *link* 1 viene riscontrata dai nodi A e B che aggiornano il proprio database e mandano un messaggio di *update* sui *link* 2, 3 e 4
 - nodo A: Da A, Verso B, Link 1, Costo=inf, Number=2
 - nodo B: Da B, Verso A, link 1, Costo= inf, Number=2



Esempio: Link State

- I messaggi sono ricevuti dai nodi D,E ed C che aggiornano il proprio database e li trasmettono sui *link* locali
- Il nuovo database dopo il *flooding*

Da	Verso	Link	Costo	Sequence Number
A	B	1	1->inf	1->2
A	D	3	1	1
B	A	1	1->inf	1->2
B	C	2	1	1
B	E	4	1	1
C	B	2	1	1
C	E	5	1	1
D	A	3	1	1
D	E	6	1	1
E	B	4	1	1
E	C	5	1	1
E	D	6	1	1



Instradamento a minima distanza

Osservazioni

- Algoritmi distance vector e link state convergono alla stessa soluzione in condizioni statiche
- Possono essere implementati sia in forma centralizzata che in forma distribuita
- **Algoritmo Distance-Vector** (Dialoga con *vicini* informandoli sulla raggiungibilità di *tutti* i nodi)
 - Ha convergenza più lenta in condizioni dinamiche
 - Ogni nodo deve conoscere solo ciò che vedono i suoi vicini
- **Algoritmo Link State** (Dialoga con *tutti* i nodi informandoli sulla raggiungibilità dei *vicini*)
 - Nella forma distribuita, ogni nodo deve “vedere” l’intera rete

