



**La Sapienza**

Università degli Studi di Roma

Dipartimento di Informatica e Sistemistica

# RETI DI CALCOLATORI II

## Routing protocols - Overview

**Emiliano Trevisani**

**[trevisani@dis.uniroma1.it](mailto:trevisani@dis.uniroma1.it)**

**A.A. 2008/2009**

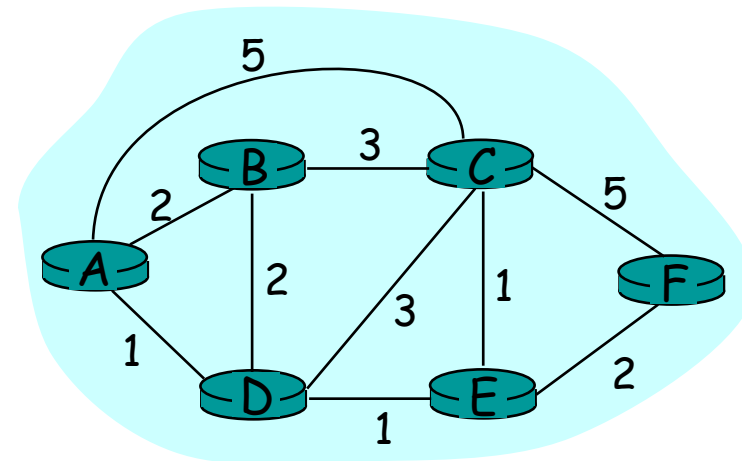
# Obiettivi

- Descrivere approcci e meccanismi base
- Descrivere gli aspetti chiave in fase implementativa
- Analizzare due proposte adottate in pratica:
  - RIP
  - OSPF (più in dettaglio)
- Studio di alcuni aspetti di dettaglio a cura dello studente [vedi riferimenti]

# Routing (instradamento)

## Protocollo di Routing

Obiettivo: determinare “buon”  
cammino sorg.-dest.



Astrazione usando grafi:

- I nodi rappresentano I router
- Gli archi rappresentano le sottoreti
  - Costi sugli archi: ritardo, costo in €, livello di congestione
  - Caso particolare: link P2P

- Cammino “buono”:
  - Di solito significa cammino a costo minimo
  - Possibili def. alternative

# Classificazione degli algoritmi di routing

## Info globale vs distribuita

Globale:

- Tutti i nodi hanno info completa su grafo e costi
- Algoritmi “link state”

Distribuita:

- Il router conosce i nodi vicini e i costi degli archi verso essi
- Processo di calcolo iterativo e distribuito; scambio di informazione con i vicini
- Algoritmi “distance vector”

## Statico o dinamico

Statico:

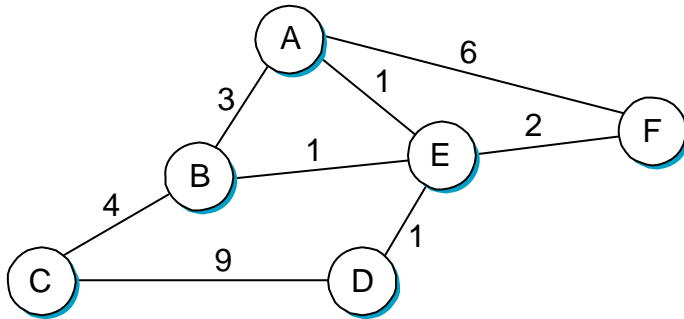
- I cammini cambiano lentamente nel tempo

Dinamico:

- I cammini mutano rapidamente
  - Aggiornamenti periodici
  - A seguito di cambiamenti nei costi dei link

# Tabella di instradamento (forwarding o routing) table

- Costo per raggiungere una data destinazione
- Prossimo router nel percorso d'instradamento
- Informazioni di base
  - Tabelle reali sono piu' complesse

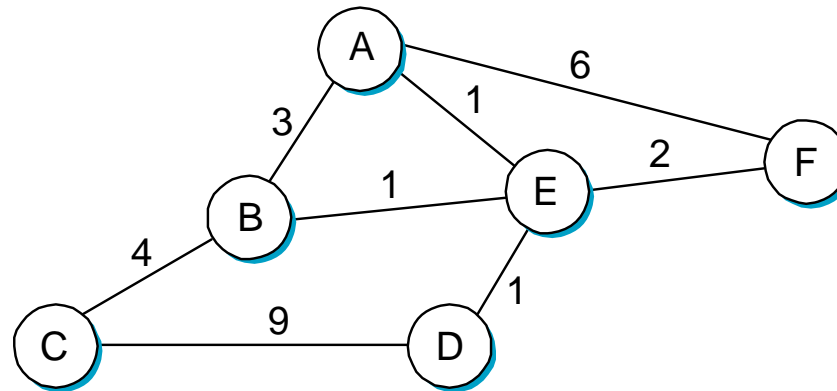


	Link in uscita (outgoing), da usare, costo
A	E,2
C	C,4
D	E,2
E	E,1
F	E,3

Tabella di routing di B

# Overview

- Forwarding vs Routing
  - forwarding: to select an output port based on destination address and routing table
  - routing: process by which routing table is built



- Factors
  - static: topology
  - dynamic: load
- In sostanza:
  - il routing e' l'individuazione dei cammini minimi e la costruzione delle tabelle di instradamento
  - Il forwarding e' l'inoltro di un pacchetto da parte di un router a un suo vicino in base alla destinazione del pacchetto, usando l'informazione memorizzata nelle tabelle di instradamento

# Distance Vector Algorithms – overview

- Base algorithm: Bellman –Ford (distributed version)
- Each node maintains a set of triples
  - (Destination, Cost, NextHop)
- Directly connected neighbors exchange updates
  - periodically (on the order of several seconds)
  - whenever table changes (called *triggered* update)
- Each update is a list of pairs:
  - (Destination, Cost)
  - Se nodo A riceve (C, 10) da B (deve essere un suo vicino) significa che il costo per arrivare a C passando per A e'  $10 + \text{costo}(A, B)$
- Update local table if receive a “better” route
  - smaller cost
  - came from next-hop
- Refresh existing routes; delete them if they time out

# Dynamic evolution

- Evoluzione al nodo A

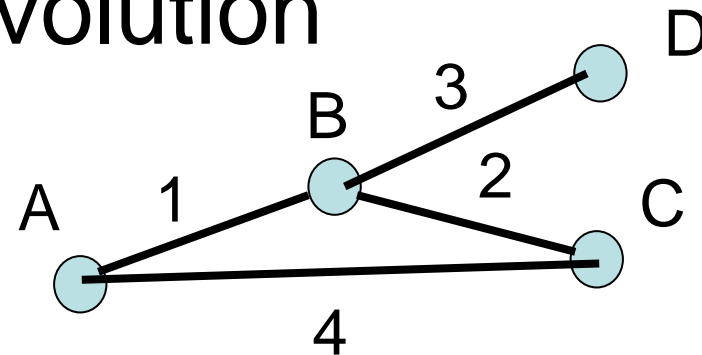


Tabella iniziale di A

Dest	Cost	NextHop
B	1	B
C	4	C
D	inf	-

Update da B

Dest	Cost
C	2
D	3

*Manca riga relativa ad A*

Update da C

Dest	Cost
B	2

*Manca riga relativa ad A*

Dopo update da B

Dest	Cost	NextHop
B	1	B
C	3	B
D	4	B

# Algoritmo Distance Vector Routing

## Iterativo:

- Continua finché non si scambia più info.
- *self-terminating*: non esiste “segnale” di stop

## Asincrono:

- I nodi non eseguono le operazioni in passi sincroni!

## Distribuito:

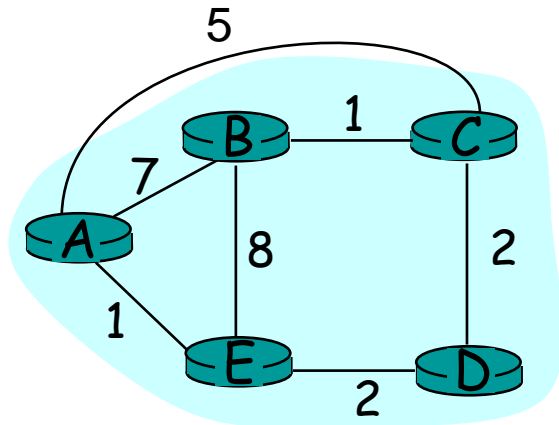
- Ogni nodo comunica *solo* con i vicini immediati

Distance Table (tabella delle distanze, presente in ogni nodo)

- una riga per ogni destinazione
- una colonna per ogni vicino del nodo
- Esempio: in nodo X, per la dest. Y passando per Z:

$$\begin{aligned} D^X(Y,Z) &= \text{Distanza da X a} \\ &= Y \text{ passando per Z} \\ &= c(X,Z) + \min_w \{D^Z(Y,w)\} \end{aligned}$$

# Tabella delle distanze: esempio



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

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

$$= 2+2 = 4$$

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

$$= 2+3 = 5 \text{ Ciclo!}$$

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

$$= 8+6 = 14 \text{ Ciclo!}$$

# La tabella delle distanze dà la tabella di routing

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

	Link in uscita (outgoing), da usare, costo
A	A,1
B	D,5
C	D,4
D	D,2

Destinazione

Tabella delle distanze



Tabella di routing

# Distance Vector Routing: generalità

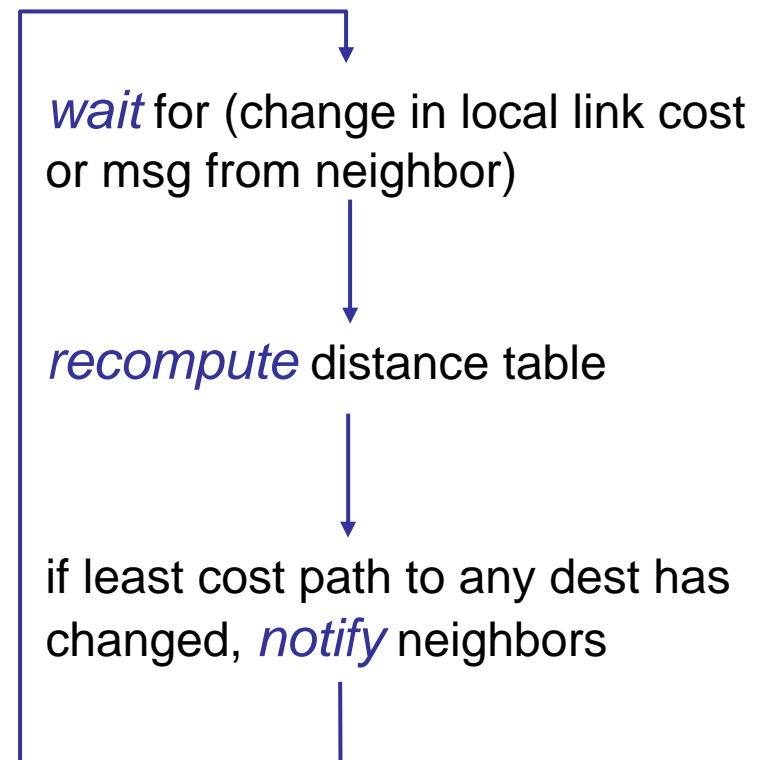
Iterative, asynchronous:  
each local iteration caused  
by:

- 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:



# Algoritmo Distance Vector:

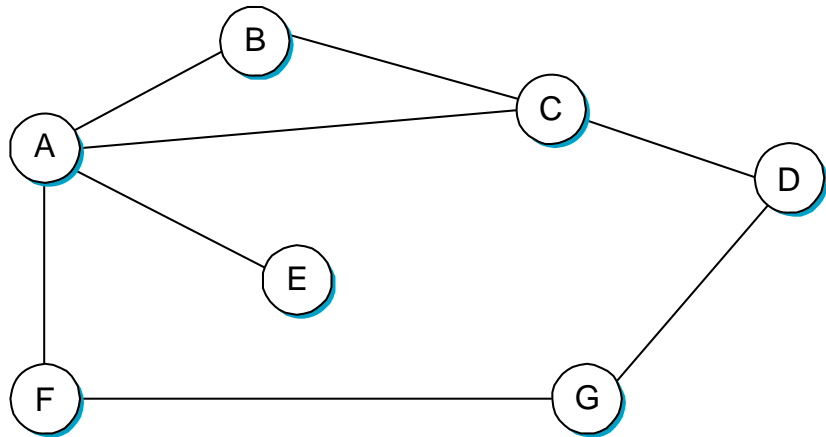
In ogni nodo X:

- 1 inizializzazione:
- 2 per ogni nodo v adiacente:
- 3  $D^X(*,v) = \text{infinito}$  /\* "\*" significa "per ogni riga" \*/
- 4  $D^X(v,v) = c(X,v)$
- 5 per ogni destinazione y
- 6 invia  $\min_w D^X(y,w)$  a ogni vicino /\* w su tutti i vicini di X \*/

# Distance Vector Algorithm (cont.):

```
8 loop
9  wait (finché il costo del link verso un vicino V cambia
10      o si riceve un aggiornamento da un vicino V)
11
12  if (c(X,V) cambia di una quantità d)
13      /* cambia i costi per tutte le dest. via v di d */
14      /* nota: d può essere positivo o negativo */
15      per ogni destinazione y:  $D^X(y,V) = D^X(y,V) + d$ 
16
17  else if (aggiornamento da V rispetto alla destinazione Y)
18      /* camm. minimo da V a qualche Y cambiato */
19      /* V ha inviato nuovo valore per  $\min_w D^V(Y,w)$  */
20      /* sia "newval" il nuovo valore */
21      per la generica destinazione y:  $D^X(Y,V) = c(X,V) + \text{newval}$ 
22
23  if esiste nuovo min  $D^X_w(Y,w)$  verso una destinazione Y
24      invia nuovo valore di  $\min_w D^X(Y,w)$  a ogni vicino
25
26 forever
```

# Example - Table at B



**Esempio completo su libro  
Di Peterson e Davie**

**Initial**

Destination	Cost	NextHop
A	1	A
C	1	C
D	$\infty$	-
E	$\infty$	-
F	$\infty$	-
G	$\infty$	-

**Final**

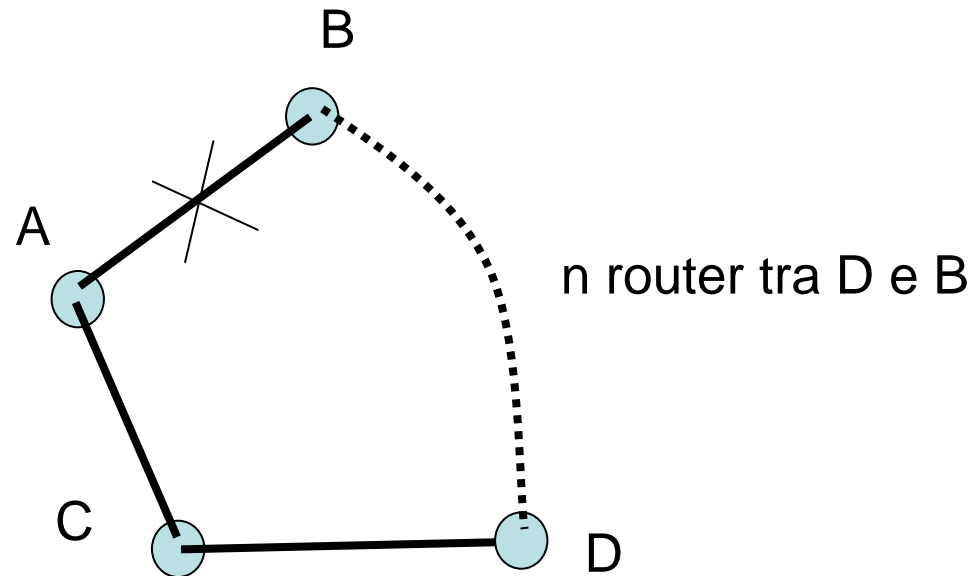
Destination	Cost	NextHop
A	1	A
C	1	C
D	2	C
E	2	A
F	2	A
G	3	A

# Routing Loops

- Costi unitari. Link (A, B) cade, costo da A a B  $\rightarrow$  inf.

Tabella di A

Dest	Cost	NextHop
B	1	B
C	1	C
D	2	C
.....		



- A riceve update periodico da C prima di inviare proprio
- A aggiorna riga relativa a B (riga diventa (B 3 C)) e invia update a C
- Processo prosegue finche' il costo da C a B passando per D diventa piu' basso di quello stimato passando per A

# Loop-Breaking Heuristics

- Algoritmo converge ma cio' puo' avvenire lentamente
- Rimedi
  - Set infinity to 16
  - Split horizon
  - Split horizon with poison reverse
    - Non inviare a un vicino aggiornamenti causati da un suo aggiornamento
  - Non funzionano sempre

# Algoritmi link state

- Algoritmo di base: Dijkstra
  - Ogni router deve conoscere l'intera topologia della rete (connettività e metrica dei rami)
- Aspetti implementativi
  - Raccolta delle informazioni
    - Flooding
  - Metriche di costo
- Un protocollo di routing link state: OSPF
  - Algoritmo di Dijkstra
  - Protocollo per lo scambio di informazioni di stato
  - Aspetti avanzati
    - Instradamento gerarchico
    - Scalabilità -> route summarization

# Algoritmo di Dijkstra/cont.


- Topologia, costi archi noti a tutti i vertici
  - Si può fare con “link state broadcast”
  - Tutti i nodi hanno stessa info
- Calcolo cammini minimi da un nodo (“sorgente”) a tutti gli altri
  - Dà tabella di routing per il nodo
- Iterativo: dopo  $k$  iterazioni, cammini minimi verso  $k$  destinazioni

## Notazione:

- $c(i,j)$ : costo arco tra nodo  $i$  e  $j$ .
- $D(v)$ : ultimo valore calcolato costo da sorg. a dest.  $V$
- $p(v)$ : predecessore di  $v$  nel cammino minimo corrente dalla sorgente a  $v$
- $N$ : insieme dei nodi il cui cammino minimo dalla sorgente è definitivamente noto

# Algoritmo di Dijkstra/cont.

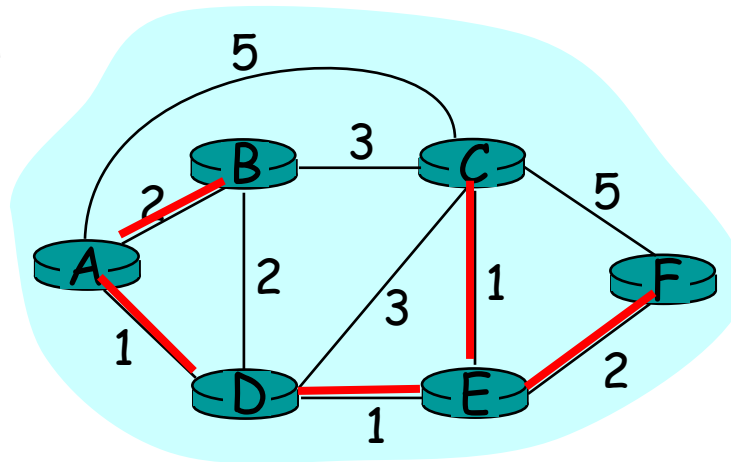
```
1 Inizializzazione:  
2 N = {A}  
3 for all nodes v  
4   if v adjacent to A  
5     then  $D(v) = c(A,v)$   
6     else  $D(v) = \text{infty}$   
7  
8 loop  
9   find w not in N such that  $D(w)$  is a minimum  
10  add w to N  
11  update  $D(v)$  for all v adjacent to w and not in N:  
12     $D(v) = \min( D(v), D(w) + c(w,v) )$   
13    /* new cost to v is either old cost to v or known  
14     shortest path cost to w plus cost from w to v */  
15 until all nodes in N
```



# Algoritmo di Dijkstra: esempio

Passo	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
→ 2	ADE	2,A	3,E			4,E
→ 3	ADEB		3,E			4,E
→ 4	ADEBC					4,E
5	ADEBCF					

Alla fine si ottiene  
Tabella di routing



Attenzione: ogni  
sotto-cammino di un  
cammino minimo è  
minimo

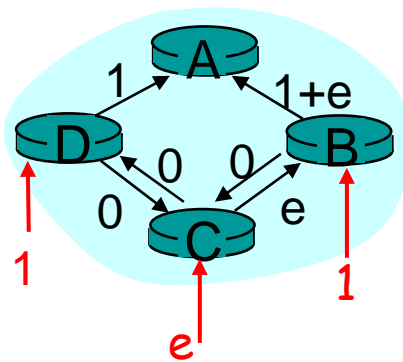
# Caratteristiche dell'algoritmo

Complessità:  $n$  nodi

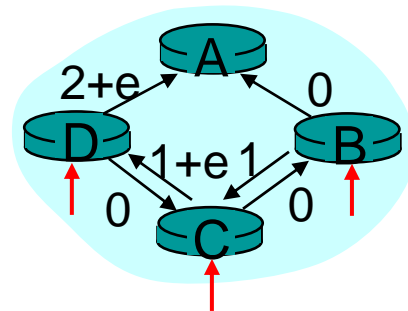
- Ogni iterazione: controllo di tutti i nodi  $w$  non in  $N$
- $n*(n+1)/2$  confronti:  $O(n^2)$
- Possibili implementazioni più efficienti:  $O(n \log n)$

Possibili oscillazioni:

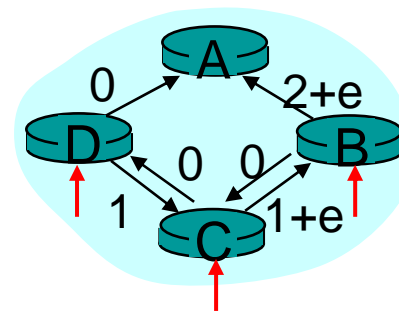
- es., costo link = quantità di traffico trasportato
  - Perché questa metrica?



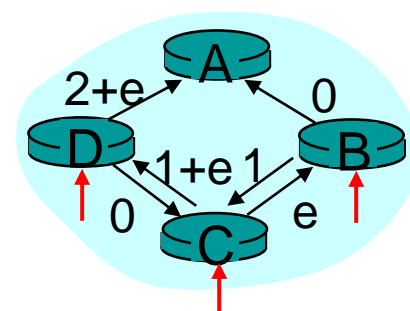
Inizialmente



ricalcolo  
routing



... ricalcolo



... ricalcolo

# Aspetti implementativi

Strategie per la raccolta delle  
informazioni di stato

# Link State

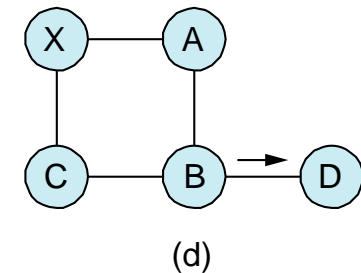
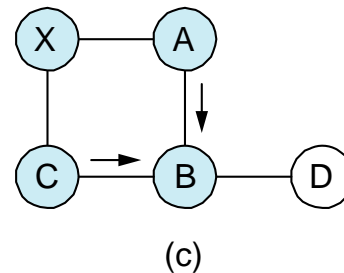
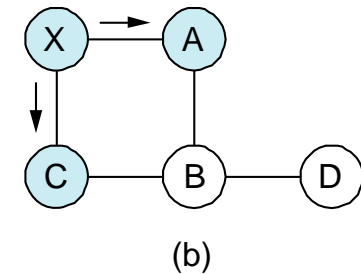
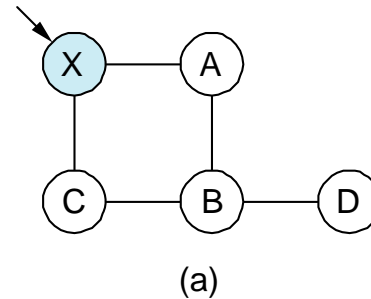
- Algoritmo di Dijkstra richiede conoscenza completa del grafo a ogni router
- Metodo: Reliable flooding
  - Un nodo invia link-state info a tutti i vicini
  - Un nodo che riceve info di stato la inoltra ai vicini
  - Aggiornamenti periodici
  - Uso di TTL per eliminare informazione di stato obsoleta

# Link State Packets

- ID del nodo che lo ha generato
  - Lista (vicino, costo link)
  - SEQNO
  - TTL del pacchetto
- 
- Generazione LSP
    - Scadenza timer
    - Cambio nella topologia/costi dei link

# Link State Packets

- **Reliable flooding**
  - store most recent LSP from each node
  - forward LSP to all nodes except one that sent it
  - generate new LSP periodically
    - increment SEQNO
  - start SEQNO at 0 when reboot
  - decrement TTL of each stored LSP
    - discard when TTL=0



Esempio: LSP proveniente da X e ricevuto da B

$(X, ((C, 10), (A, 2)), 15, 5)$

ID    Vicino    Costo    SEQNO    TTL

# Algoritmo Link State

- All' avvio o a ogni cambiamento: si invia un LSA (Link State Advertisement)
  - LSA contiene un LSP con i link state relativi a tutti i link aventi un estremo nel router
- I router si scambiano link state mediante flooding
- Appena il DB di un router e' completo: calcola Shortest Path Tree verso tutti gli altri router
- Costruisci tabella di routing:
  - Generica entry: (dest, cost, next hop)
- Aggiornamento in corrispondenza di modifiche