RETI DI CALCOLATORI

Autunno 2018

docente: Laura Ricci

laura.ricci@unipi.it

Lezione 4:
IL LIVELLO IP:
ROUTING DISTANCE VECTOR
04/10/2018

parte di queste slides sono
ricavati da slides pubblicate in corsi
universitari tenuti da colleghi
italiani e stranieri

MATERIALE DIDATTICO

- Forouzan
  - Paragrafo 1.4
  - Paragrafo 4.3.2 (Routing basato su vettori distanza)

L'ALGORITMO DI BELLMANN FORD

- Link based routing: problema principale:
  - scalabilità, tutti devono avere una visione globale della rete
- Approccio alternativo:
  - distance vector basati su una implementazione distribuita (peer to peer!) dell'algoritmo di Bellman-Ford
  - protocollo corrispondente: RIP (Routing Internet Protocol)
- L'algoritmo di Dijkstra
  - difficile una implementazione distribuita
  - ad ogni passo ha bisogno dell'intera frontiera dell'albero dei cammini minimi, per individuare il nodo più vicino da espandere al passo successivo
- Bellman-Ford, più adatto ad una implementazione distribuita

BELLMANN FORD SEQUENZIALE

- calcola
  - il costo del cammino minimo da un vertice sorgente s a tutti i restanti vertici nel grafo
  - un albero di copertura di cammini minimi (shortest path tree) con radice s.
- i pesi degli archi possono essere negativi
  - non interessante, nel caso dei protocolli di routing solo valori positivi
- può essere utilizzato per determinare se il grafo contiene un ciclo con peso negativo. In tal caso, non esiste uno shortest path tree e l'algoritmo segnala questa situazione
- costo computazionale: O(|V||E|)
  - |V| iterazioni, ad ogni iterazione si considerano tutti gli archi

L'ALGORITMO DI BELLMANN FORD

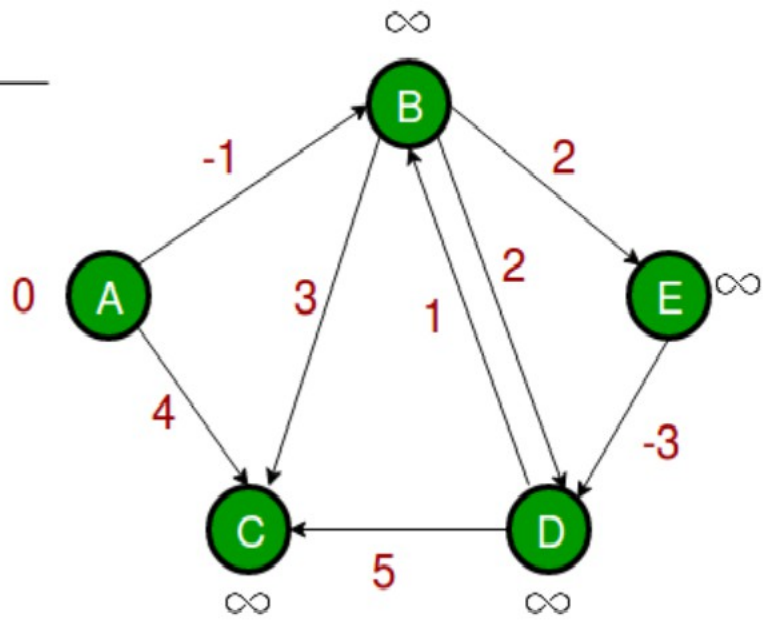
s: nodo sorgente

d_v: stima della distanza del nodo sorgente da un qualsiasi altro nodo v

- Initialize $d_s := 0$ and $d_v := \infty$ for all $v \neq s$
- For $k = 1, \ldots, n-1$:
  - For each edge $u \to v$ of cost c:
    - $d_v := \min(d_v, d_u + c)$ ← ciclo principale
- For each edge $u \to v$ of cost c:
  - If $d_v > d_u + c$:
    - Then the graph contains a negative-weight cycle

L'ALGORITMO DI BELLMANN FORD

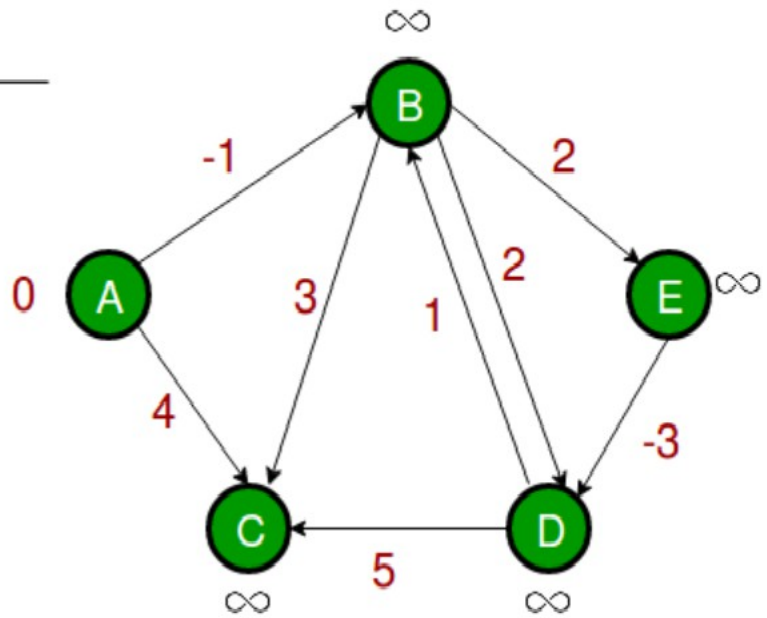
	A	B	C	D	E
Iterazione 0	0	∞	∞	∞	∞



- supponiamo che la sorgente sia il nodo A
- si vuole calcolare la distanza dal nodo A a tutti gli altri nodi del grafo
  - tutte le distanze sono inizializzate ad infinito, ad eccezione della distanza del vertice 0 con se stesso

L'ALGORITMO DI BELLMANN FORD

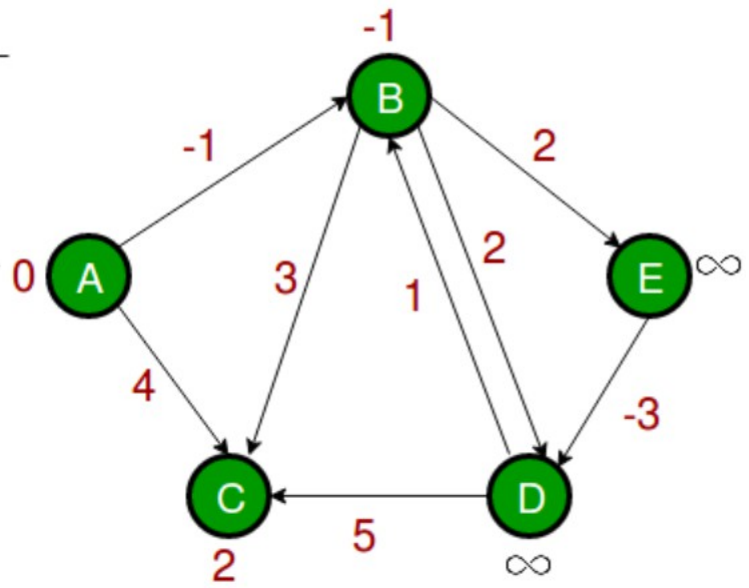
	A	B	C	D	E
Iterazione 0	0	∞	∞	∞	∞



- supponiamo che la sorgente sia il nodo A
- l'algoritmo effettua |V|-1, iterazioni, dove V è il numero di vertici del grafo
- ad ogni iterazione vengono considerati tutti gli archi del grafo, in un ordine qualsiasi, e si migliora la stima della distanza verso gli altri nodi

L'ALGORITMO DI BELLMANN FORD

	A	B	C	D	E
Iterazione 0	0	∞	∞	∞	∞
Iterazione 1	0	-1	∞	∞	∞
	0	-1	4	∞	∞
	0	-1	2	∞	∞



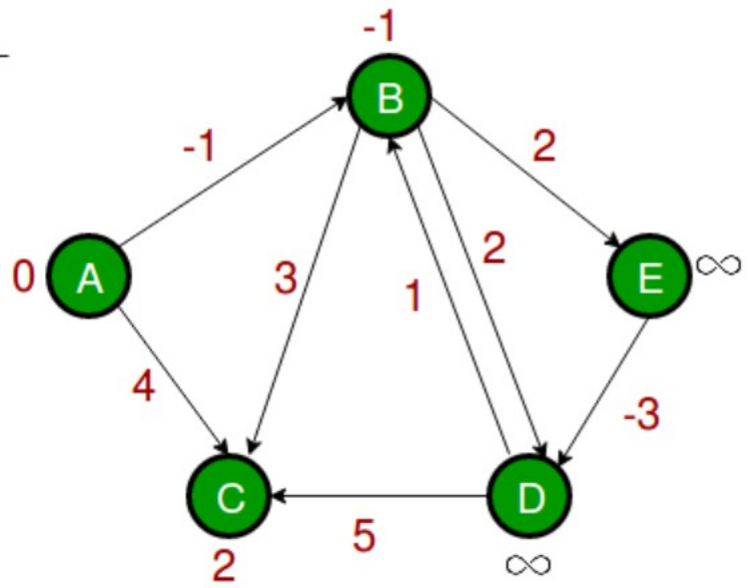
Iterazione 1:

- dobbiamo ipotizzare un ordine con cui gli archi vengono analizzati

- supponiamo che gli archi vengano visitati sempre nel seguente ordine:
  - (B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D).

L'ALGORITMO DI BELLMANN FORD

	A	B	C	D	E
Iterazione 0	0	∞	∞	∞	∞
Iterazione 1	0	-1	∞	∞	∞
	0	-1	4	∞	∞
	0	-1	2	∞	∞

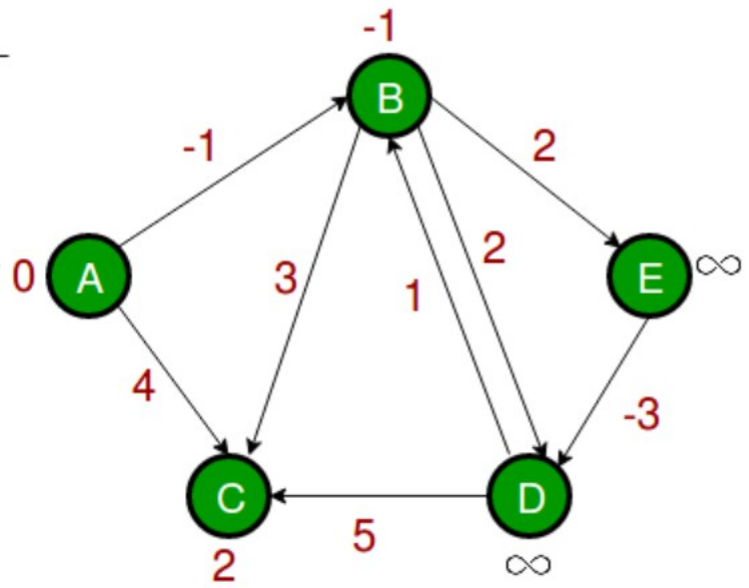


Iterazione 1:

- ordine visita degli archi:(B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D).
- tutti i cammini minimi che contengano al massimo un arco (AB) (AC), vengono sicuramente individuati nella prima iterazione
- notare che, anche alcuni cammini di lunghezza maggiore possono venir analizzati, Questo dipende dall'ordine di visita degli archi

L'ALGORITMO DI BELLMANN FORD

	A	B	C	D	E
Iterazione 0	0	∞	∞	∞	∞
Iterazione 1	0	-1	∞	∞	∞
	0	-1	4	∞	∞
	0	-1	2	∞	∞

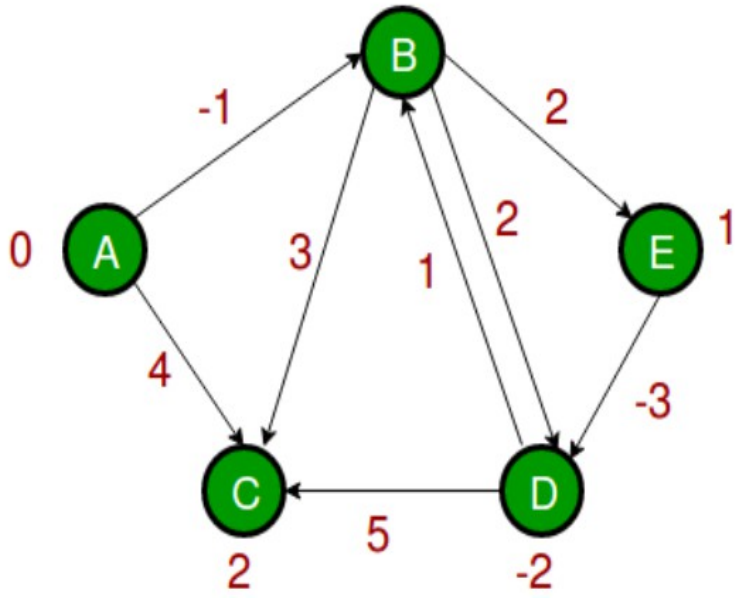


Iterazione 1:

- Ordine visita degli archi:(B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D).
- notare che, anche alcuni cammini di lunghezza maggiore possono venir analizzati, Questo dipende dall'ordine di visita degli archi
- Quando visito BC ho già una stima della distanza da A a B quindi posso produrre una nuova stima della distanza da A a C. Questo non accade, ad esempio per E

L'ALGORITMO DI BELLMANN FORD

	A	B	C	D	E
Iterazione 0	0	∞	∞	∞	∞
Iterazione 1	0	-1	∞	∞	∞
	0	-1	4	∞	∞
	0	-1	2	∞	∞
Iterazione 2	0	-1	2	∞	1
	0	-1	2	1	1
	0	-1	2	-2	1



Iterazione 2:

- Ordine visita degli archi:(B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D).

- La soluzione rimane stabile durante le terza e la quarta iterazione, quindi quella rappresentata in figura è la soluzione finale

BELLMANN FORD: IMPLEMENTAZIONE JAVA

```
// A class to represent a connected, directed and weighted graph
class Graph
  {  class Edge {
        int src, dest, weight;
        Edge( )
             {  src = dest = weight = 0;  }
          };
    int V, E;
    Edge edge [ ] ;
    // Creates a graph with V vertices and E edges
    Graph(int v, int e)
    {   V = v;
        E = e;
        edge = new Edge[e];
        for (int i=0; i<e; ++i)
            edge[i] = new Edge();
    }
```

BELLMANN FORD: IMPLEMENTAZIONE JAVA

```
void BellmanFord(Graph graph,int src)
    {
        int V = graph.V, E = graph.E;
        int dist[] = new int[V];

        // Step 1: Initialize distances from src to all other
        // vertices as INFINITE
        // dist from the source as 0

        for (int i=0; i<V; ++i)
            dist[i] = Integer.MAX_VALUE;
        dist[src] = 0;
```

BELLMANN FORD: IMPLEMENTAZIONE JAVA

```
// Step 2: Relax all edges |V| - 1 times. A simple
// shortest path from src to any other vertex can
// have at-most |V| - 1 edges

        for (int i=1; i<V; ++i)
        {
            for (int j=0; j<E; ++j)
            {
                int u = graph.edge[j].src;
                int v = graph.edge[j].dest;
                int weight = graph.edge[j].weight;
                if (dist[u]!=Integer.MAX_VALUE && dist[u]+weight<dist[v])
                    dist[v]=dist[u]+weight;
            }
        }
```

BELLMANN FORD: IMPLEMENTAZIONE JAVA

```
// Step 3: check for negative-weight cycles.  The above
        // step guarantees shortest distances if graph doesn't
        // contain negative weight cycle. If we get a shorter
        //  path, then there is a cycle.
        for (int j=0; j<E; ++j)
        {
            int u = graph.edge[j].src;
            int v = graph.edge[j].dest;
            int weight = graph.edge[j].weight;
            if (dist[u] != Integer.MAX_VALUE &&
                dist[u]+weight < dist[v])
              System.out.println("Graph contains negative weight cycle");
        }
        printArr(dist, V);
    }
```

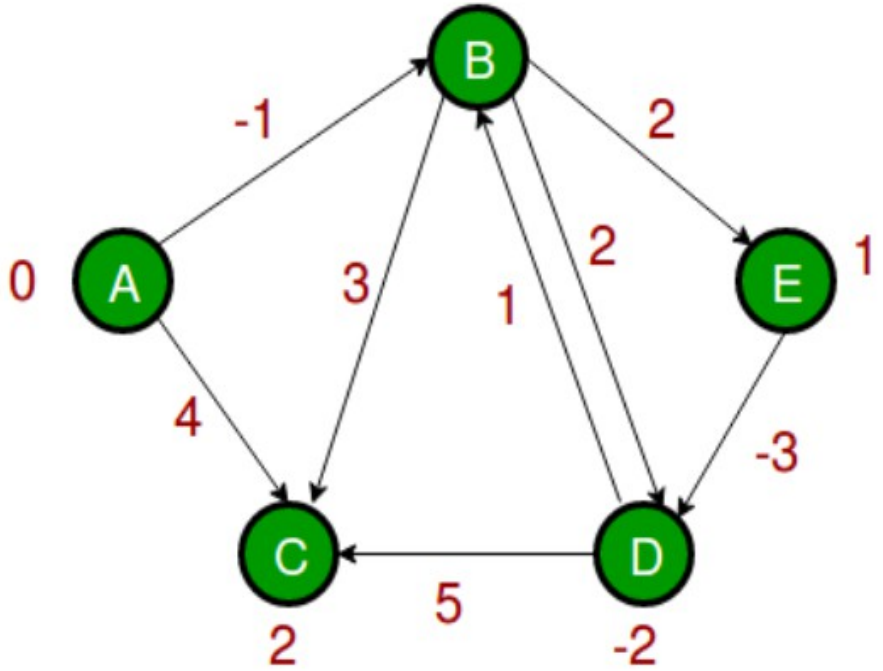
BELLMANN FORD: IMPLEMENTAZIONE JAVA

```
// A utility function used to print the solution
    void printArr(int dist[], int V)
    { System.out.println("Vertex   Distance from Source");
      for (int i=0; i<V; ++i)
            System.out.println(i+"\t\t"+dist[i]); }
// Driver method to test above function
    public static void main(String[] args)
    {   int V = 5;  // Number of vertices in graph
        int E = 8;  // Number of edges in graph
        Graph graph = new Graph(V, E);
        // add edge 0-1 (or A-B in the figure)
        graph.edge[0].src = 0;
        graph.edge[0].dest = 1;
        graph.edge[0].weight = -1;
        // add edge 0-2 (or A-C in the figure)
        …....
        graph.BellmanFord(graph, 0); } }
```

BELLMANN FORD: IMPLEMENTAZIONE JAVA

Risultato ottenuto:

```
Vertex   Distance from Source

0                0

1               -1

2                2

3               -2

4                1
```



DIJKSTRA - BELLMANN FORD

- differenze principali
  - l'algoritmo di Dijkstra itera sulla lunghezza del percorso.
  - l'algoritmo di Bellman-Ford itera sul numero di archi attraversati da un percorso minimo.

- Bellman-Ford può essere più facilmente eseguito in un sistema distribuito.

  Mentre Dijkstra cerca il valore minimo su tutti i vertici, Bellman-Ford considera gli archi, uno alla volta ed in ordine arbitrario.
  - qualsiasi ordine considerato, porta a convergenza.
  - Il cammino minimo può al massimo toccare tutti i vertici
  - Anche se l'ordinamento che ho stabilito, considera gli archi del cammino dall'ultimo al primo, dopo |V| iterazioni lo individuo sicuramente.

- i progettisti di Internet hanno utilizzato una implementazione distribuita dell'algoritmo di Bellmann Ford
  - il protocollo è uno standard di Internet
  - definito in RFC 2453

RFC 2453: DISTANCE VECTOR/ BELLMANN FORD

If it is possible to get from entity i to entity j directly, then a cost, d(i,j), is associated with the hop between i and j. The cost is infinite if i and j are not immediate neighbors. Let D(i,j) represent the metric of the best route from entity i to entity j.

Then, the best metric must be described by

$$D(i,i) = 0, \text{ forall } i$$

$$D(i,j) = \min_{k} [d(i,k) + D(k,j)], \text{ otherwise}$$

The algorithm:

Entity i gets its neighbors k to send it their estimates of their distances to the destination j. When i gets the estimates from k, it adds d(i,k) to each of the numbers. This is simply the cost of traversing the network between i and k. Now and then i compares the values from all of its neighbors and picks the smallest. It can be proven that this algorithm will converge to the correct estimates of D(i,j) in finite time in the absence of topology

changes.

REQUEST FOR COMMENT

- ma..cosa è una Request for comment?
- standard e protocolli sono la base della definizione della rete
- Inizialmente specifiche meno formali, poi necessaria maggior standardizzazione
- risultato attuale: un insieme molto complesso di standard
  - alcuni ancora in uso
  - alcuni mai stati utilizzati
  - alcuni non più in uso
- Processo gestito mediante RFC (Request for Comment)
  - idea: utilizzare la rete stessa per sviluppare e diffondere gli standard che governano la rete stessa.

INTERNET STANDARDS

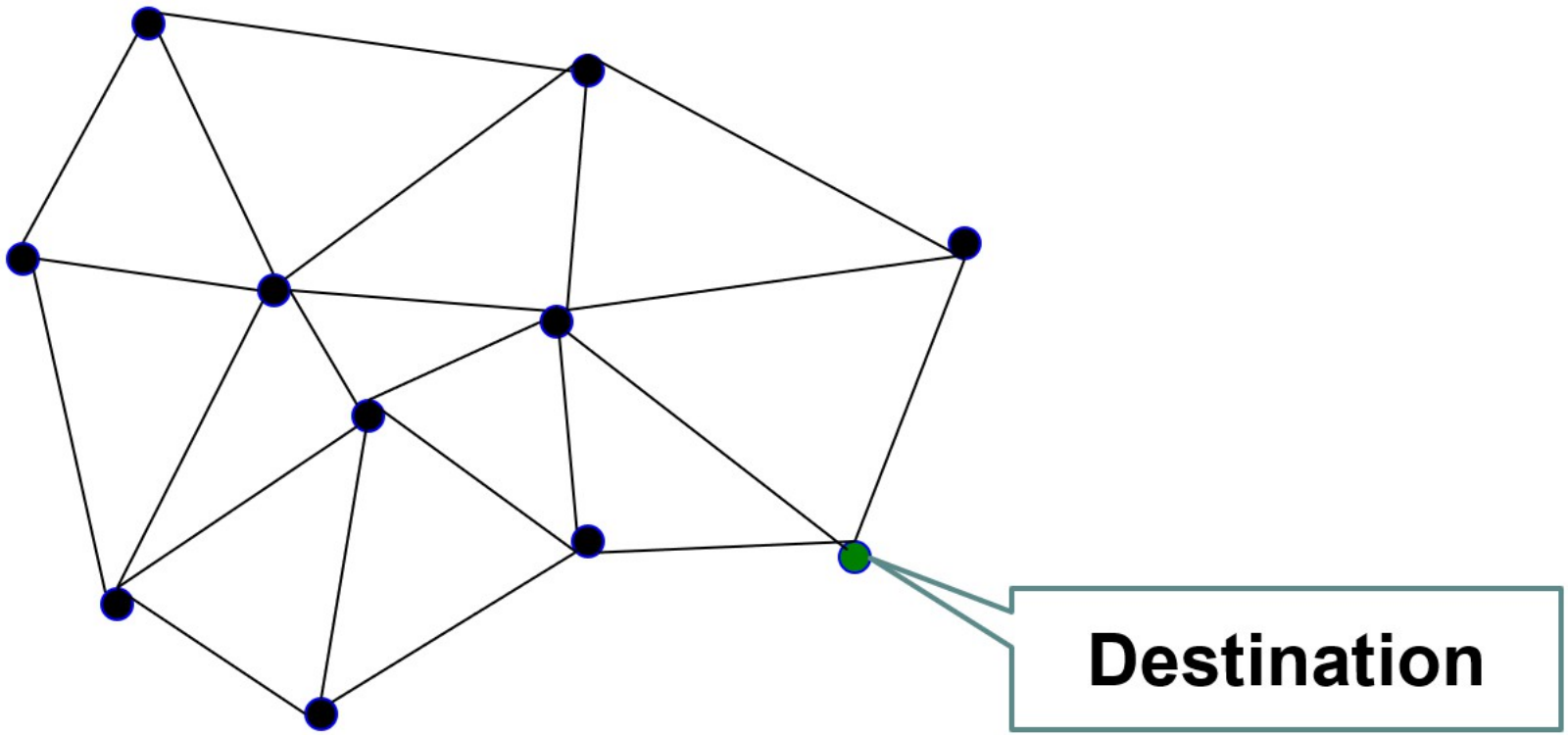
- i working group presentano i risultati del loro lavoro pubblicando, sulla rete stessa, un insieme di RFC, che sono la base degli standard di Internet
- non tutte le RFC diventano standards!
- necessario un processo di standardizzazione
  - Internet Drafts
  - RFC
  - Proposed Standard
  - Draft Standard (richiede 2 working implementations)
  - Internet Standard
- RFC per RIP:

Updated by: 4822	INTERNET STANDARD
	Errata Exist
Network Working Group	G. Malkin
Request for Comments: 2453	Bay Networks
Obsoletes: 1723, 1388	November 1998
	STD: 56

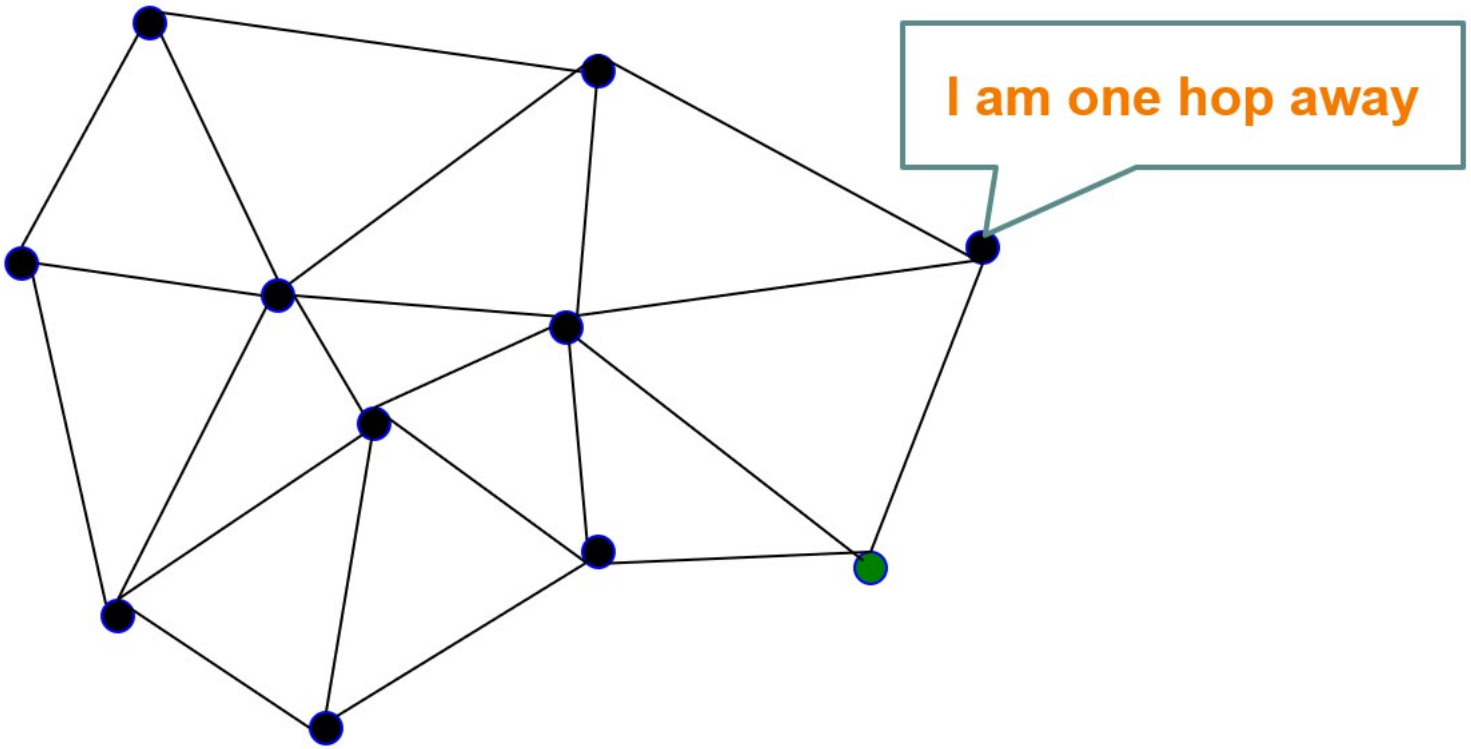
BELLMANN FORD DISTRIBUITO: L'IDEA

- Idea fondamentale dell'algoritmo sequenziale
  - cercare cammini verso ogni altro nodo
  - ogni cammino mi da una stima della distanza
  - se trovo un cammino migliore, aggiorno la stima
- Nell'algoritmo sequenziale, procedo "in avanti"
  - individuo i cammini partendo dalla radice ed aggiungendo sempre archi
- Implementazione distribuita:
  - tutti i nodi eseguono l'algoritmo, in parallelo
  - ogni nodo ha solo conoscenza dei vicini, non si ricostruisce lo stato globale della rete
  - Il cammino minimo verso un nodo X viene costruito "a ritroso", partendo dal nodo X ed andando verso la sorgente
  - Ma la logica fondamentale è la solita: costruire una stima del cammino migliore verso X e migliorarla successivamente

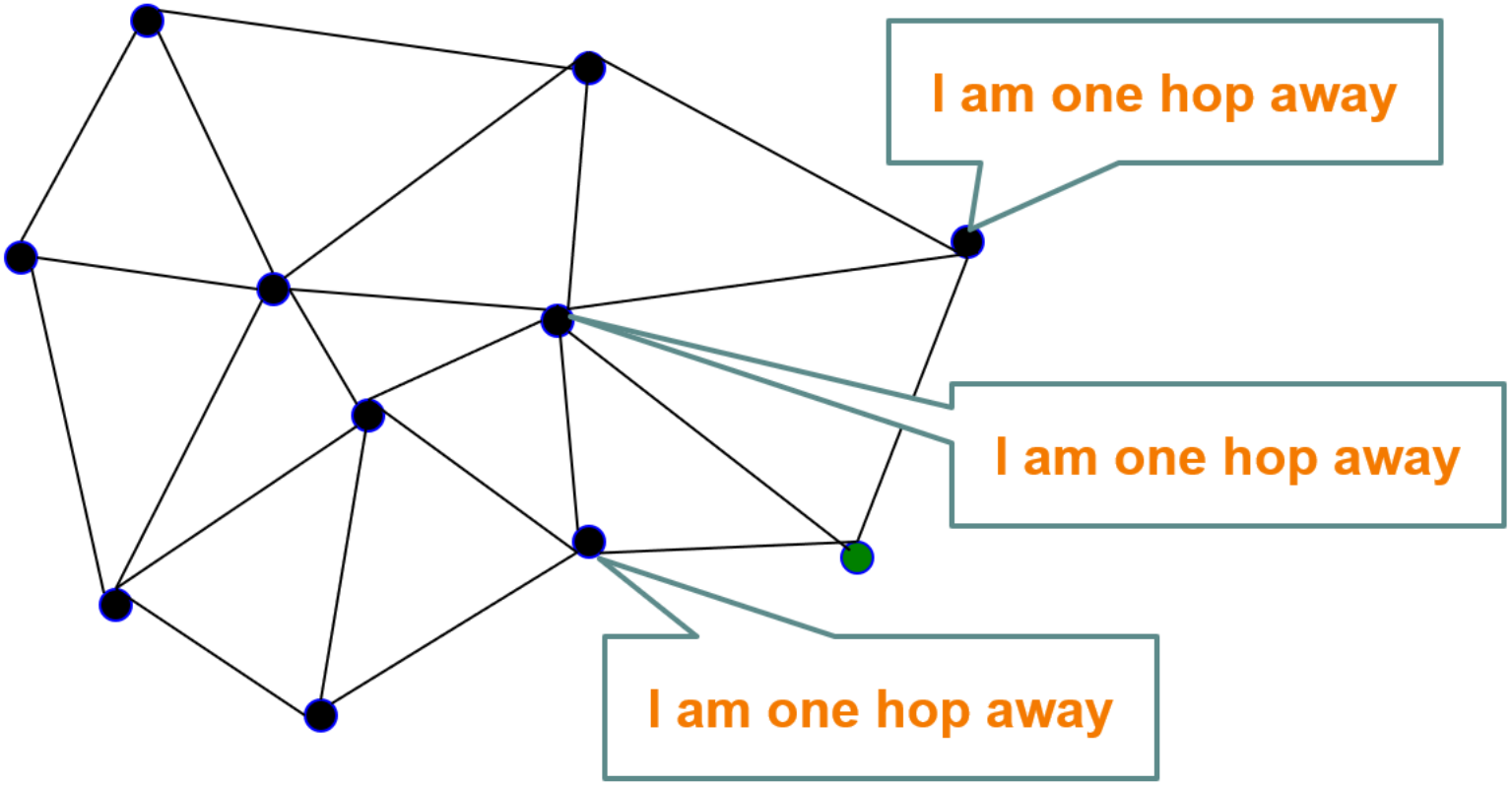
BELLMANN FORD DISTRIBUITO: L'IDEA



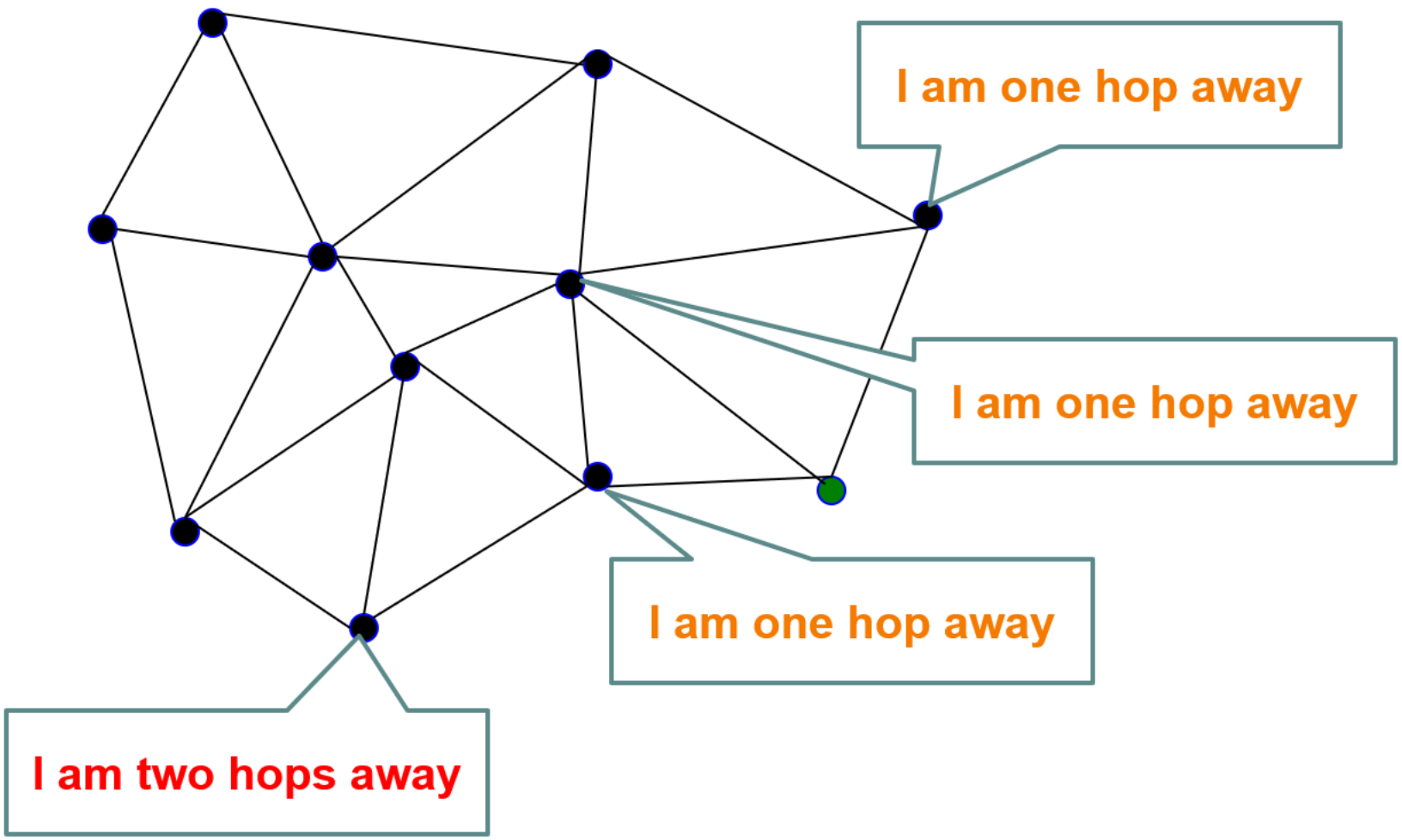
BELLMANN FORD DISTRIBUITO: L'IDEA



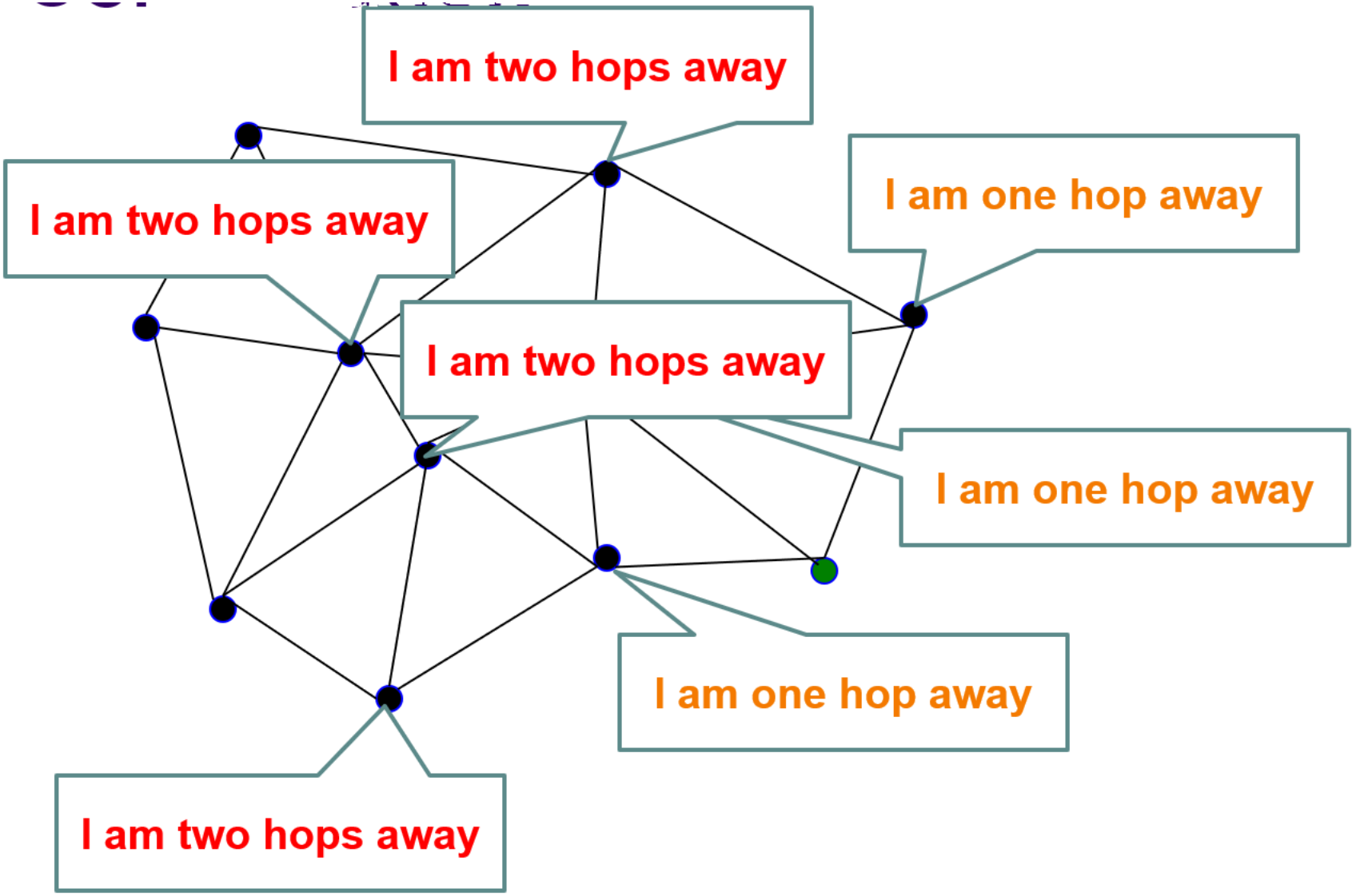
BELLMANN FORD DISTRIBUITO: L'IDEA



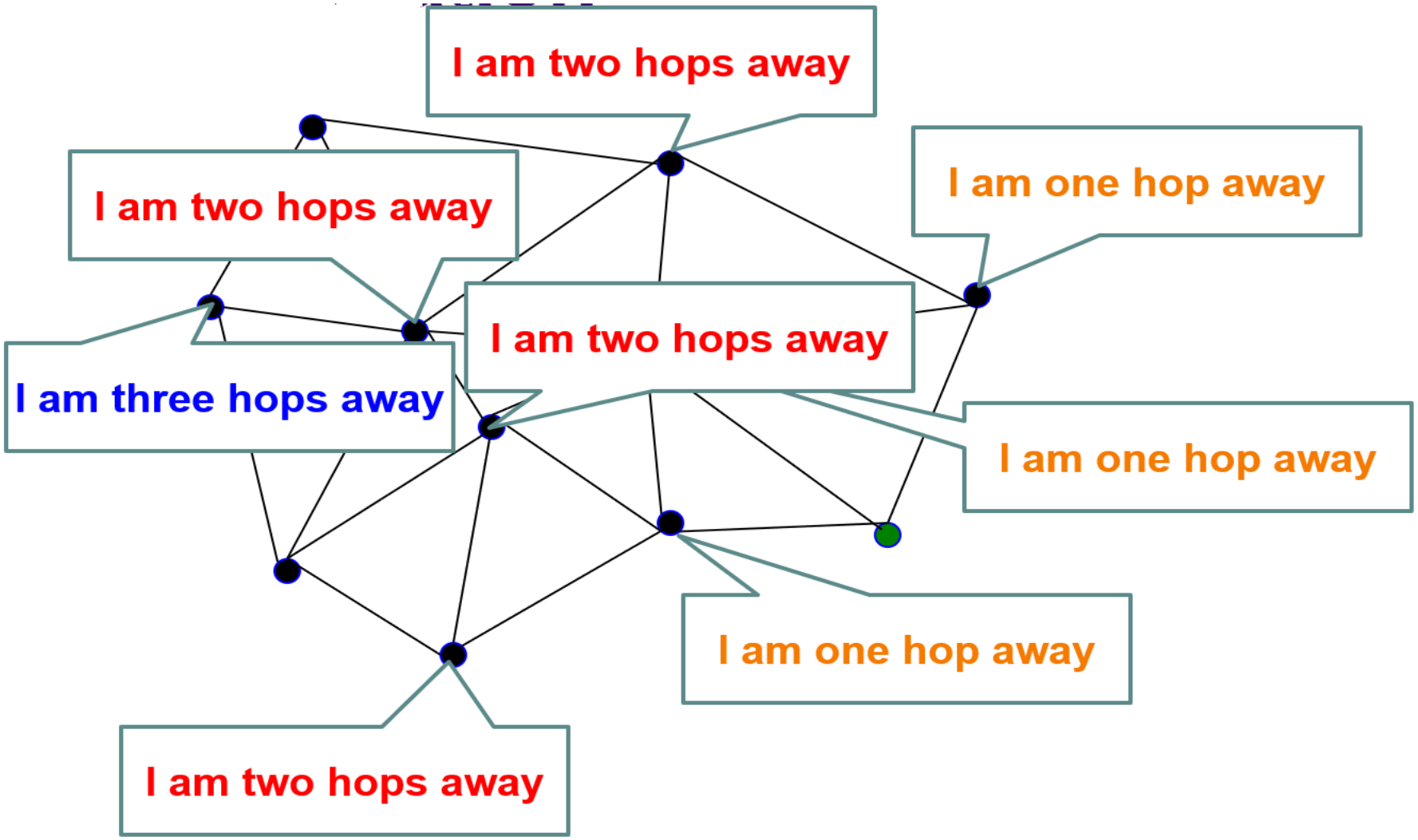
BELLMANN FORD DISTRIBUITO: L'IDEA



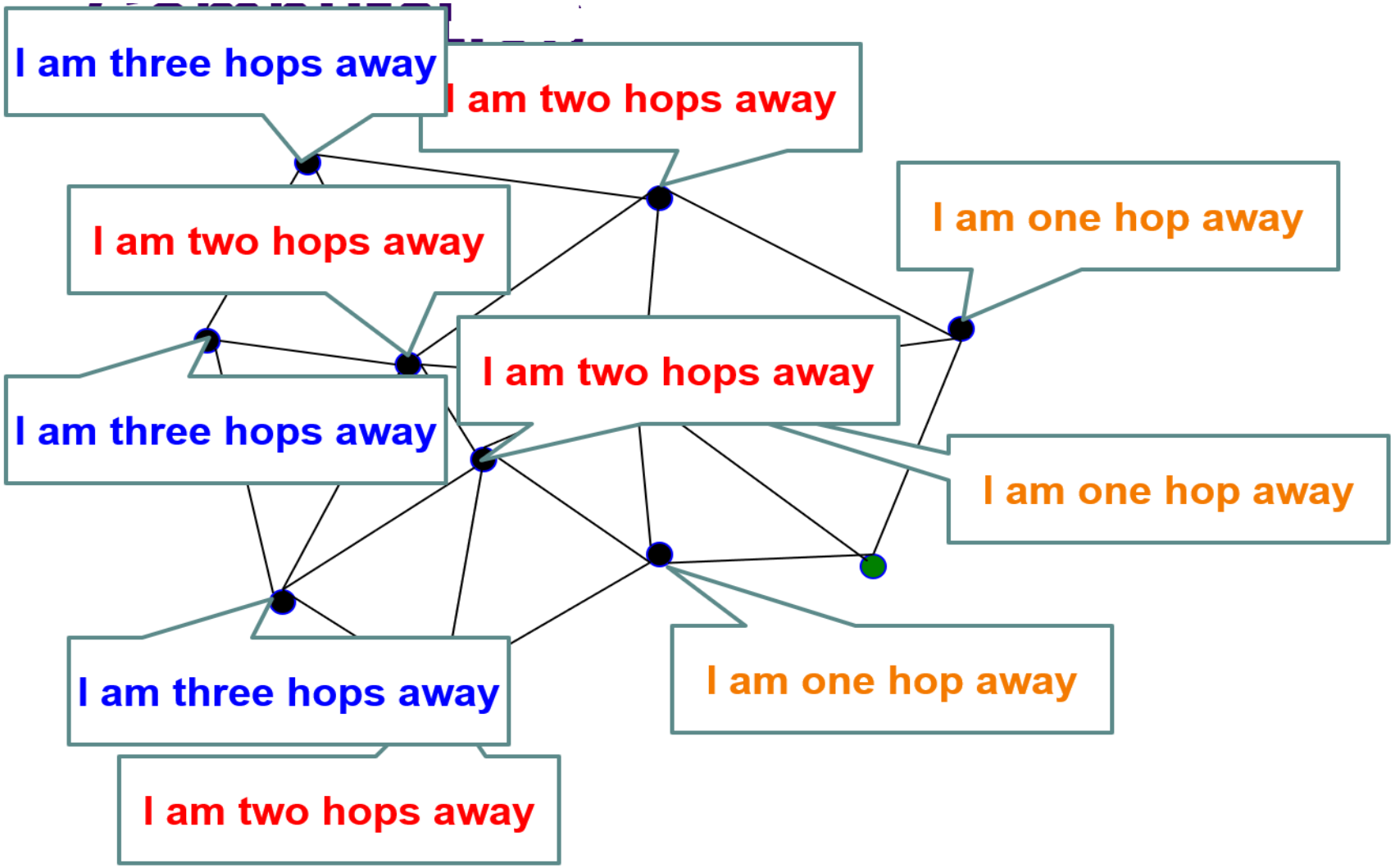
BELLMANN FORD DISTRIBUITO: L'IDEA



BELLMANN FORD DISTRIBUITO:L'IDEA



BELLMANN FORD DISTRIBUITO: L'IDEA



BELLMANN FORD DISTRIBUITO

- sia
  - D(i,j) = lunghezza del miglior cammino conosciuto (stima) tra il nodo i ed il nodo remoto j (non appartenente ai vicini di i)
  - d(i,j) = distanza tra il nodo i ed un nodo j, suo vicino
    - impostata a ∞, se i=j oppure i e j non sono vicini nel grafo
- equazione di Bellman-Ford:

$$D(i,j) = \min \{d(i,k) + D(k,j)\} \text{ for all } i \neq j, \quad k \in \text{neighbors}$$

- in Bellman-Ford distribuito tutti i nodi applicano la proprietà precedente, in modo induttivo.

BELLMAN FORD DISTRIBUITO

- l'iterazione dell'algoritmo di Bellman-Ford

$$D_i \leftarrow \min_{j \in N(i)} \{d_{ij} + D_j\}$$

dove *N*(i) è l'insieme dei nodi adiacenti ad i, viene eseguita **da ogni nodo i in parallelo con gli altri.**

Versione sincrona,

- le iterazioni sono eseguite simultaneamente su ogni nodo, in modo sincrono,
- ad ogni iterazione ogni nodo scambia i risultati con i vicini
- l'algoritmo converge in al più |V|-1 passi.

Versione asincrona, ogni nodo i, ad ogni iterazione esegue il calcolo:

- nodi diversi possono eseguire il calcolo in istanti diversi
- usa le stime Dj più recenti ricevute dai vicini e trasmette Di ai propri vicini.

ROUTING BASATO SU DISTANCE VECTORS

ogni router

- conosce il costo dei link verso i suoi vicini
- non propaga questa informazione sull'intera rete.
- ha quindi solo una visione locale della rete.
- mantiene la lunghezza provvisoria del cammino minimo verso un qualsiasi altro router
  - esempio: router A "posso arrivare al router B con un cammino di costo 23"
  - questa informazione è memorizzata in un vettore detto distance vector.

- distance Vector (DV):

  [indirizzo destinatario, distanza]

  una entrata per ogni destinazione.

BELLMANN FORD DISTRIBUITO

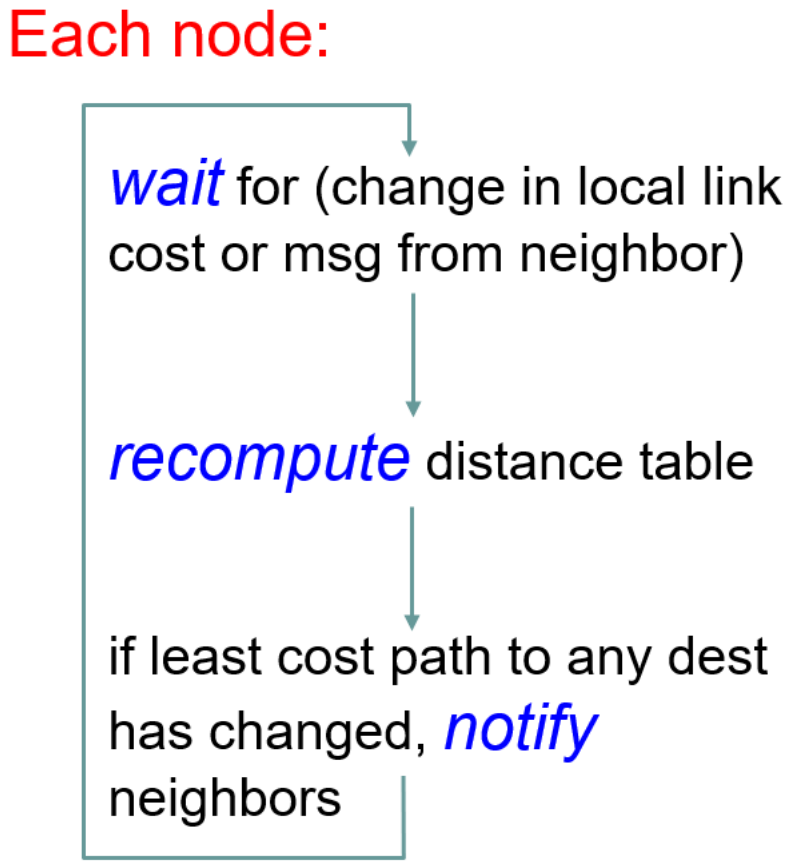
INPUT:

- Costi dei link verso i vicini
  (*non la topologia completa*)

OUTPUT:

- La tabella di routing: prossimo hop verso ogni altro router e costo corrispondente:
  (*non il cammino completo verso la destinazione*)

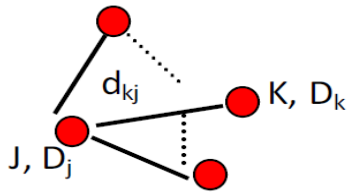
- ogni vicino X mi notifica quanto è lontani da una certa destinazione
  - calcolare (costo del link verso X) + (costo del vicino verso la destinazione)
  - scegli il minimo tra tutti i valori precedenti
  - notifica il costo del cammino a i vicini



BELLMANN FORD DISTRIBUITO

- ogni router
  - invia il suo DV ai router vicini
    - se un link a cui è connesso cambia di stato
    - se il risultato di un ricalcolo differisce dal precedente e viene a conoscenza
  - esegue il ricalcolo delle distanze se
    - riceve un DV diverso da quello precedente
    - cade/nasce una linea a cui è connesso
  - analizza l'insieme di opzioni offerte dai loro vicini, verso una certa destinazione e seleziona la migliore, la memorizza nel proprio distance vector

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

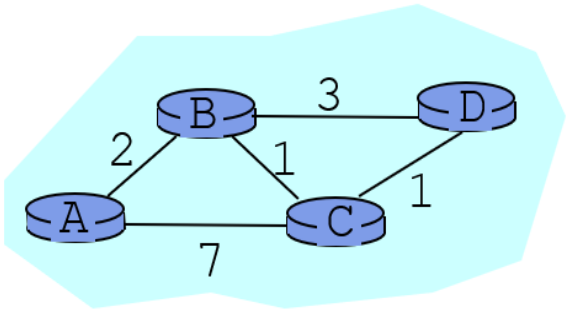


- Il processo prosegue in modo iterativo ed il vettore posseduto da ogni router converge verso i cammini minimi

BELLMANN FORD: STRUTTURE DATI

ogni router mantiene una tabells

- una colonna per ogni vicino (nodo collegato direttamente)



Node A

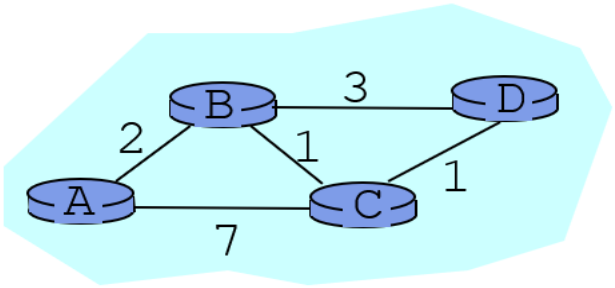
	B	C
B	2	8
C	3	7
D	4	8

Neighbor (next-hop)

BELLMANN FORD

ogni router mantiene una tabells

- una colonna per ogni vicino (nodo collegato direttamente)
- una riga per ogni possibile destinazione



Node A

	B	C
B	2	8
C	3	7
D	4	8

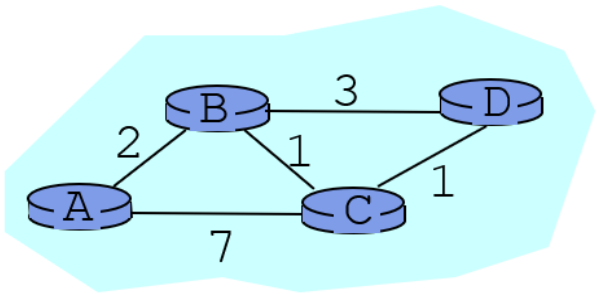
Neighbor (next-hop)

Destinations

BELLMANN FORD

ogni router mantiene una tabella

- una colonna per ogni vicino (nodo collegato direttamente)
- una riga per ogni possibile destinazione
- il contenuto dell'entrata nella riga Y e nella colonna Z di un nodo X è la migliore distanza conosciuta da X, via Z come prossimo hop $D_Z(X,Y)$



Node A

	B	C
B	2	8
C	3	7
D	4	8

Neighbor (next-hop)

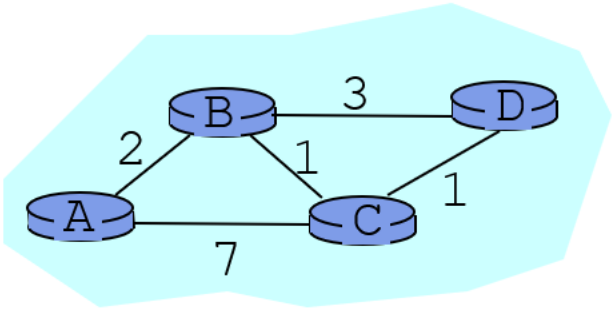
Destinations

$D_C(A, D)$

BELLMANN FORD

ogni router mantiene una tabells

- una riga per ogni possibile destinazione
- una colonna per ogni vicino (nodo collegato direttamente)
- il contenuto dell'entrata nella riga Y e nella colonna Z di un nodo X è la migliore distanza conosciuta da X, via Z come prossimo hop $D_Z(X,Y)$
- Il valore più piccolo in una riga è la stima della distanza minima verso una de



Node A

	B	C
B	2	8
C	3	7
D	4	8

Smallest distance in row Y = shortest Distance of A to Y, **D(A, Y)**

BELLMANN FORD DISTRIBUITO

```
1 Initialization:
2   for all neighbors V  do
3        if V adjacent to A
4             D(A, V) = c(A,V);
5        else
6             D(A, V) = ∞;
7     send D(A, Y) to all neighbors
```

c(i,j): costo del link dal nodo i ed il nodo j

Dz(A,V): costo del cammino da A a V via Z

D(A,V): costo del miglior cammino di A verso V

BELLMANN FORD DISTRIBUITO

```
1 Initialization:
2   for all neighbors V  do
3        if V adjacent to A
4             D(A, V) = c(A,V);
5        else
6             D(A, V) = ∞;
7     send D(A, Y) to all neighbors
 loop:
8   wait (until A sees a link cost change to neighbor V  /* case 1 */
9          or until A receives update from neighbor V)    /* case 2 */
```

c(i,j): costo del link dal nodo i ed il nodo j

Dz(A,V): costo del cammino da A a V via Z

D(A,V): costo del miglior cammino di A verso V

BELLMANN FORD DISTRIBUITO

```
1 Initialization:
2   for all neighbors V do
3        if V adjacent to A
4             D(A, V) = c(A,V);
5        else
6             D(A, V) = ∞;
7     send D(A, Y) to all neighbors
 loop:
8   wait (until A sees a link cost change to neighbor V  /* case 1 */
9          or until A receives update from neighbor V)    /* case 2 */
10  if (c(A,V) changes by ±d)  /* ⇐ case 1 */
11        for all destinations Y that go through V do
12             D_V(A,Y) =  D_V(A,Y) ± d
```

c(i,j): costo del link dal nodo i ed il nodo j

Dz(A,V): costo del cammino da A a V via Z

D(A,V): costo del miglior cammino di A verso V

BELLMANN FORD DISTRIBUITO

```
1 Initialization:
2   for all neighbors V  do
3        if V adjacent to A
4             D(A, V) = c(A,V);
5        else
6             D(A, V) = ∞;
7     send D(A, Y) to all neighbors
 loop:
8   wait (until A sees a link cost change to neighbor V  /* case 1 */
9          or until A receives update from neighbor V)    /* case 2 */
10   if (c(A,V) changes by ±d)  /* ⇐ case 1 */
11         for all destinations Y that go through V do
12              D_V(A,Y) =  D_V(A,Y) ± d
13   else if (update D(V, Y) received from V) /* ⇐ case 2 */
         /* shortest path from V to some Y has changed  */
14         D_V(A,Y) = D_V(A,V) + D(V, Y);    /* may also change D(A,Y) */
```

c(i,j): costo del link dal nodo i ed il nodo j

Dz(A,V): costo del cammino da A a V via Z

D(A,V): costo del miglior cammino di A verso V

BELLMANN FORD DISTRIBUITO

```
1 Initialization:
2   for all neighbors V  do
3        if V adjacent to A
4             D(A, V) = c(A,V);
5        else
6             D(A, V) = ∞;
7    send D(A, Y) to all neighbors
 loop:
8   wait (until A sees a link cost change to neighbor V  /* case 1 */
9          or until A receives update from neighbor V)    /* case 2 */
10  if (c(A,V) changes by ±d)  /* ⇐ case 1 */
11        for all destinations Y that go through V do
12             D_V(A,Y) =  D_V(A,Y) ± d
13  else if (update D(V, Y) received from V) /* ⇐ case 2 */
          /* shortest path from V to some Y has changed  */
14        D_V(A,Y) = D_V(A,V) + D(V, Y);    /* may also change D(A,Y) */
15  if (there is a new minimum for destination Y)
16        send D(A, Y) to all neighbors
```

c(i,j): costo del link dal nodo i ed il nodo j

Dz(A,V): costo del cammino da A a V via Z

D(A,V): costo del miglior cammino di A verso V

BELLMANN FORD DISTRIBUITO

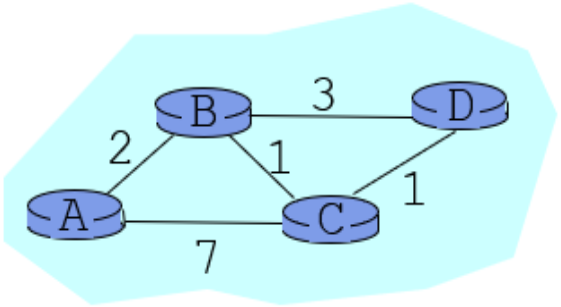
```
1 Initialization:
2   for all neighbors V  do
3        if V adjacent to A
4             D(A, V) = c(A,V);
5        else
6             D(A, V) = ∞;
7    send D(A, Y) to all neighbors
 loop:
8   wait (until A sees a link cost change to neighbor V  /* case 1 */
9          or until A receives update from neighbor V)    /* case 2 */
10  if (c(A,V) changes by ±d)  /* ⇐ case 1 */
11        for all destinations Y that go through V do
12             D_V(A,Y) =  D_V(A,Y) ± d
13  else if (update D(V, Y) received from V) /* ⇐ case 2 */
          /* shortest path from V to some Y has changed  */
14        D_V(A,Y) = D_V(A,V) + D(V, Y);    /* may also change D(A,Y) */
15  if (there is a new minimum for destination Y)
16        send D(A, Y) to all neighbors
17  forever
```

c(i,j): costo del link dal nodo i ed il nodo j

Dz(A,V): costo del cammino da A a V via Z

D(A,V): costo del miglior cammino di A verso V

ESEMPIO: INIZIALIZZAZIONE



Node A

	B	C
B	2	∞
C	∞	7
D	∞	∞

Node B

	A	C	D
A	2	∞	∞
C	∞	1	∞
D	∞	∞	3

```
1 Initialization:
2   for all neighbors V do
3       if V adjacent to A
4           D(A, V) = c(A,V);
5       else
6           D(A, V) = ∞;
7    send D(A, Y) to all neighbors
```

Node C

	A	B	D
A	7	∞	∞
B	∞	1	∞
D	∞	∞	1

Node D

	B	C
A	∞	∞
B	3	∞
C	∞	1

INTERPRETAZIONE COLORI IN FIGURA

- elementi in giallo nella matrice rappresentano i cammini minimi (provvisorio) da un nodo ad una certa destinazione passanti per il nodo associato alla colonna corrispondente all'elemento

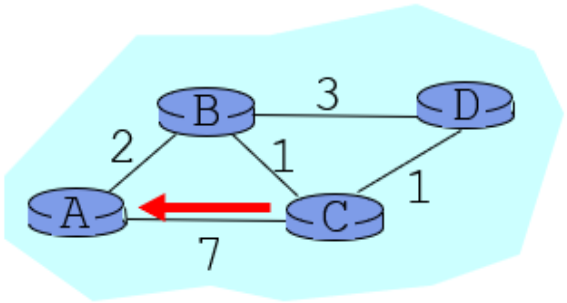
- Il distance vector contiene quindi i valori in giallo della tabella

- elementi in rosso sono quelli modificati a quella iterazione dell'algoritmo

ESEMPIO: C INVIA UN UPDATE AD A

$D_C(A, B) = D_C(A,C) + D(C, B) = 7 + 1 = 8$

$D_C(A, D) = D_C(A,C) + D(C, D) = 7 + 1 = 8$



Node A

	B	C
B	2	8
C	∞	7
D	∞	8

Node B

	A	C	D
A	2	∞	∞
C	∞	1	∞
D	∞	∞	3

D(C, B) = 1

D(C, D) = 1

Node C

	A	B	D
A	7	∞	∞
B	∞	1	∞
D	∞	∞	1

Node D

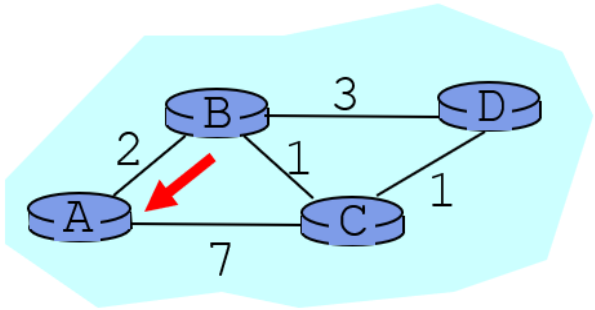
	B	C
A	∞	∞
B	3	∞
C	∞	1

```
7    loop:
     ...
13   else if (update D(A, Y) from C)
14     DC(A,Y) = DC(A,C) + D(C, Y);
15   if (new min. for destination Y)
16     send D(A, Y) to all neighbors
17   forever
```

ESEMPIO: B INVIA UN UPDATE AD A

$D_B(A, C) = D_B(A,B) + D(B, C) = 2 + 1 = 3$

$D_B(A, D) = D_B(A,B) + D(B, D) = 2 + 3 = 5$



Node A

	B	C
B	2	8
C	3	7
D	5	8

Node B

	A	C	D
A	2	∞	∞
C	∞	1	∞
D	∞	∞	3

D(B, C) = 1

D(B, D) = 3

```
7    loop:
     ...
13   else if (update D(A, Y) from B)
14     D_B(A,Y) = D_B(A,B) + D(B, Y);
15   if (new min. for destination Y)
16     send D(A, Y) to all neighbors
17   forever
```

Node C

	A	B	D
A	7	∞	∞
B	∞	1	
D	∞	∞	1

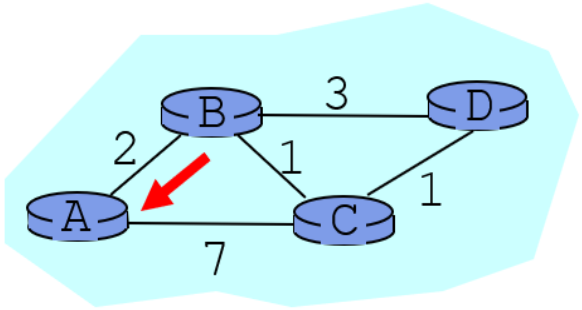
Node D

	B	C
A	∞	∞
B	3	∞
C	∞	1

ESEMPIO: B INVIA UN UPDATE AD A

$D_B(A, C) = D_B(A,B) + D(B, C) = 2 + 1 = 3$

$D_B(A, D) = D_B(A,B) + D(B, D) = 2 + 3 = 5$



Node A

	B	C
B	2	8
C	3	7
D	5	8

Node B

	A	C	D
A	2	∞	∞
C	∞	1	∞
D	∞	∞	3

D(B, C) = 1

D(B, D) = 3

Perchè il valore di questa entrata è **5** e non **4**?

```
7   loop:
    ...
13  else if (update D(A, Y) from B)
14    D_B(A,Y) = D_B(A,B) + D(B, Y);
15  if (new min. for destination Y)
16    send D(A, Y) to all neighbors
17  forever
```

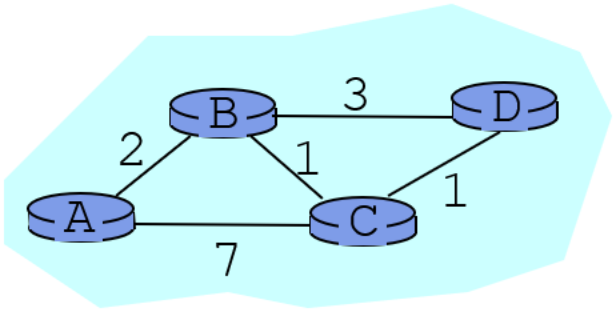
Node C

	A	B	D
A	7	∞	∞
B	∞	1	
D	∞	∞	1

Node D

	B	C
A	∞	∞
B	3	∞
C	∞	1

DOPO IL PRIMO SCAMBIO COMPLETO DI INFORMAZIONI



Assumiamo che tutti i nodi spediscano un update nello stesso momento

Alla fine della prima iterazione ogni nodo conosce il cammino migliore verso i vicini a due hop di distanza

Node A

	B	C
B	2	8
C	3	7
D	5	8

Node B

	A	C	D
A	2	8	∞
C	9	1	4
D	∞	2	3

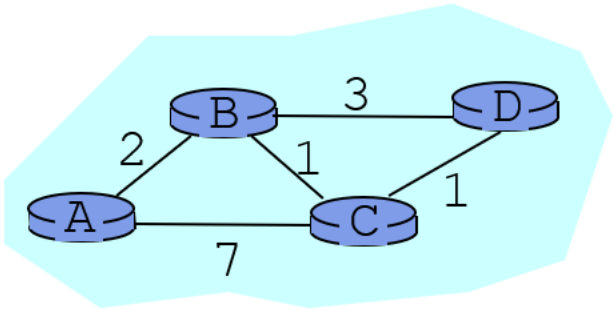
Node C

	A	B	D
A	7	3	∞
B	9	1	4
D	∞	4	1

Node D

	B	C
A	5	8
B	3	2
C	4	1

A INVIA UN UPDATE A B



Node A

	B	C
B	2	8
C	3	7
D	5	8

Node B

	A	C	D
A	2	8	∞
C	5	1	4
D	7	2	3

Riflettere su questi valori!

$D_A(B, C) = D_A(B,A) + D(A, C) = 2 + 3 = 5$

$D_A(B, D) = D_A(B,A) + D(A, D) = 2 + 5 = 7$

```
7    loop:
     …
13   else if (update D(B, Y) from A)
14     D_A(B,Y) = D_A(B,A) + D(A, Y);
15   if (new min. for destination Y)
16     send D(B, Y) to all neighbors
17   forever
```

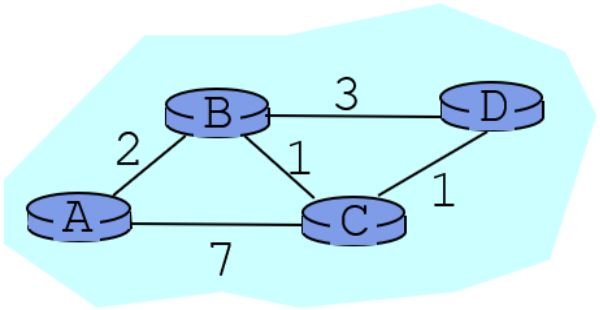
Node C

	A	B	D
A	7	3	∞
B	9	1	4
D	∞	4	1

Node D

	B	C
A	5	8
B	3	2
C	4	1

ALLA FINE DELLA SECONDA ITERAZIONE



Node A

	B	C
B	2	8
C	3	7
D	4	8

Node B

	A	C	D
A	2	4	8
C	5	1	4
D	7	2	3

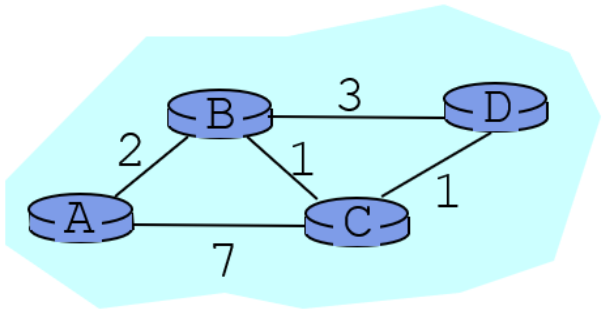
Node C

	A	B	D
A	7	3	6
B	9	1	3
D	12	3	1

Node D

	B	C
A	5	4
B	3	2
C	4	1

ALLA FINE DELLA TERZA ITERAZIONE



Node A

	B	C
B	2	8
C	3	7
D	4	8

Node B

	A	C	D
A	2	4	7
C	5	1	4
D	6	2	3

Node C

	A	B	D
A	7	3	5
B	9	1	3
D	11	3	1

Node D

	B	C
A	5	4
B	3	2
C	4	1

Riflettere su questo valore

DISTANCE VECTOR: RETI DINAMICHE

- Intuizione:
  - stato iniziale: i migliori cammini ad un hop
  - dopo un round eseguito simultaneamente da tutti: migliori cammini a due-hop
  - dopo k round eseguiti simultaneamente: i migliori cammini a (k+1) hop

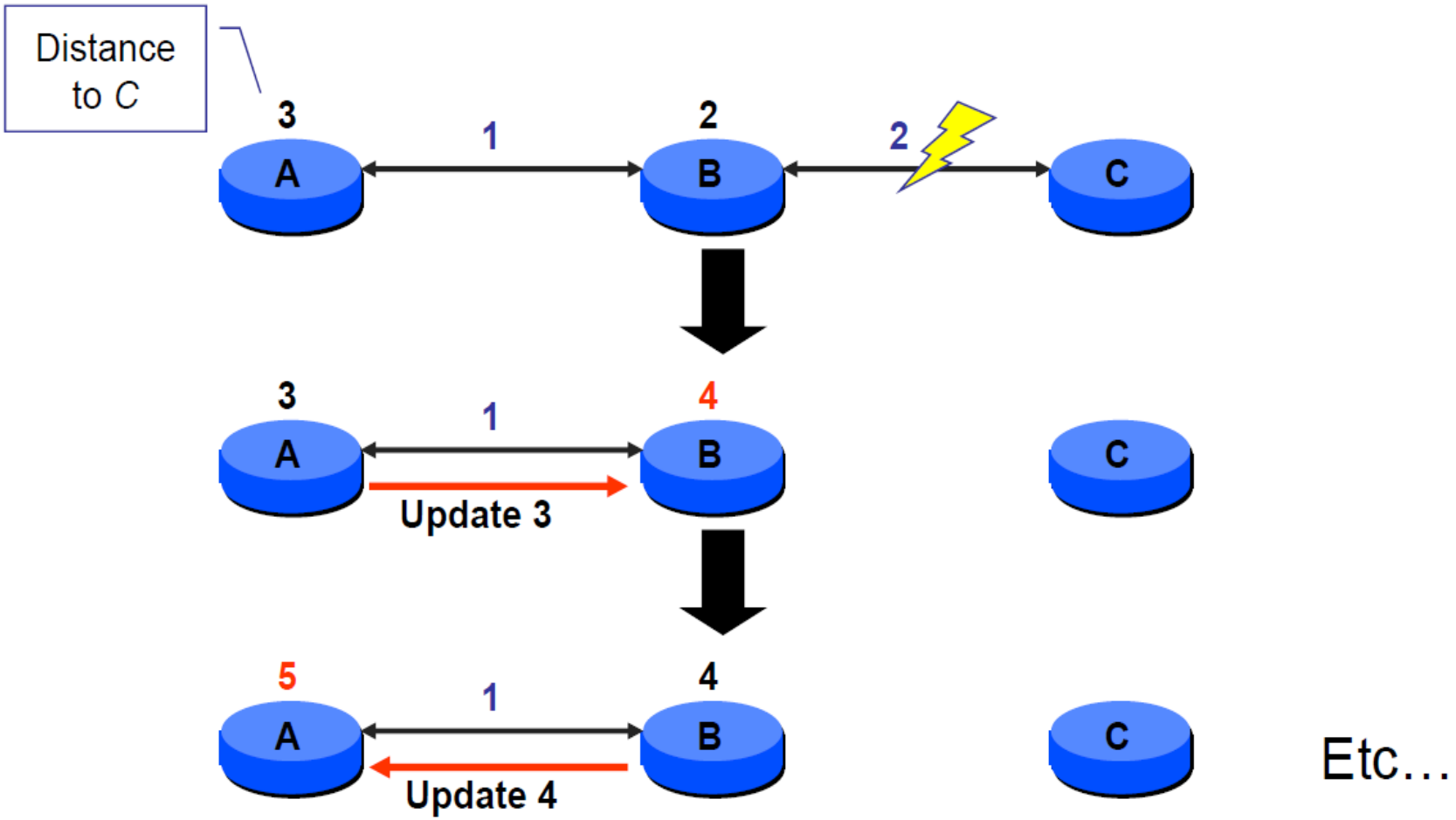
    …...
  - convergenza avviene quando si raggiunge il cammino migliore più lungo

- ma quale è il comportamento del protocollo su una rete dinamica in presenza di fallimenti e di churn?
  - proprietà di convergenza del protocollo sono meno ovvie e più difficili da provare
  - velocità di convergenza
    - diverso se considerata al momento del bootstrap della rete oppure in seguito alla modifica di un arco
    - può dipendere da valori su archi, in caso di modifica dinamica dei costi

DISTANCE VECTOR: RETI DINAMICHE

- Reti dinamiche, con possibilità di fallimenti e di cambiamento di stato
- La parte più interessante è studiare come si comporta il protocollo in caso di rete dinamica
  - a differenza del link-state protocol, il distance vector distribuisce il calcolo dei cammini minimi
  - di conseguenza, le proprietà di convergenza del protocollo sono meno ovvie e più difficili da provare
  - per una rete dinamica, possibili
    - routing loops
    - convergenza molto lenta verso una visione consistente della rete
    - richiede la definizione di ulteriori meccanismi
      - split horizon
      - poisonous reverse

COUNTING TO INFINITY: LOOP INFINITO

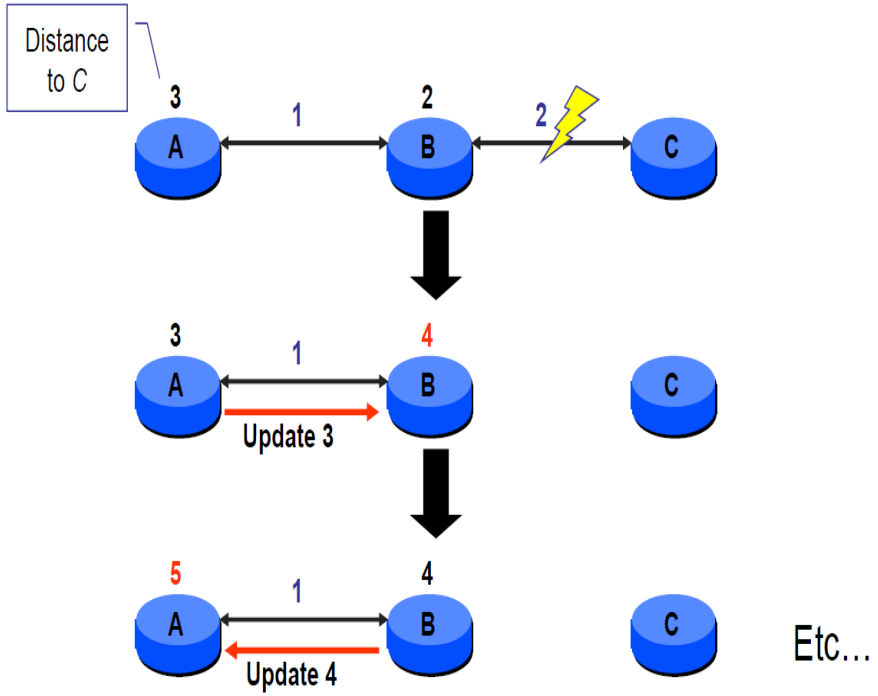


COUNTING TO INFINITY: LOOP INFINITO

- inizialmente, sia B che A hanno una rotta verso il router C, con distanze, rispettivamente, 2 e 3

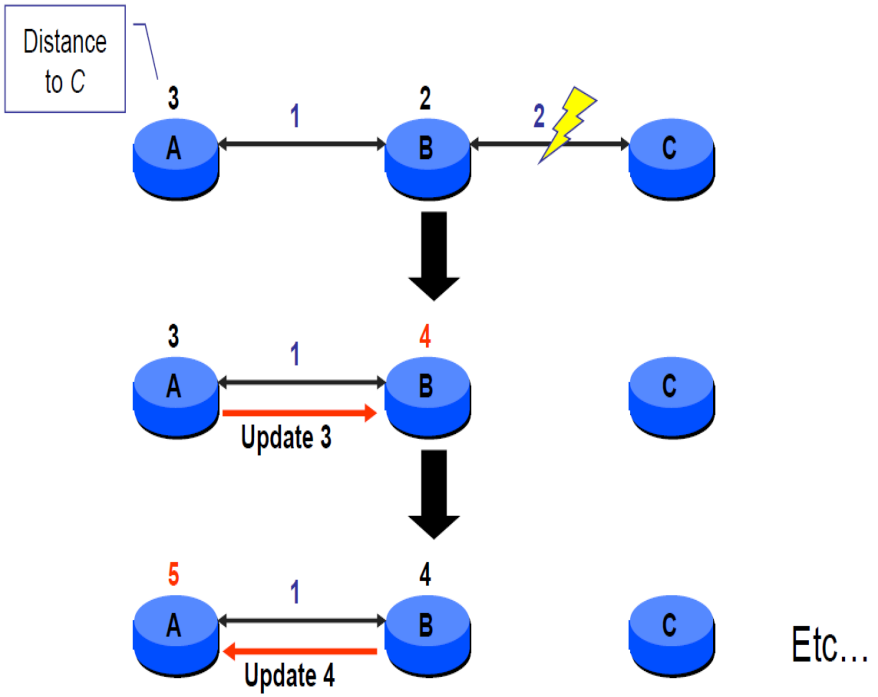
- Il link da B ad C subisce un fallimento

- B rimuove la sua rotta verso C associando a C, nella tabella di routing, il valore infinito.



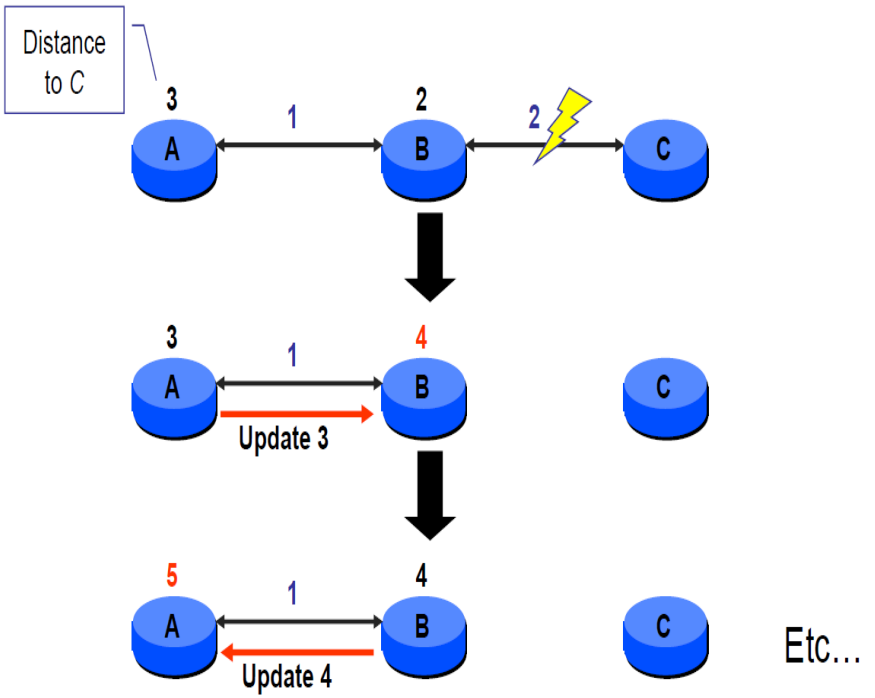
COUNTING TO INFINITY: LOOP INFINITO

- cosa può accadere dopo che B ha rimosso la sua rotta?
- dipende dall'ordinamento degli aggiornamenti scambiati tra A e B
- Scenario corretto:
  - B riporta ad A, per primo, la sua rotta (infinita) verso C
  - anche A imposta la sua rotta verso C ad infinito
  - la rete si è auto-riconfigurata correttamente



COUNTING TO INFINITY: LOOP INFINITO

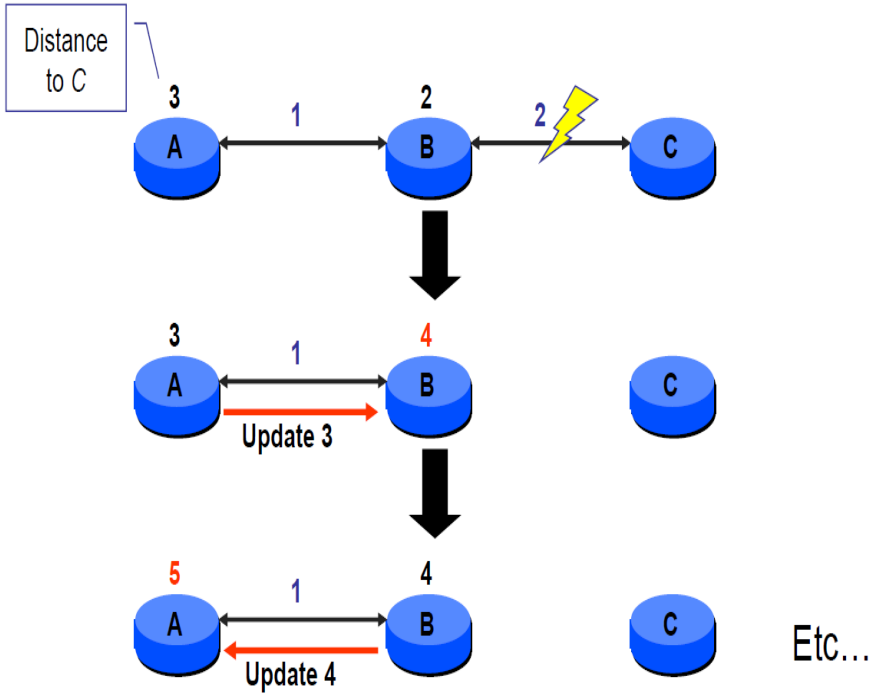
- Scenario errato:
  - A, prima che gli arrivi l'aggiornamento da B, invia la sua distanza (che è ancora valida) verso C a B
  - B riceve il messaggio da A, e non sa che è relativo proprio alla rotta che se stesso. Potrebbe riguardare un'altra rotta
- Loop! I due nodi iniziano a rimbalzarsi i pacchetti, con costo crescente
  - il loop potrebbe continuare all'infinito



COUNTING TO INFINITY: LOOP INFINITO

Counting to infinity, il problema:

- A ha una rotta sbagliata verso C, perchè pensa che B abbia modo per raggiungere C con costo = 3
- B in realtà non sa che il cammino di A è basato su quello che esso stesso ha precedentemente comunicato a A



COUNTING TO INFINITY: IL PROBLEMA

- Gli update non contengono sufficiente informazione
  - ogni nodo ha solo una "next-hop-view
  - B non in è grado di ordinare gli updates, ad esempio
    - prima le "cattive notizie" (un link è fallito)
    - poi le "buone notizie" (un link è disponibile)
- B accetta il cammino di A verso C che implicitamente attraversa se stesso
  - non individuabile con una visione "next hop"
- Le buone notizie prendono il sopravvento su quelle cattive
  - la ragione perchè le buone notizie si diffondono più rapidamente

COUNTING TO INFINITY: POSSIBILI SOLUZIONI

- Solutions to count-to-infinity in RIP
  - Hop count limit
  - Split Horizon
  - Split Horizon with Poison Reverse
  - Triggered Updates
  - Hold-down

- Altra soluzione: usare protocolli alternativi
  - notificare sempre l'intero cammino in ogni aggiornamento
  - Path Vectors: molto costoso se i cammini sono lunghi
    - utilizzato in BGP, interdomain routing protocol

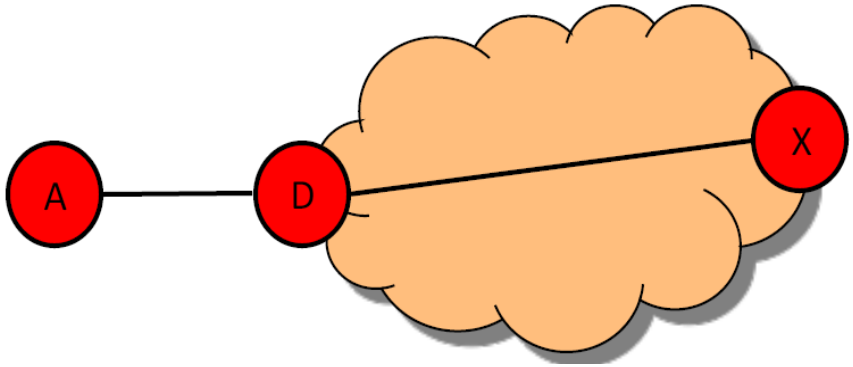
HOP COUNT LIMIT

- 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
  - valore = 16 in RIP
- 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
  - e la convergenza verso uno stato stabile è lenta

SPLIT HORIZON: IDEA GENERALE

Split horizon in forma semplice:

- se un nodo A viene a conoscenza del miglior cammino verso un nodo X dal vicino D, allora A non notifica questo cammino in risposta a D, perchè D fa parte di questo cammino



- Il nodo A non annuncia a Dil costo del suo cammino verso X

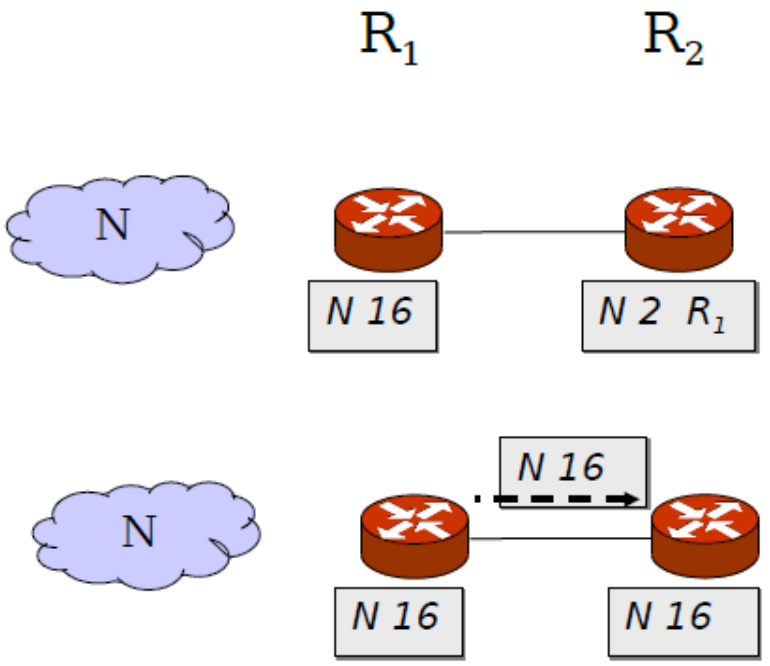
Split horizon con poison reverse:

- per assicurare ulteriormente che D non utilizzerà la informazione inviata da A, A invia a D la notifica di una distanza da X con un costo infinito
- Poison reverse: perchè A "avvelena" il suo cammino verso X quando lo notifica a D

SPLIT HORIZON

- Se ricevo un update da un vicino e decido che un cammino verso un nodo passa per quel vicino, non invio allo stesso vicino rotte per quel nodo.
- Distance vector non inviato su tutte le interfacce indiscriminatamente
- Serve per evitare un "mutuo inganno": due router che continuano a notificarsi a vicenda che possono raggiungere la destinazione uno attraverso l'altro

  - R2, non notifica ad R1 la sua distanza verso N, poiché dovrebbe rimandarla sulla stessa interfaccia da cui ha ricevuto la notifica
  - prima o poi, R1 notifica la sua distanza verso N ad R2 (16 è infinito in RIP), lo stato della rete risulta corretto.
    - previene solo i loop tra due router

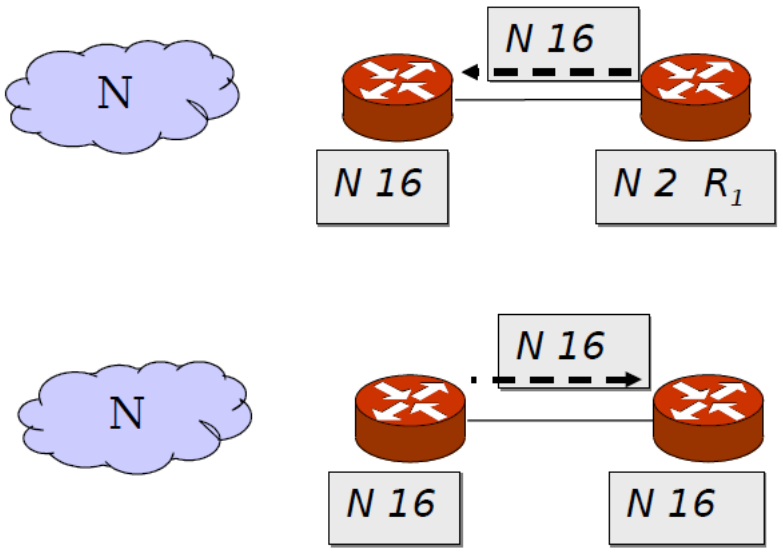


SPLIT HORIZON + POISOUNOS REVERSE

- il router inviea al router router da cui a ricevuto un cammino verso un nodo X, un distance vector che riporta il valore infinito verso X, comunicando di fatto a quel router che per lui la destinazione è irraggiungibile.
  - non aggiunge informazione, ma consente di risolvere i loop più rapidamente
  - aggiunge overhead al protocollo
  - poison reverse deve essere supportato da tutte le implementazioni di RIP, ma è ammesso che venga disabilitato

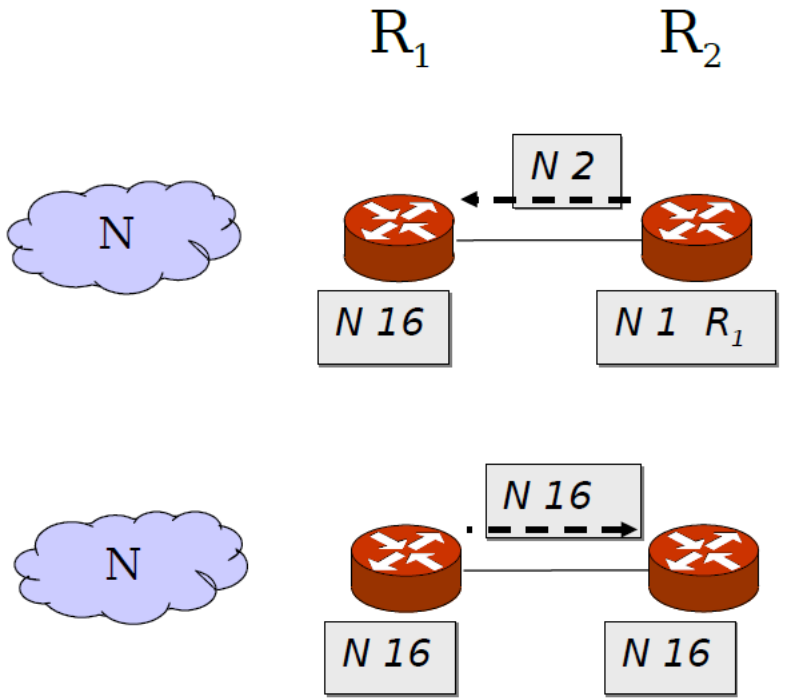
R_1 R_2

- R2 annuncia ad R1 un cammino infinito verso N (N irraggiungibile), perchè il suo cammino passa proprio per R1
  - Prima o dopo, R1 invia la sua rotta verso N ad R2 ed R2 registra la irraggiungibilità di N



HOLD-DOWN

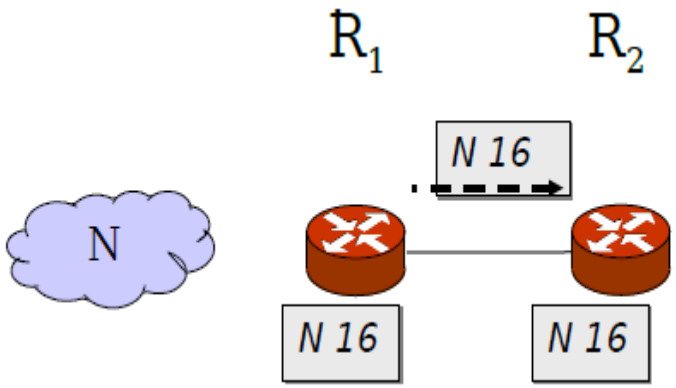
- Quando un cammino viene rimosso, nessun aggiornamento per questo cammino viene accettato per un certo periodo di tempo (hold-down time) per dare a chiunque l'opportunità di rimuovere il cammino
- Non in RIP RFC, ma implementato da CISCO



- R1 ignora gli aggiornamenti che arrivano da R2 e riguardano N, per un certo periodo di tempo
  - prima o poi, R1 manderà un aggiornamento ad R2

TRIGGERED UPDATE

- quando la metrica associata ad un link cambia, invia immediatamente un update sui link in uscita.
- l'aggiornamento riguarda solo il valore relativo al link, non tutto il vettore distanza
- questo può provocare una sequenza di updates
  - la regola viene applicata ricorsivamente
  - RIP filtra questi update in modo da non permetterne più di uno ogni 1-5 secondi
- usato quando vengono cancellate delle rotte
- CISCO implementa queste "flash updates"
  - R1 annuncia immediatamente la rottura del link, quando questa accade
- Problema: non funziona in caso di perdita di pacchetti



CONVERGENZA LENTA

Distanze verso

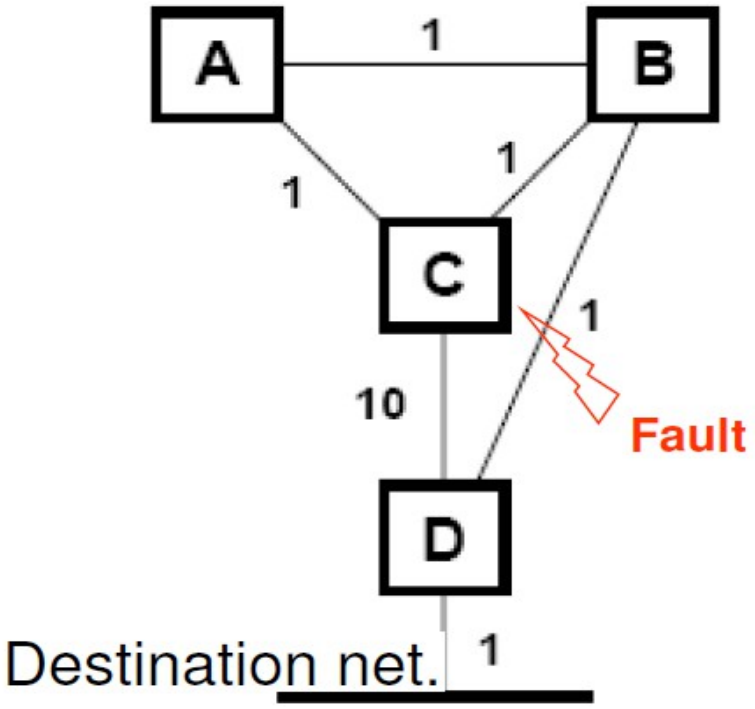
Destination net, Next hop

Node A → Distanza = 3, Router B

Node B → Distanza= 2, Router D

Node C → Distanza = 3, Router B

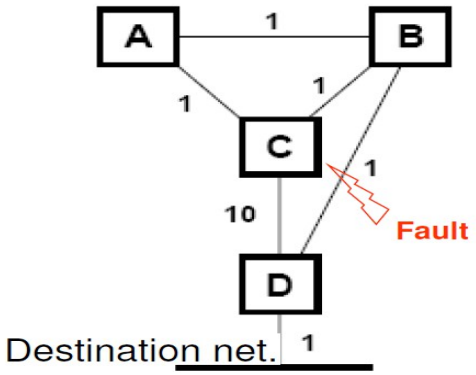
Node D → Distanza = Diretto



- i cammini minimi da A, B, C verso la Destination net, passano tutti da B
- fault del link BD
  - dopo il fault tutti i cammini minimi devono passare tramite D

CONVERGENZA LENTA

- dopo il fault, i cammini da A, C e B, verso la rete destinazione, devono riconfigurarsi, in modo da passare da C e non più da B
- seguiamo l'evoluzione del cammino dai routers A,B,C verso la Destination network



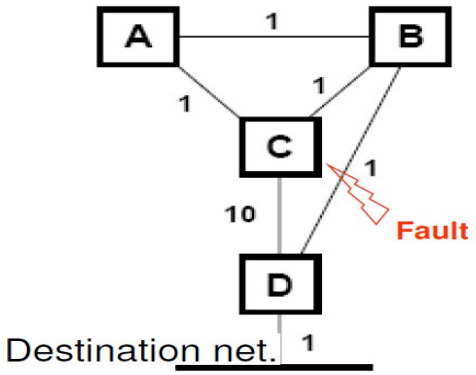
Step 1

Nodo	Next hop	Metr.
A	B	3
B	----	∞
C	B	3
D	Diretto	1

- B updates A and C with new vector
- Meanwhile, B receives from A and C the information (incorrect) that from A and C it is possible to reach destination network with metric cost 3

CONVERGENZA LENTA

- dopo il fault, i cammini da A, C e B, verso la rete destinazione, devono riconfigurarsi, in modo da passare da C e non più da B
- seguiamo l'evoluzione del cammino dai routers A,B,C verso la Destination network



Step 1

Nodo	Next hop	Metr.
A	B	3
B	----	∞
C	B	3
D	Diretto	1

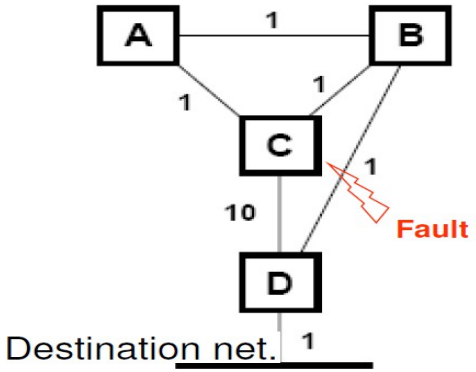
Step 2

Nodo	Next hop	Metr.
A	C	4
B	C	4
C	A	4
D	Diretto	1

- A and C receive B's vector and change their paths toward destination

CONVERGENZA LENTA

- dopo il fault, i cammini da A, C e B, verso la rete destinazione, devono riconfigurarsi, in modo da passare da C e non più da B
- seguiamo l'evoluzione del cammino dai routers A,B,C verso la Destination network



Step 1

Nodo	Next hop	Metr.
A	B	3
B	----	∞
C	B	3
D	Diretto	1

Step 2

Nodo	Next hop	Metr.
A	C	4
B	C	4
C	A	4
D	Diretto	1

Step 3

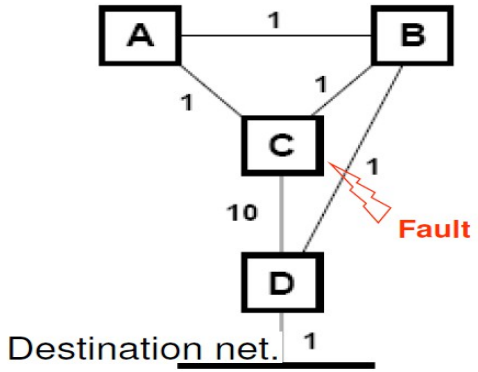
Nodo	Next hop	Metr.
A	C	5
B	C	5
C	A	5
D	Diretto	1

At this point:

- B receives vectors from A and C
- A receives vector from B and C
- C receives vector from A and B

CONVERGENZA LENTA

- dopo il fault, i cammini da A, C e B, verso la rete destinazione, devono riconfigurarsi, in modo da passare da C e non più da B
- seguiamo l'evoluzione del cammino dai routers A,B,C verso la Destination network



Step 1

Nodo	Next hop	Metr.
A	B	3
B	----	∞
C	B	3
D	Diretto	1

Step 2

Nodo	Next hop	Metr.
A	C	4
B	C	4
C	A	4
D	Diretto	1

Step 3

Nodo	Next hop	Metr.
A	C	5
B	C	5
C	A	5
D	Diretto	1

Step 4

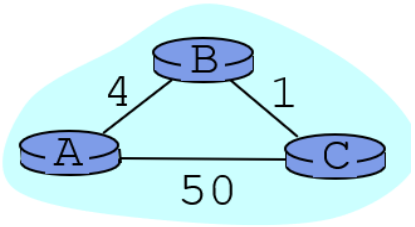
Nodo	Next hop	Metr.
A	C	6
B	C	6
C	A	6
D	Diretto	1

Loop terminates when metric of temporary path becomes larger than that of real, alternative path

Step 10

Nodo	Next hop	Metr.
A	C	12
B	C	12
C	D	11
D	Diretto	1

DISTANCE VECTOR: VELOCITA' DI CONVERGENZA



	Stable state	A-B changed	A sends tables to B, C	B sends tables to C	C sends tables to B

Node A

	B	C
B	4	51
C	5	50

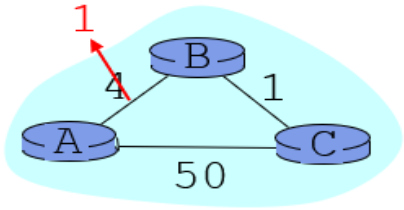
Node B

	A	C
A	4	6
C	9	1

Node C

	A	B
A	50	5
B	54	1

DISTANCE VECTOR: VELOCITA' DI CONVERGENZA

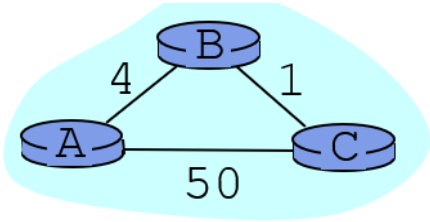


	Stable state			A-B changed			A sends tables to B, C			B sends tables to C			C sends tables to B		
Node A		B	C		B	C		B	C		B	C		B	C
	B	4	51	B	1	51	B	1	51	B	1	51	B	1	51
	C	5	50	C	2	50	C	2	50	C	2	50	C	2	50
Node B		A	C		A	C		A	C		A	C		A	C
	A	4	6	A	1	6	A	1	6	A	1	6	A	1	3
	C	9	1	C	6	1	C	3	1	C	3	1	C	3	1
Node C		A	B		A	B		A	B		A	B		A	B
	A	50	5	A	50	5	A	50	5	A	50	2	A	50	2
	B	54	1	B	54	1	B	51	1	B	51	1	B	51	1

Link cost changes here

"good news travels fast"

DISTANCE VECTOR: VELOCITA' DI CONVERGENZA



	Stable state	A-B changed	A sends tables to B, C	B sends tables to C	C sends tables to B

Node A

	B	C
B	4	51
C	5	50

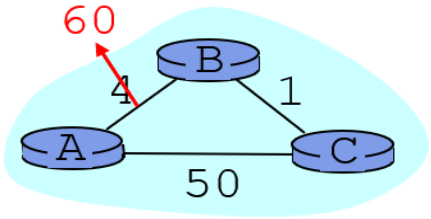
Node B

	A	C
A	4	6
C	9	1

Node C

	A	B
A	50	5
B	54	1

DISTANCE VECTOR: VELOCITA' DI CONVERGENZA

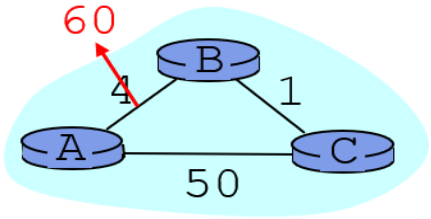


	Stable state		A-B changed		A sends tables to B, C		B sends tables to C		C sends tables to B	
Node A		B	C	B	C	B	C	B	C	B	C
	B	4	51	B 60	51	B 60	51	B 60	51	B 60	51
	C	5	50	C 61	50	C 61	50	C 61	50	C 61	50
Node B		A	C	A	C	A	C	A	C	A	C
	A	4	6	A 60	6	A 60	6	A 60	6	A 60	8
	C	9	1	C 65	1	C 110	1	C 110	1	C 110	1
Node C		A	B	A	B	A	B	A	B	A	B
	A	50	5	A 50	5	A 50	5	A 50	7	A 50	7
	B	54	1	B 54	1	B 101	1	B 101	1	B 101	1

Link cost changes here

"bad news travels slowly"
(not yet converged)

DISTANCE VECTOR: VELOCITA' DI CONVERGENZA

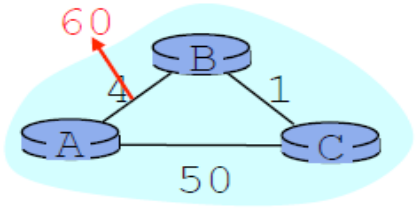


	Stable state		A-B changed		A sends tables to B, C		B sends tables to C		C sends tables to B	
Node A		B	C	B	C	B	C	B	C	B	C
	B	4	51	B 60	51	B 60	51	B 60	51	B 60	51
	C	5	50	C 61	50	C 61	50	C 61	50	C 61	50
Node B		A	C	A	C	A	C	A	C	A	C
	A	4	6	A 60	6	A 60	6	A 60	6	A 60	8
	C	9	1	C 65	1	C 110	1	C 110	1	C 110	1
Node C		A	B	A	B	A	B	A	B	A	B
	A	50	5	A 50	5	A 50	5	A 50	7	A 50	7
	B	54	1	B 54	1	B 101	1	B 101	1	B 101	1

Link cost changes here

"bad news travels slowly"
(not yet converged)

DISTANCE VECTOR: POISONED REVERSe



- If B routes through C to get to A:
  - B tells C its (B's) distance to A is infinite (so C won't route to A via B)

	Stable state	A-B changed	A sends tables to B, C	B sends tables to C	C sends tables to B
Node A	B: 4, 51 C: 5, 50	B: 60, 51 C: 61, 50	B: 60, 51 C: 61, 50	B: 60, 51 C: 61, 50	B: 60, 51 C: 61, 50
Node B	A: 4, ∞ C: ∞, 1	A: 60, ∞ C: ∞, 1	A: 60, ∞ C: 110, 1	A: 60, ∞ C: 110, 1	A: 60, 51 C: 110, 1
Node C	A: 50, 5 B: 54, 1	A: 50, 5 B: 54, 1	A: 50, 5 B: ∞, 1	A: 50, 61 B: ∞, 1	A: 50, 61 B: ∞, 1

Link cost changes here

Note: this converges after C receives another update from B

CONFRONTO LINK STATE - DISTANCE VECTOR

	Link State	Distance Vector
size of (update) routing info	small, contains only neighbours' link costs ☺	potentially long distance vectors
communication overhead	flood to all nodes – overhead O(N*E), where N = # of nodes, E = # of edges	send distance vectors only to neighbours – O(N*K) if each of N routers has K neighbours ☺
convergence speed	do NOT need to recalculate LSP's before forwarding ⇒ faster ☺	takes a while to propagate changes to rest of network
space requirements	maintains entire topology in a link database – O(N*K) if each of N routers has K neighbours	maintains only neighbours' states – O(K) distance vectors ☺
computational complexity per one destination	$O(N*(N-1)/2)=O(N^2)$	O(N*K*Diameter) ☺
computational robustness	each router computes paths on its own – no error propagation ☺	routers compute paths collectively – errors propagate
security / fault tolerance	false/corrupt LSPs can be flooded to all routers	false/corrupt LSPs can be flooded to all routers