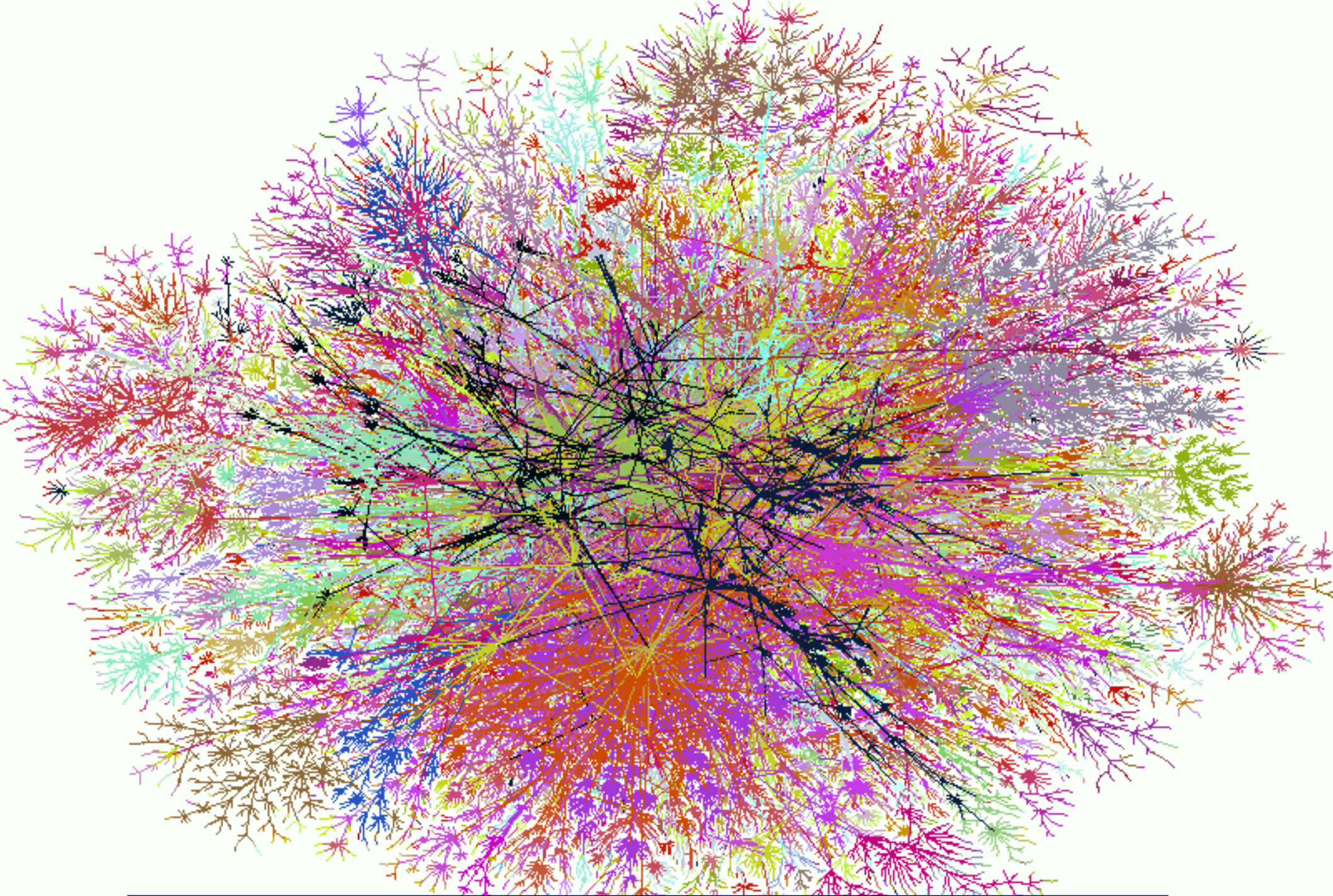


# **PARTE 4c**

## **LIVELLO IP**

**(La “dorsale” di Internet)**

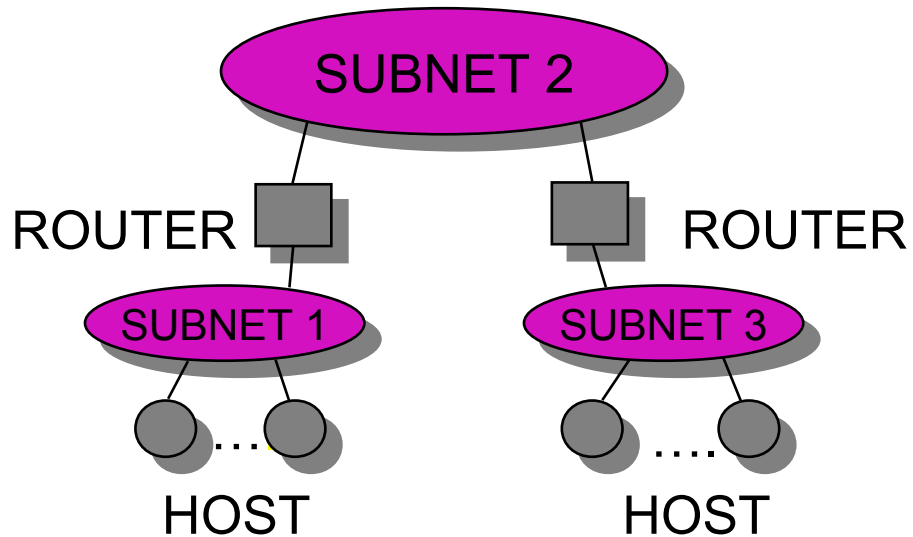


How do IP packets find their way through this mess?

# *Ricordare i servizi principali del livello IP*

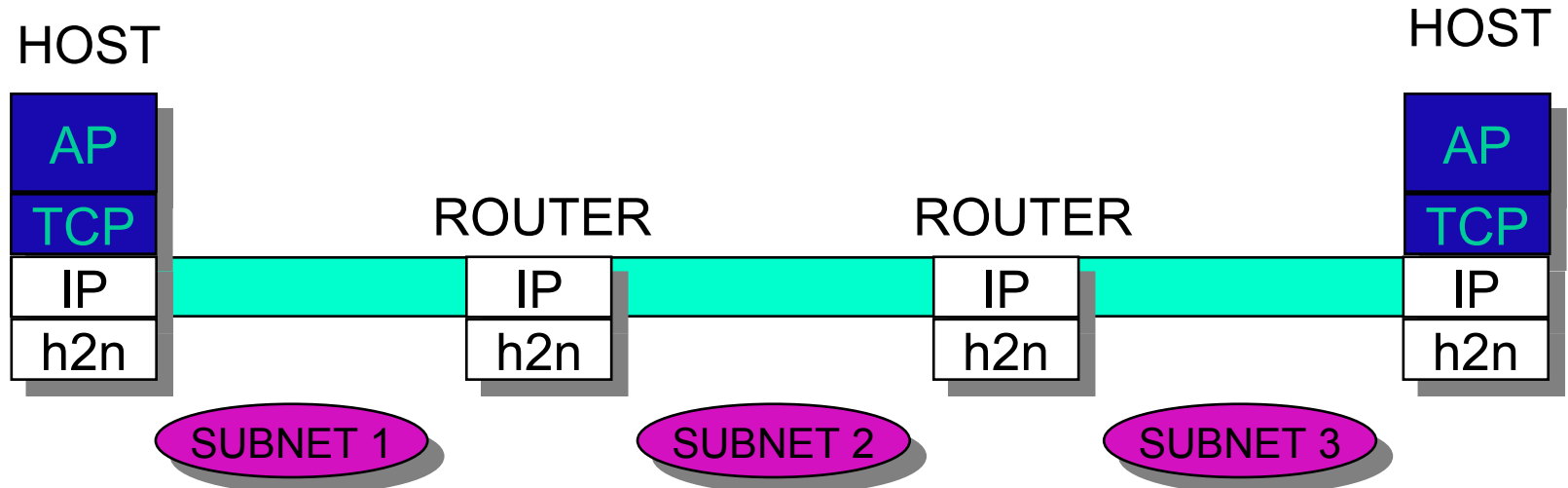
- **Indirizzamento univoco degli host**
- **Unità di trasferimento dati**
- **Architettura di Internet**
- **Funzione di routing:**
  - sceglie il percorso nella rete attraverso il quale consegnare i pacchetti
  - consegna i pacchetti da un host a un altro, ma in modo best effort, privo di connessione, e quindi non garantito

# Distinguere i due casi fondamentali



Host mittente e destinatario sulla stessa sottorete  
(*netid uguali*)

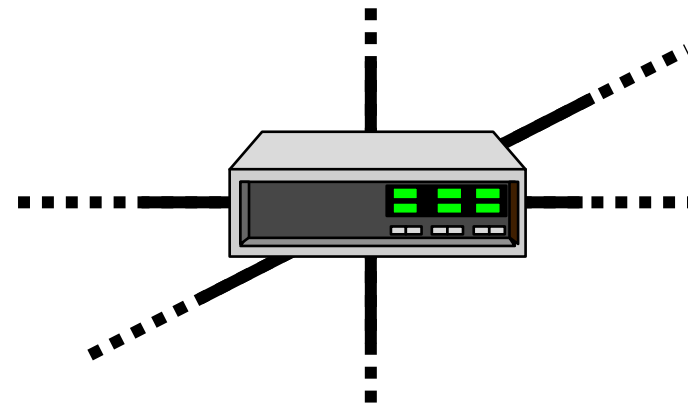
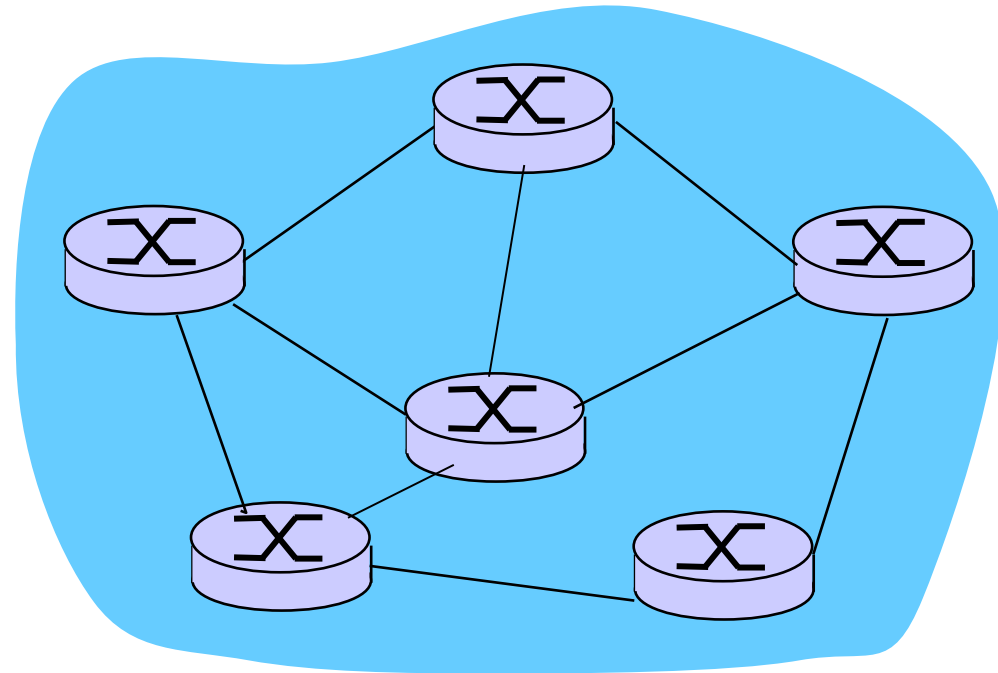
Host mittente e destinatario su sottoreti differenti  
(*netid differenti*)



## **Modulo 8: Routing e router**

# Router

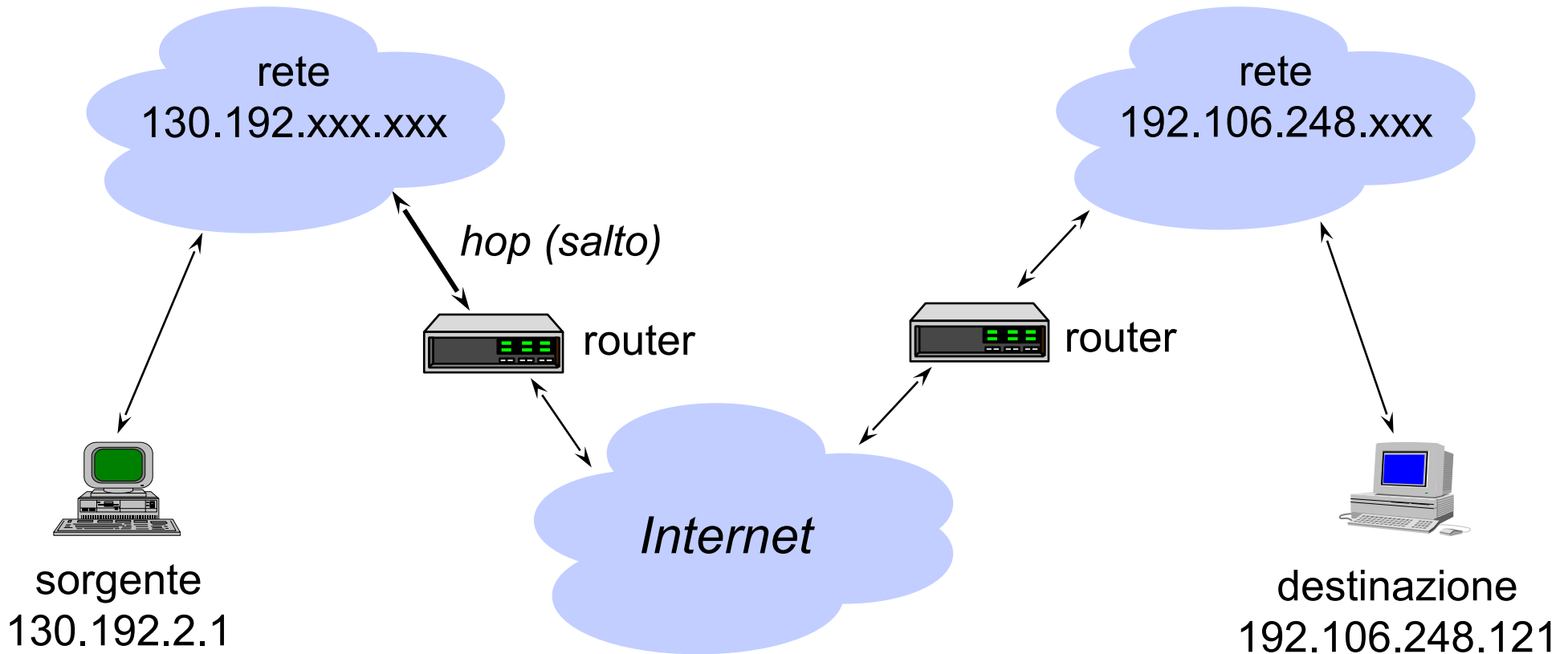
Il router deve risolvere un problema molto ben definito: Instradare i pacchetti nella rete da un qualsiasi host ad un qualsiasi altro host



# *Problema del routing*

- **Instradare i pacchetti nella rete da un qualsiasi host ad un qualsiasi altro host è un problema complesso**
- **Quando un problema è complesso si suddivide in sottoproblemi più semplici:**
  - Sottoproblema 1: ad ogni hop inoltrare il pacchetto ad un altro nodo in modo che si avvicini alla destinazione (IP forwarding)
  - Sottoproblema 2: mantenere informazioni aggiornate per poter risolvere il sottoproblema1 (Gestione delle tabelle di routing)

# Routing IP



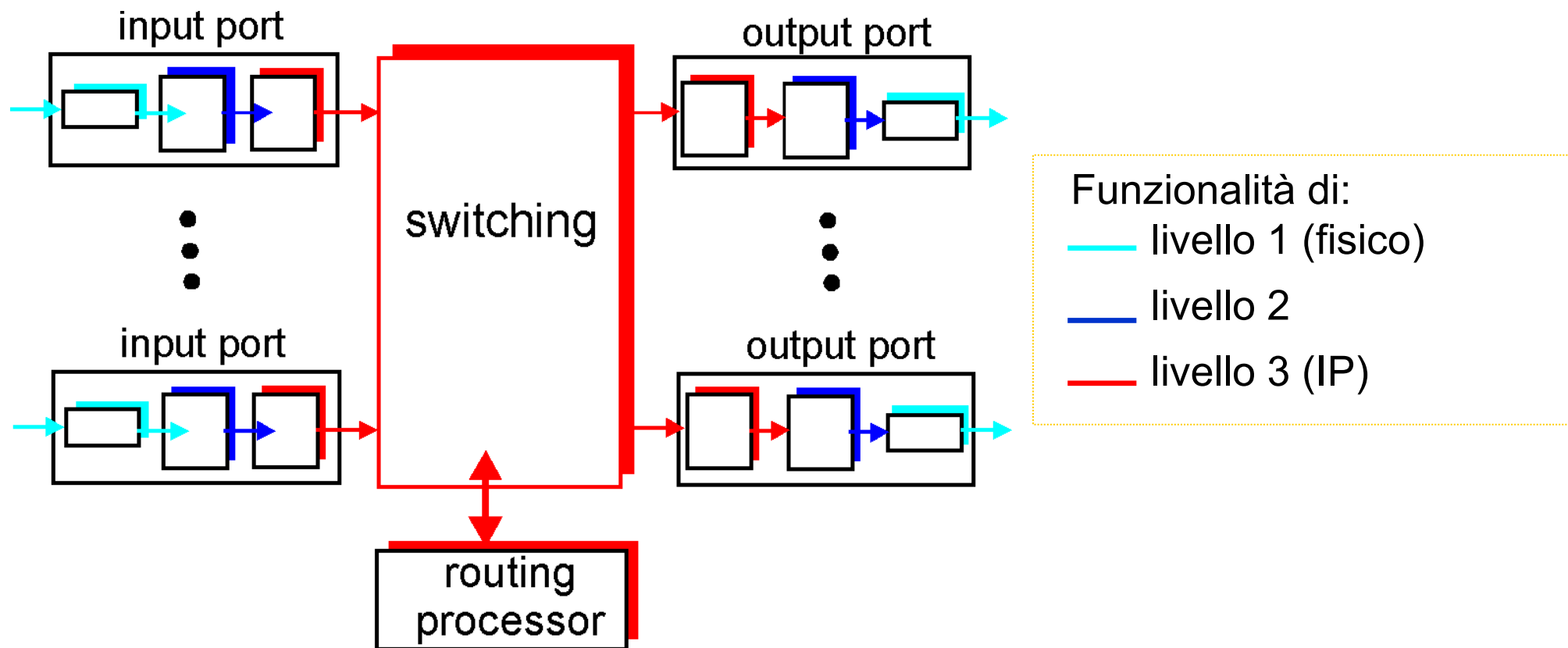
- I router si passano i pacchetti come una “patata bollente”: bisogna conoscere solo l’indirizzo del prossimo *hop*
- A volte il routing non ha successo perché i router sovraccarichi scartano pacchetti (*limite fisico*) o vi possono errori di routing (*errore logico*)



# Architettura di un router

4 componenti fondamentali nell'architettura di un router:

- porta di ingresso
- commutatore
- processore di routing
- porta di uscita

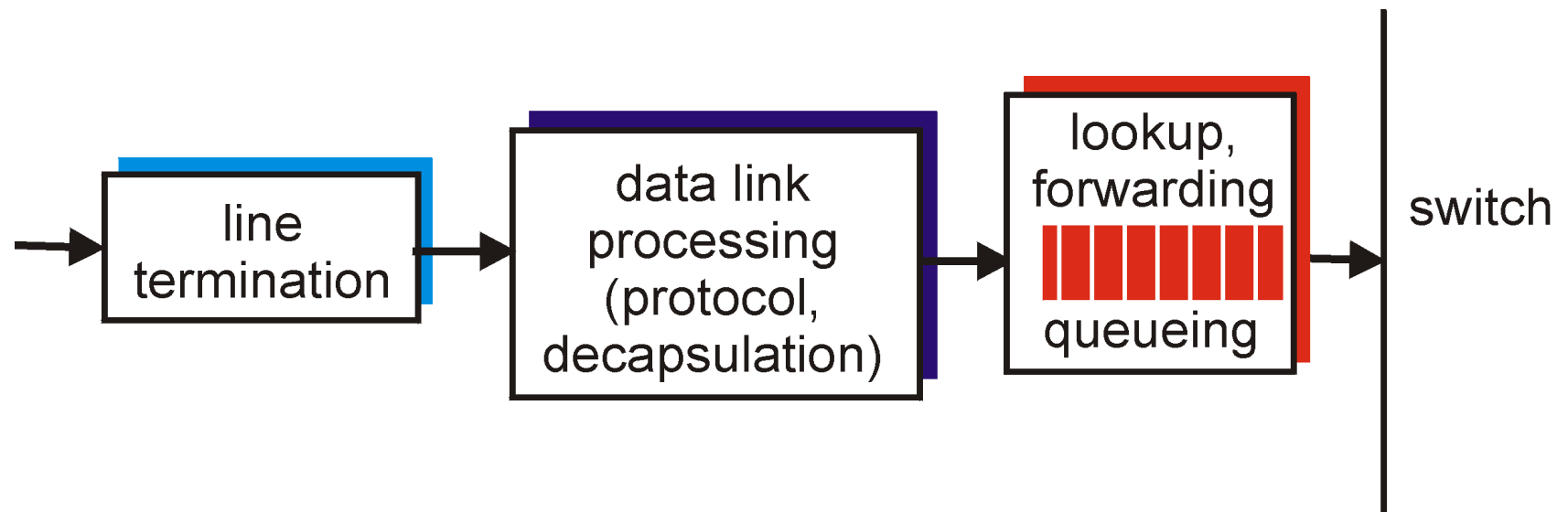


# Architettura di un router

## Porta di ingresso:

- funzioni del livello 1
- funzioni del livello 2
- funzioni del livello 3 → funzioni di ricerca e forwarding della porta di uscita; ottimizzazione della ricerca nella tabella di routing

} associate ad un singolo link di ingresso

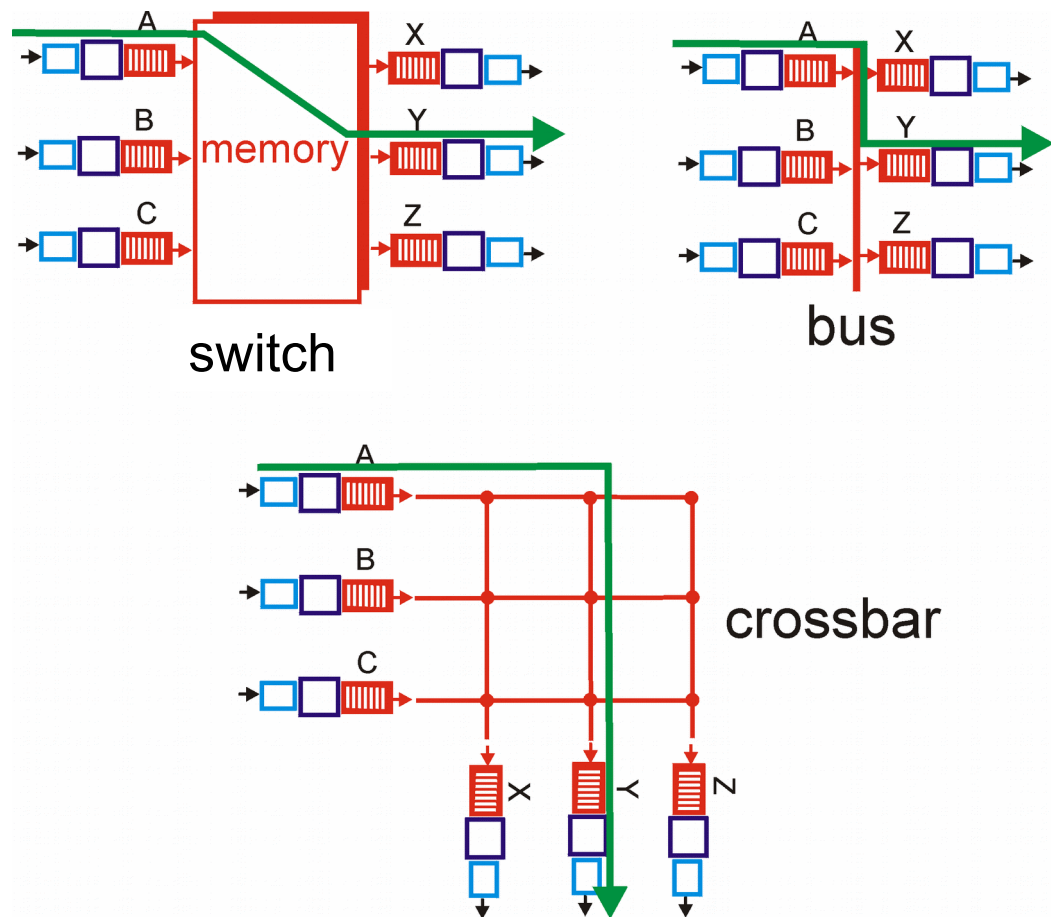


# Architettura di un router

## Componenti di switching

FUNZIONE: spostamento del pacchetto dalla porta di ingresso a quella di uscita “opportuna”

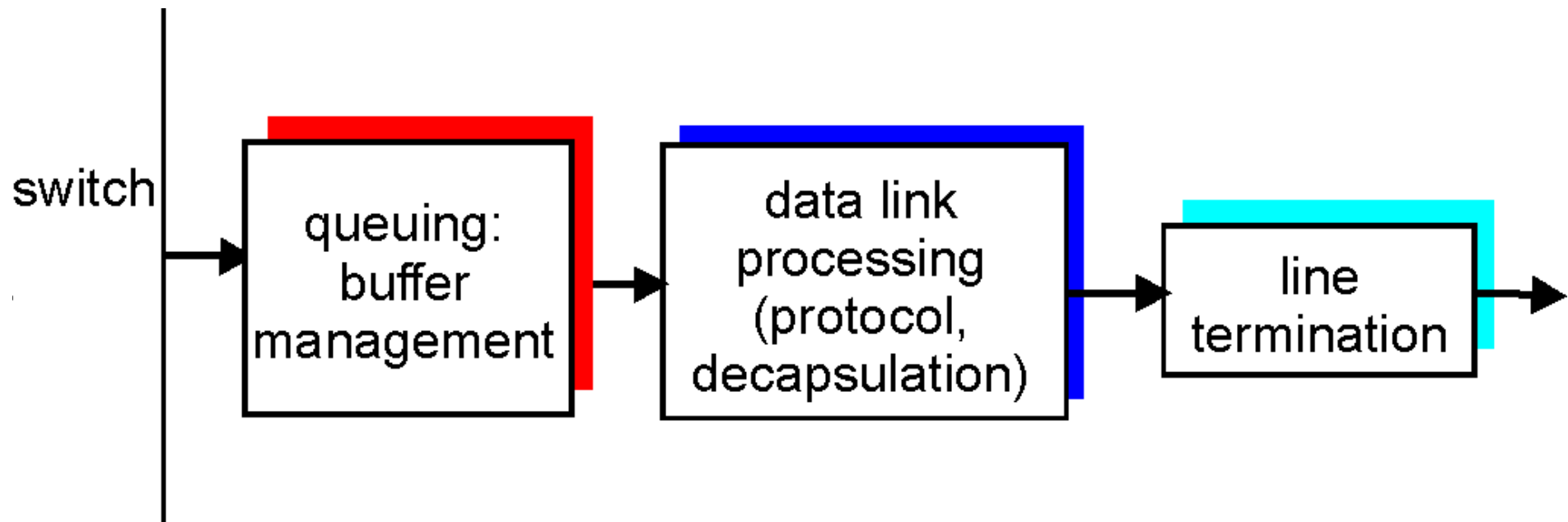
TECNICHE: Commutazione basata su switch, bus o rete di interconnessione crossbar



# Architettura di un router

## Porta di uscita:

- funzioni del livello 1
  - funzioni del livello 2
  - funzioni del livello 3 → funzioni di gestione della coda e del buffer di uscita (la velocità con cui il commutatore consegna i pacchetti deve essere superiore alla capacità del link di uscita)
- } associate con un singolo link di uscita



# *IP Forwarding*

- **IP forwarding (inoltro):** meccanismo con cui un router trasferisce i pacchetti da un'interfaccia d'ingresso a quella in uscita
- **Effettuato da ogni router**
- **Il next-hop router appartiene ad una rete alla quale il router è collegato direttamente**
- **Per inoltrare i pacchetti:**
  - l'indirizzo di destinazione viene estratto dall'header del pacchetto
  - l'indirizzo di destinazione è usato come indice nella tabella di routing

# *Tabella di routing*

- **Ogni host e ogni router hanno una tabella di routing in cui ciascuna riga fornisce il next-hop per ogni possibile destinazione**
- **Le tabelle di routing possono avere anche più di 50000 righe**
- **Le dimensioni (crescenti) delle tabelle di routing potrebbero essere un limite allo sviluppo di Internet**
- **In realtà, una riga può fornire informazioni per molte destinazioni con l'utilizzo di tecniche di aggregazione**

# Tecniche di aggregazione

- **Utilizzo del solo netid** per un insieme di indirizzi IP di destinazione che condividono lo stesso next-hop, cioè le reti per le quali il next-hop è rappresentato dallo stesso router
- Nel caso in cui gli indirizzi di rete che possono essere aggregati non condividono lo stesso next-hop, si aggregano gli indirizzi aggregabili con prefissi diversi e seguendo il principio del **longest prefix matching** (ordinando la tabella mettendo prima le maschere più lunghe e poi le più corte)

# Tecniche di aggregazione

- Si organizzano le tabelle creando un'organizzazione gerarchica che riflette l'architettura di Internet
- E' possibile sfruttare anche il routing geografico, sapendo che le classi di indirizzi IP vengono assegnati in funzione della posizione geografica continentale
- Uso del router di default: si definisce un router di default comune a più indirizzi di destinazione



# *Caratteristiche delle tabelle di routing*

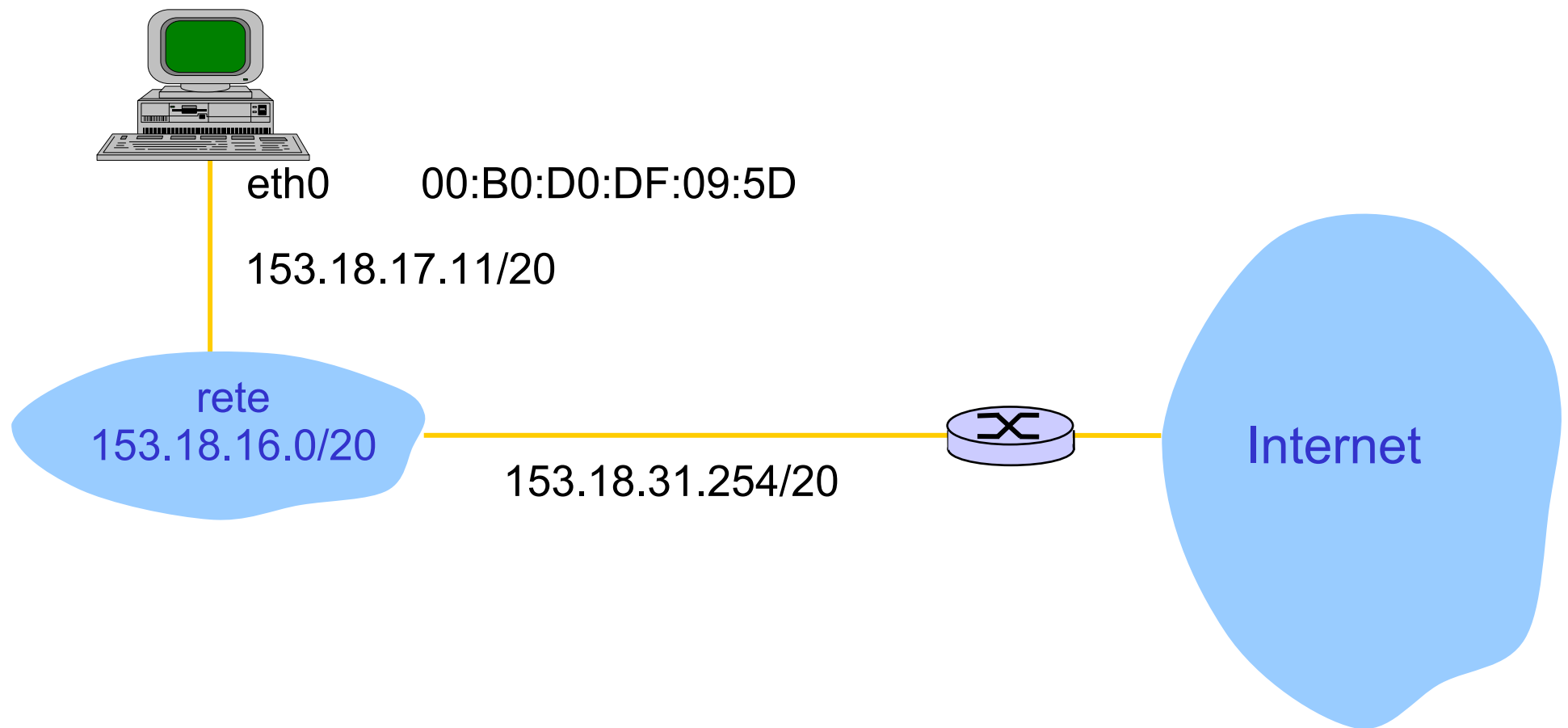
- **Routing statico: la tabella di routing non è modificata dal router**
  - L'amministratore di rete deve inserire o modificare righe della tabella di routing
  - Svantaggio: impossibilità di reagire automaticamente ai cambiamenti topologici
  - Da utilizzare nel caso di reti piccole con pochi cambiamenti
- **Routing dinamico: la tabella di routing è modificata dal router al variare delle condizioni sulla rete (stato di funzionamento degli apparati e dei collegamenti)**
  - Lo scambio di informazioni ed il calcolo dei valori della tabella di routing avvengono mediante qualche protocollo di routing: RIP, OSPF, BGP

# Funzionamento del router

- **Estrae l'indirizzo IP del destinatario D dall'header del pacchetto e determina il suo netid N**
  - Se N corrisponde ad una rete connessa direttamente al router, consegna il pacchetto al destinatario D sulla rete (ciò comporta la risoluzione di D nel corrispondente indirizzo fisico e l'invio del frame)
  - Se la tabella contiene un router per la rete N, invia il pacchetto al next-hop router specificato nella tabella
  - Se la tabella contiene un router di default, invia il pacchetto a quel router
  - Altrimenti, si verifica un errore di routing

# Cosa dedurre dalle info precedenti?

## → La topologia minima della rete



# *Caratteristiche del forwarding IP*

- **Indipendenza dal mittente: il next-hop routing non dipende (tipicamente) dal mittente del pacchetto o dal cammino che il pacchetto ha attraversato fino a quel momento**
  - Il router estrae dal pacchetto soltanto l'indirizzo del destinatario
- **Routing universale: la tabella di routing deve contenere un next-hop router per ciascuna destinazione**
- **Routing ottimo: il next-hop router è scelto in modo da minimizzare il cammino verso la destinazione → utilizzo degli algoritmi di routing**

# Next-hop forwarding

- Next-hop forwarding: il router possiede l'informazione sul salto successivo (next-hop) che il pacchetto deve compiere per giungere a destinazione
- Il next-hop router appartiene ad una rete alla quale il router è collegato direttamente



netid	router
10.0.0.0/8	diretto
20.0.0.0/8	diretto
30.0.0.0/8	20.0.0.6

Tabella di routing per  
il router R

# Esempi di IP forwarding

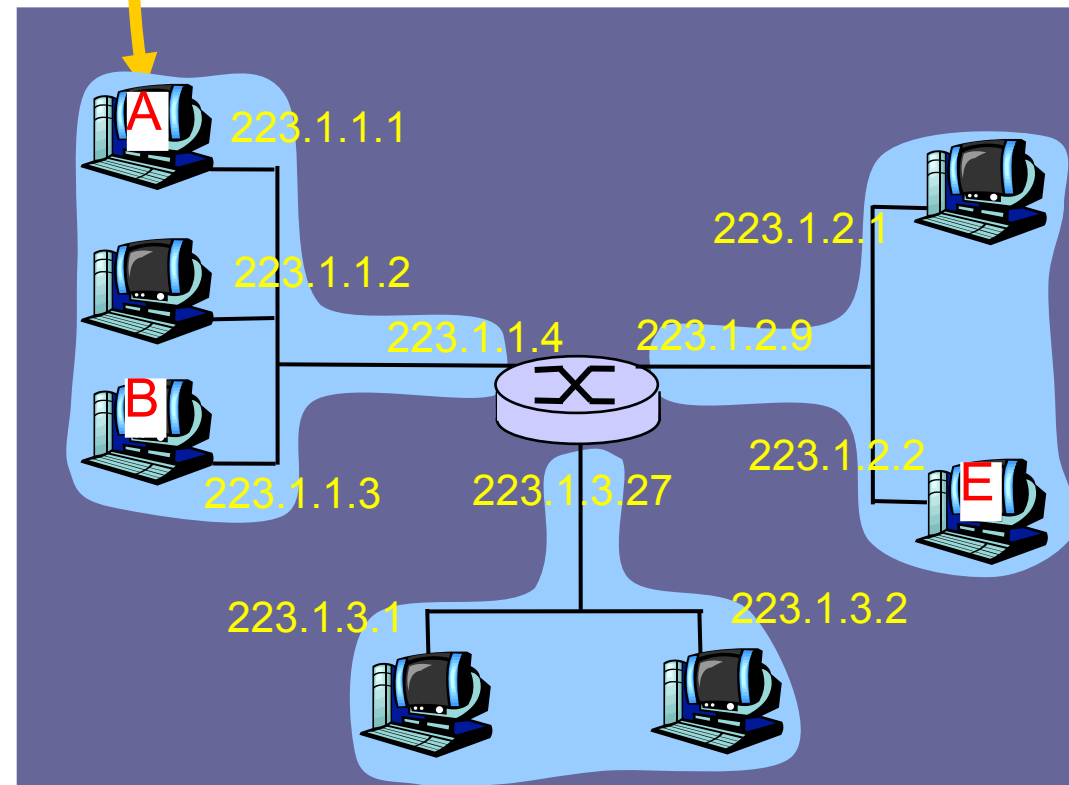
- Il pacchetto non viene modificato nel tragitto da mittente a destinatario
- Campi address di interesse

IP datagram:

misc fields	source IP addr	dest IP addr	data
-------------	----------------	--------------	------

Tabella di routing in A

Dest. Netid	Next router	Nhops
223.1.1.0/24		1
223.1.2.0/24	223.1.1.4	2
223.1.3.0/24	223.1.1.4	2



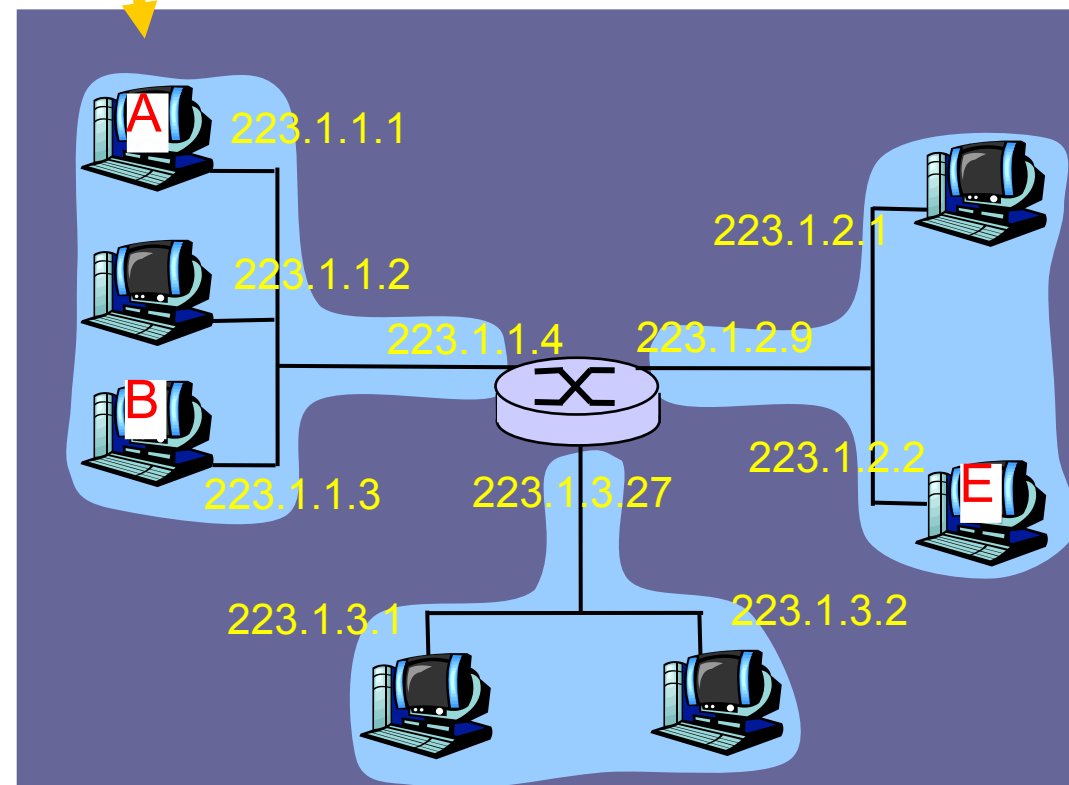
# Esempi di IP forwarding: $A \rightarrow B$

misc fields	223.1.1.1	223.1.1.3	data
-------------	-----------	-----------	------

Partendo da A, se il pacchetto è destinato a B:

- ricerca l'indirizzo di rete di B
- B è sulla stessa rete di A
- il livello host-to-network invia il pacchetto direttamente a B in un frame

Dest. Netid	Next router	Nhops
223.1.1.0/24		1
223.1.2.0/24	223.1.1.4	2
223.1.3.0/24	223.1.1.4	2



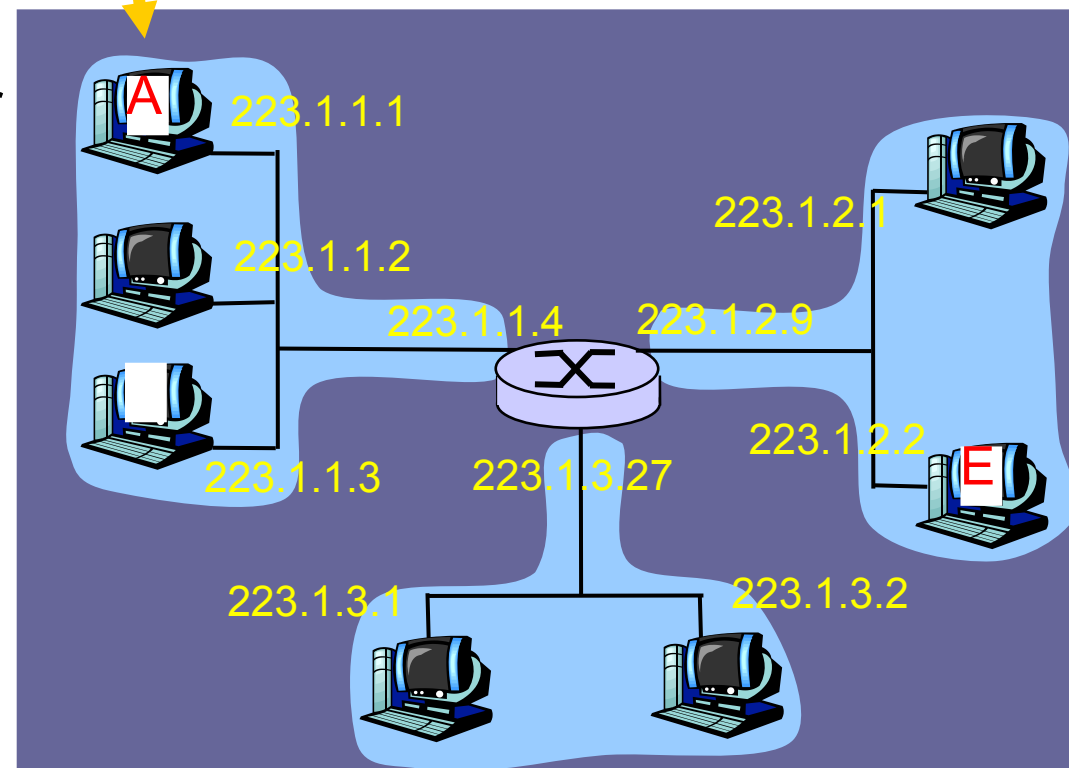
# Esempi di IP forwarding: A → E

misc fields	223.1.1.1	223.1.2.2	data
-------------	-----------	-----------	------

Partendo da A, destinazione E:

- ricerca l'indirizzo di rete di E
- E è su una rete diversa
- routing table: next hop router per E è 223.1.1.4
- il livello host-to-network invia il pacchetto al router 223.1.1.4 in un frame
- il pacchetto arriva a 223.1.1.4
- *segue...*

Dest. Netid	Next router	Nhops
223.1.1.0/24		1
223.1.2.0/24	223.1.1.4	2
223.1.3.0/24	223.1.1.4	2





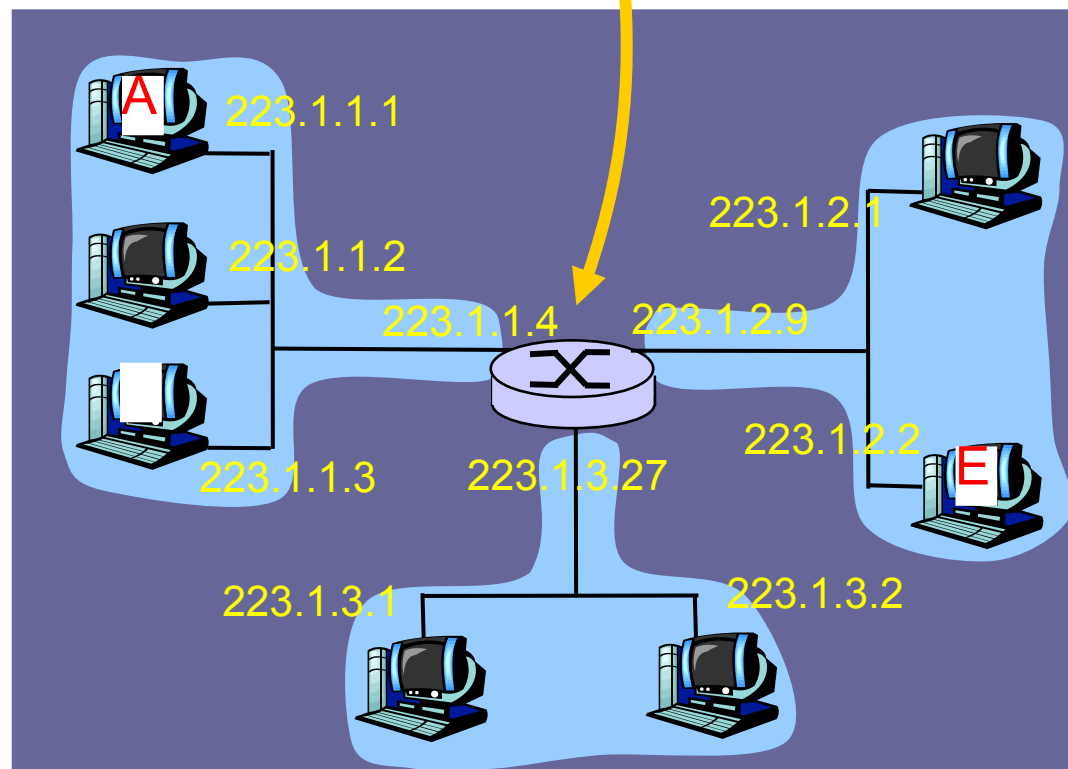
# Esempi di IP forwarding: A → E

misc fields	223.1.1.1	223.1.2.2	data
-------------	-----------	-----------	------

Il frame destinato ad E arriva a 223.1.1.4

- ricerca l'indirizzo di rete di E
- E è sulla stessa rete della interfaccia del router 223.1.2.9
- il livello host-to-network invia il pacchetto a 223.1.2.2 in un frame
- il pacchetto arriva a 223.1.2.2

Dest. Netid	Next router	Nhops	Interfaccia
223.1.1.0/24	-	1	223.1.1.4
223.1.2.0/24	-	1	223.1.2.9
223.1.3.0/24	-	1	223.1.3.27



**Modulo 9:  
Basi teoriche per il routing**

# *Algoritmi di routing*

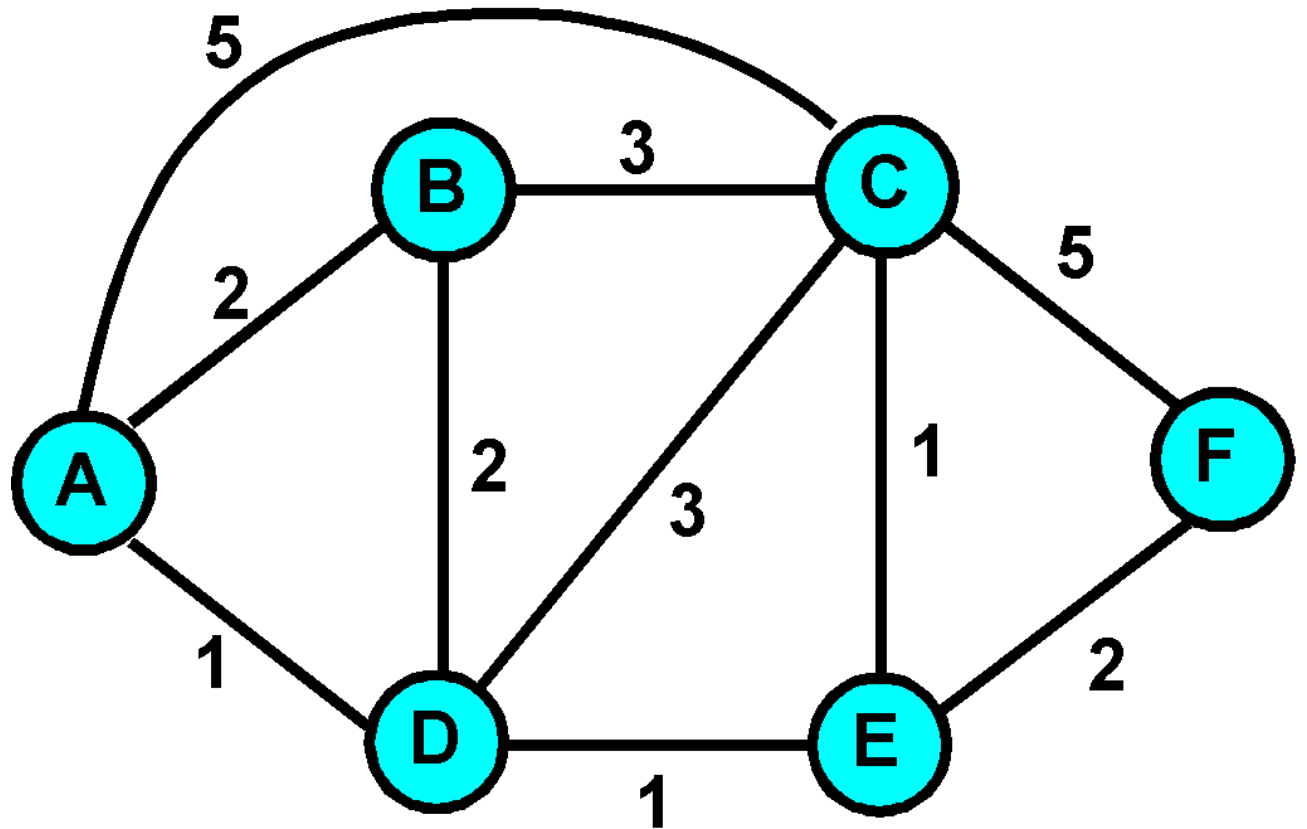
- **OBIETTIVO: Determinare il percorso ottimale**
- **Dato un insieme di router interconnessi, determinare il cammino ottimale dall'host mittente all'host destinatario**
- **Cammino ottimale → costo minimo**

# *Algoritmi di routing*

- Per formulare un algoritmo di routing, si modella la rete tramite un grafo pesato  $G(N,E)$  dove:
  - i nodi  $N$  rappresentano i router (oppure gli AS)
  - gli archi rappresentano le connessioni tra i router
  - le etichette  $E$  degli archi rappresentano il “costo” delle connessioni tra i router

# Algoritmi di routing

*Etichetta sull'arco:*  
"costo" (*tempo*) per  
l'invio di un pacchetto



Qual è il cammino minimo (e il suo costo) tra A e F?  
Qual è il cammino minimo (e il suo costo) tra A e C?

# ***Fattori che influenzano il routing***

- ***Fattori statici: topologia della rete***
- ***Fattori dinamici: traffico della rete, guasti***
- **Politiche di routing**

# *Principali algoritmi di routing*

- **Algoritmi di routing distribuito**
  - Nessun nodo ha un'informazione completa del costo di tutti i link della rete
  - Es., Distance vector protocol
  
- **Algoritmi di routing centralizzato**
  - Ogni nodo possiede un'informazione globale sulla rete
  - Es., Link state protocol

# **Modulo 9a: Distance vector protocol**



# *Algoritmi Distance Vector*

- **Usati nel primo periodo di Internet (ARPANET)**
- **Calcolo distribuito del next hop**
  - Algoritmo adattativo rispetto a cambi di stato
- **Unità di scambio dell'informazione:**
  - Vettore di distanze rispetto alle varie destinazioni
- **Esempio principale di implementazione:**
  - Algoritmo Bellman-Ford distribuito

# Algoritmo Bellman-Ford: premessa

- **Ogni nodo:**
  - aggiorna il proprio vettore di distanze in risposta a variazioni di costi sui link adiacenti
  - invia un aggiornamento ai nodi adiacenti se il proprio vettore di distanze cambia
- **Ogni nodo  $x$  mantiene i dati:**
  - $c(x,v)$  costo del link tra nodo  $x$  e nodo  $v$
  - $D_x = [D_x(y) : y \in N]$  vettore di distanze del nodo  $x$  verso tutti i nodi  $y$  nella rete  $N$
  - $D_v = [D_v(y) : y \in N]$  vettori di distanze dei vicini  $v$  di  $x$

# Algoritmo Bellman-Ford: premessa

- **Si usa la formula di Bellman-Ford per il calcolo del costo minimo tra x e y:**
- **$Dx(y) = \min_v \{ c(x,v) + Dv(y) \}$** 
  - dove  $\min_v$  è calcolato tra tutti i vicini v del nodo x
- **Intuitivamente, la formula è molto chiara:**
  - tra tutti i nodi v adiacenti al nodo x, il percorso da scegliere è quello che mi porta con il minor costo da v a y,
  - a meno che (da cui la considerazione del primo addendo) il costo tra x e v sia talmente alto che mi conviene percorrere altre strade

# Algoritmo Bellman-Ford

## Start

Per tutte le destinazioni  $y$  in  $N$ :

$D_x(y) = c(x,y)$  se  $y$  è adiacente

$D_x(y) = \infty$  se  $y$  non è adiacente

Invia il vettore di distanze  $D_x = [D_x(y) \mid y \text{ in } N]$  ad ogni vicino  $v$

## Loop

Attendi (finchè il costo di un collegamento verso qualche vicino  $v$  cambia o ricevi un vettore di distanze da un vicino  $v$ )

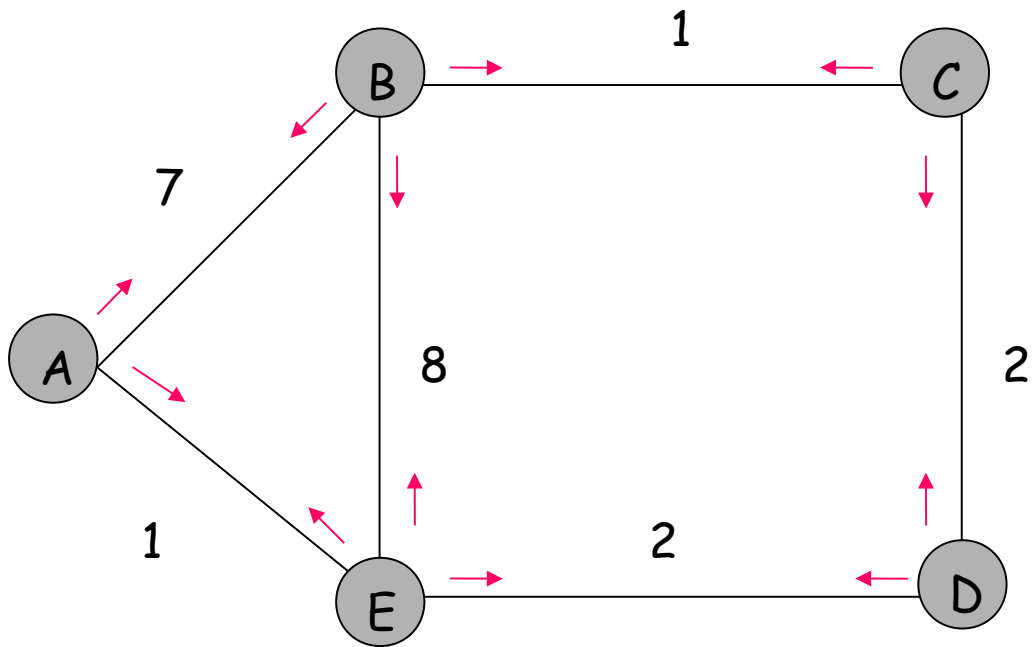
Per ogni destinazione  $y$  in  $N$ :

$$D_x(y) = \min_v \{ c(x,v) + D_v(y) \}$$

Se  $D_x(y)$  è cambiato per qualche destinazione  $y$

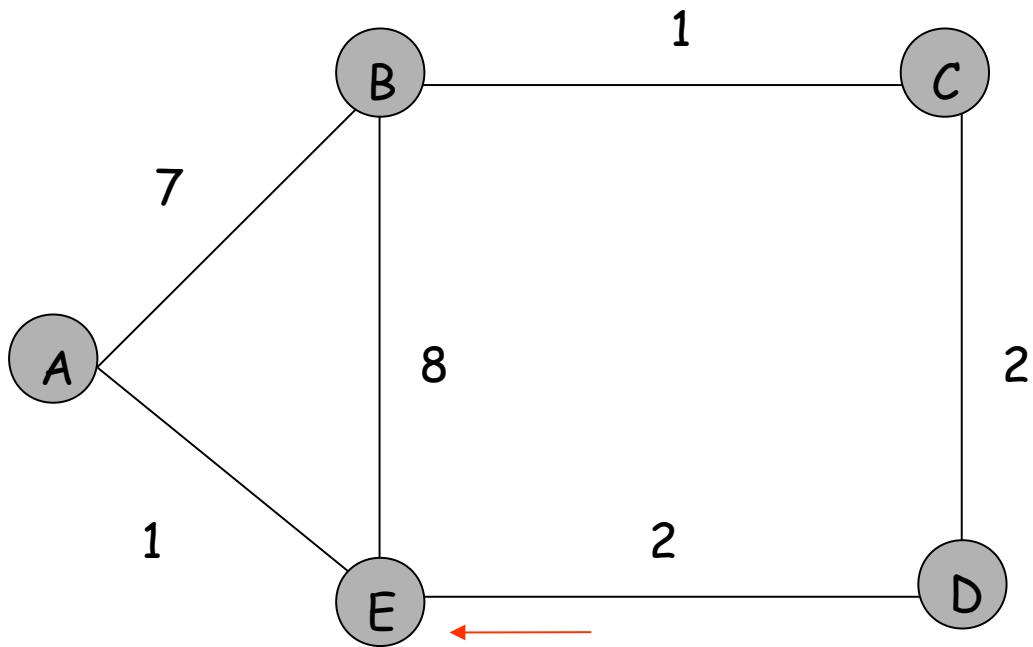
invia il vettore di distanze  $D_x = [D_x(y) : y \text{ in } N]$  a tutti i vicini

# Esempio: distanze iniziali (start)



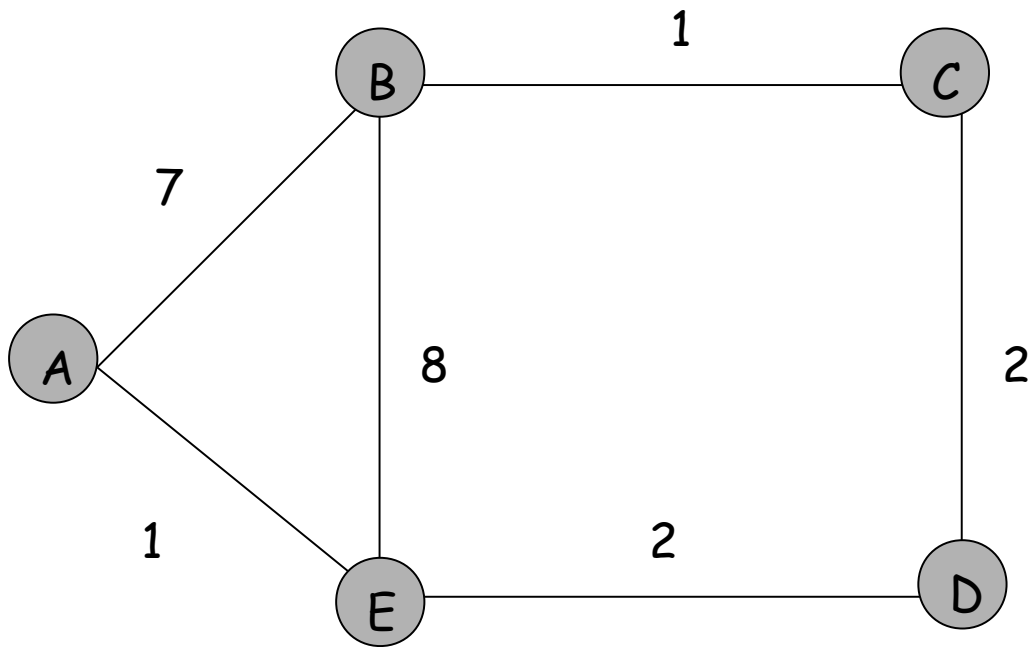
	Distanza dal nodo				
	A	B	C	D	E
A	0	7	$\infty$	$\infty$	1
B	7	0	1	$\infty$	8
C	$\infty$	1	0	2	$\infty$
D	$\infty$	$\infty$	2	0	2
E	1	8	$\infty$	2	0

# E riceve il vettore di distanze da D



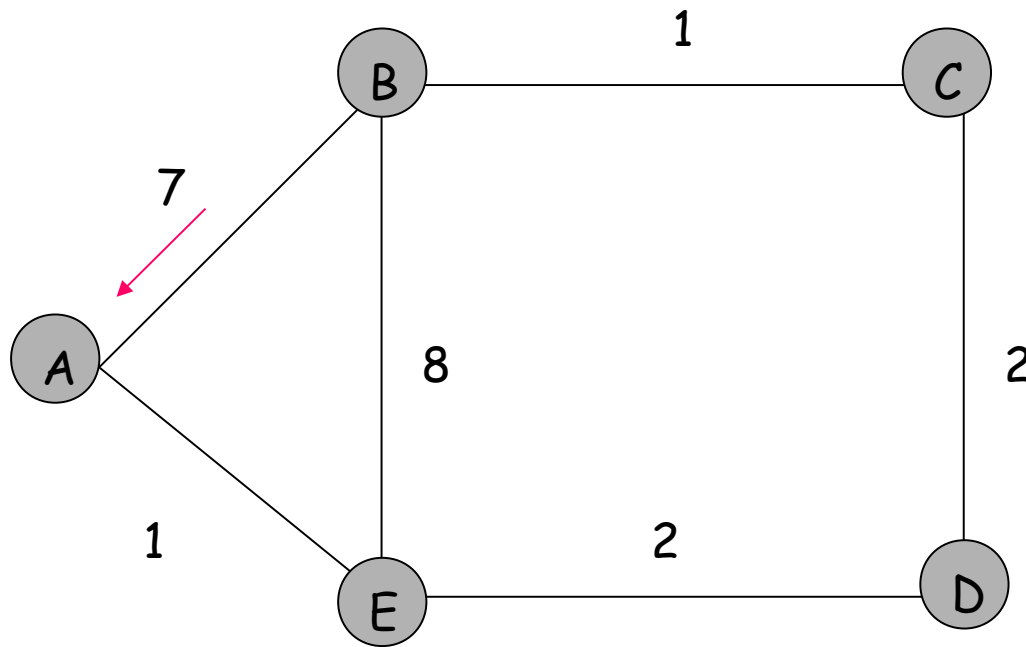
	Distanza dal nodo				
	A	B	C	D	E
A	0	7	$\infty$	$\infty$	1
B	7	0	1	$\infty$	8
C	$\infty$	1	0	2	$\infty$
D	$\infty$	$\infty$	2	0	2
E	1	8	$\infty$	2	0

# E aggiorna i costi per C



	Distanza dal nodo				
	A	B	C	D	E
A	0	7	$\infty$	$\infty$	1
B	7	0	1	$\infty$	8
C	$\infty$	1	0	2	$\infty$
D	$\infty$	$\infty$	2	0	2
E	1	8	4	2	0

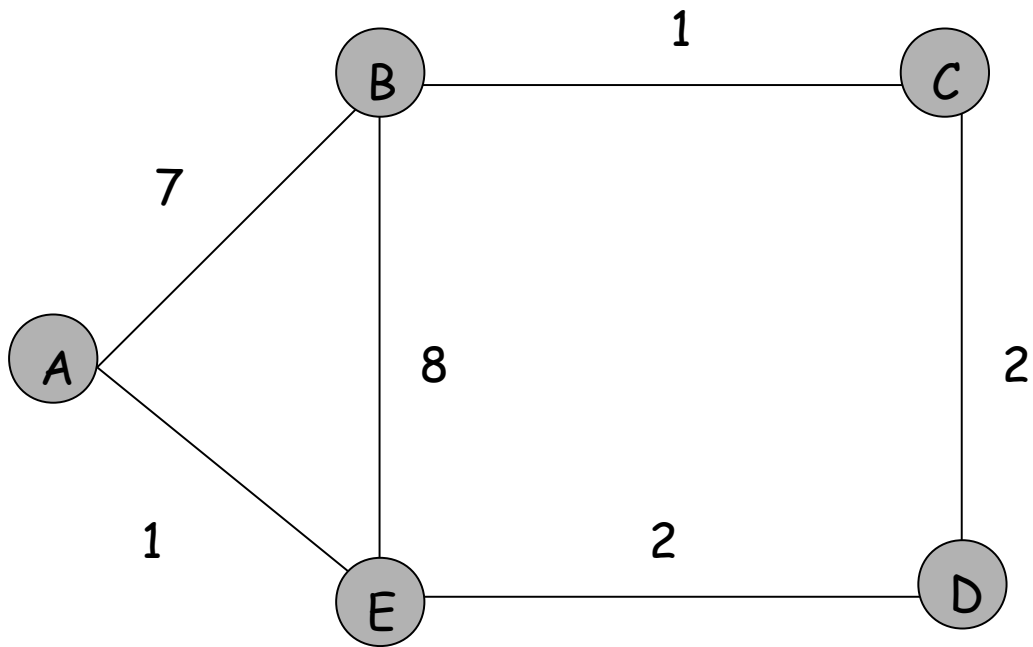
# A riceve il vettore di distanze da B



	Distanza dal nodo				
	A	B	C	D	E
A	0	7	$\infty$	$\infty$	1
B	7	0	1	$\infty$	8
C	$\infty$	1	0	2	$\infty$
D	$\infty$	$\infty$	2	0	2
E	1	8	4	2	0

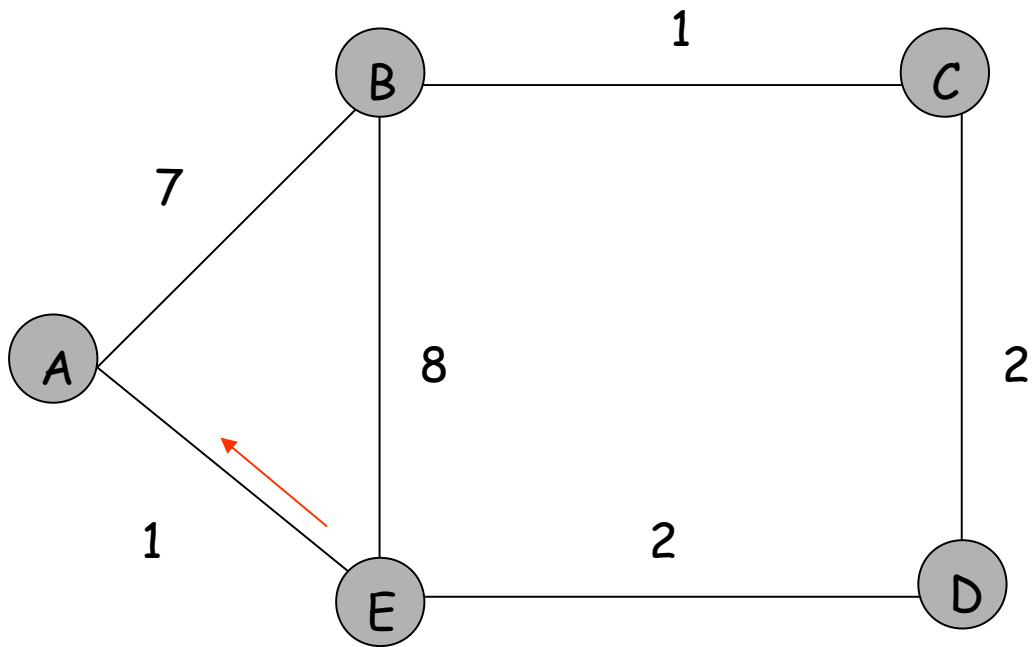


# A aggiorna i costi per C



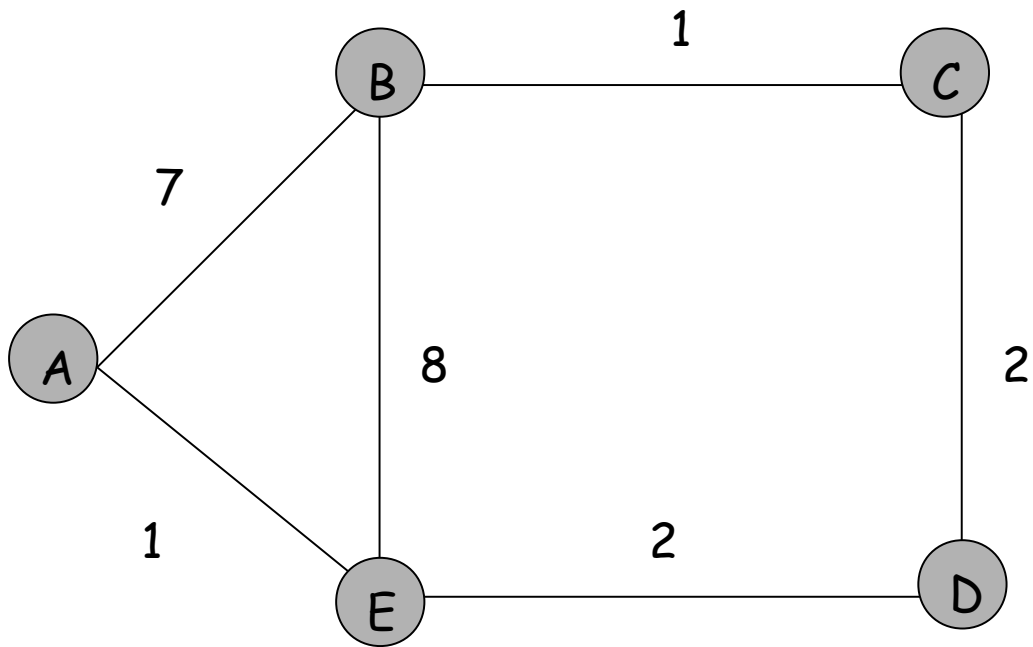
	Distanza dal nodo				
	A	B	C	D	E
A	0	7	8	$\infty$	1
B	7	0	1	$\infty$	8
C	$\infty$	1	0	2	$\infty$
D	$\infty$	$\infty$	2	0	2
E	1	8	4	2	0

# A riceve il vettore di distanze da E



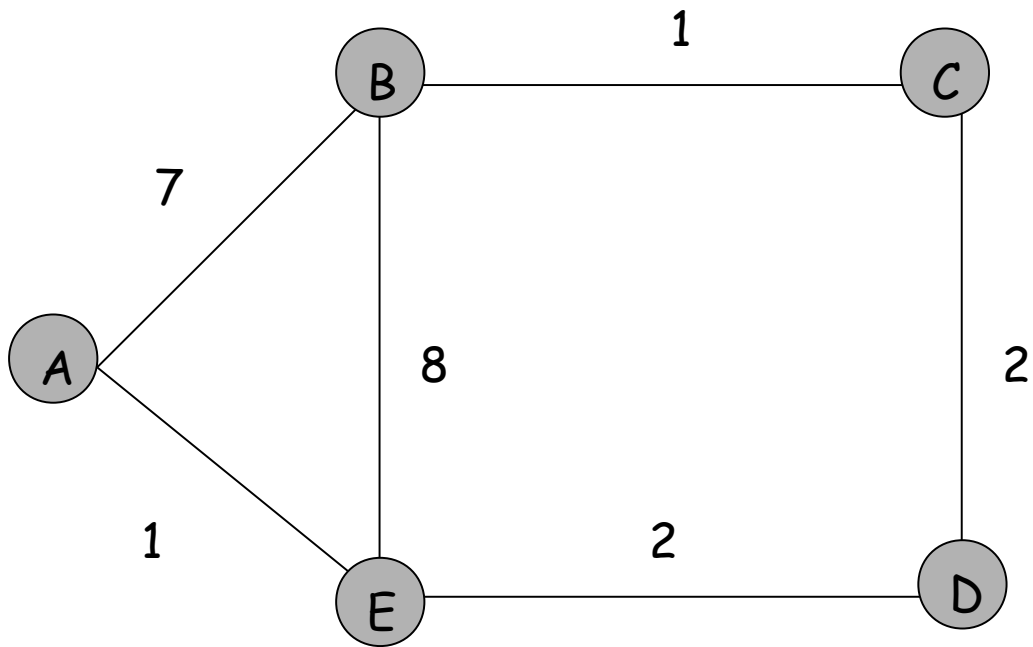
	Distanza dal nodo				
	A	B	C	D	E
A	0	7	8	$\infty$	1
B	7	0	1	$\infty$	8
C	$\infty$	1	0	2	$\infty$
D	$\infty$	$\infty$	2	0	2
E	1	8	4	2	0

# A aggiorna i costi per C e D



	Distanza dal nodo				
	A	B	C	D	E
A	0	7	5	3	1
B	7	0	1	$\infty$	8
C	$\infty$	1	0	2	$\infty$
D	$\infty$	$\infty$	2	0	2
E	1	8	4	2	0

# Distanze finali (aggiornate)



	Distanza dal nodo				
	A	B	C	D	E
A	0	6	5	3	1
B	6	0	1	3	5
C	5	1	0	2	4
D	3	3	2	0	2
E	1	5	4	2	0

# Tabella di routing

- **L' algoritmo di Bellman-Ford ha un' immediata ricaduta pratica. Serve, infatti per calcolare i valori della Tabella di routing di ciascun router**
- **La Tabella di routing del nodo x ha:**
  - una riga per ogni nodo destinazione nella rete (router o AS)
  - tante colonne quanti sono i nodi adiacenti al nodo x
  - i costi di cammino come elementi della tabella
- **In questo modo, nel momento in cui arriva un pacchetto con un indirizzo destinazione, il router può facilmente decidere su quale link inoltrarlo**

# Instradamento visto dal nodo E

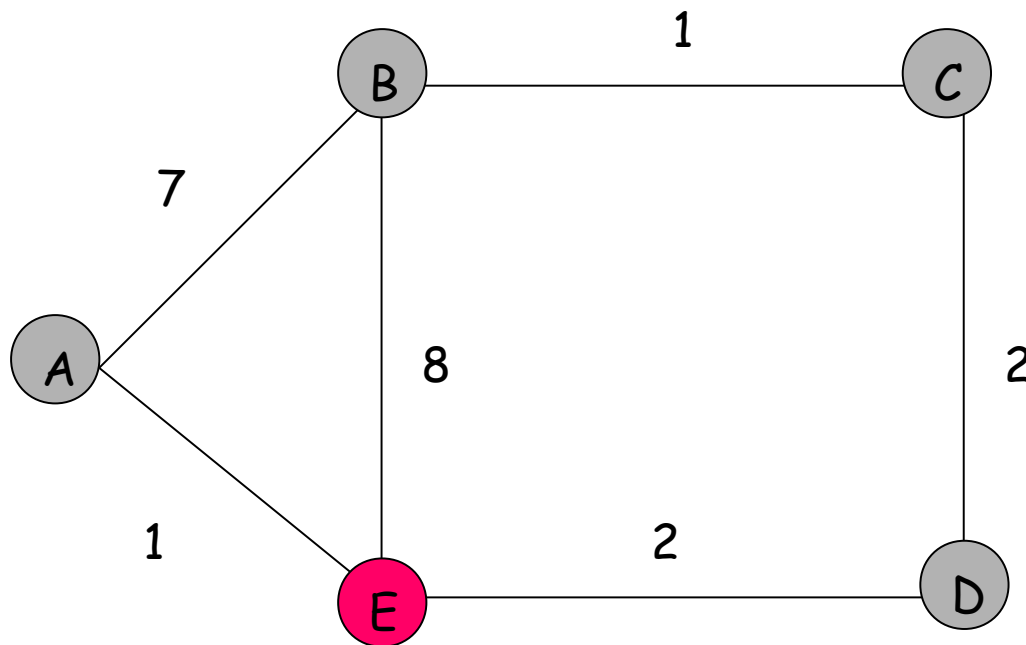


Tabella di routing di E

Dest	Next hop		
	A	B	D
A	1	14	5
B	7	8	5
C	6	9	4
D	4	11	2

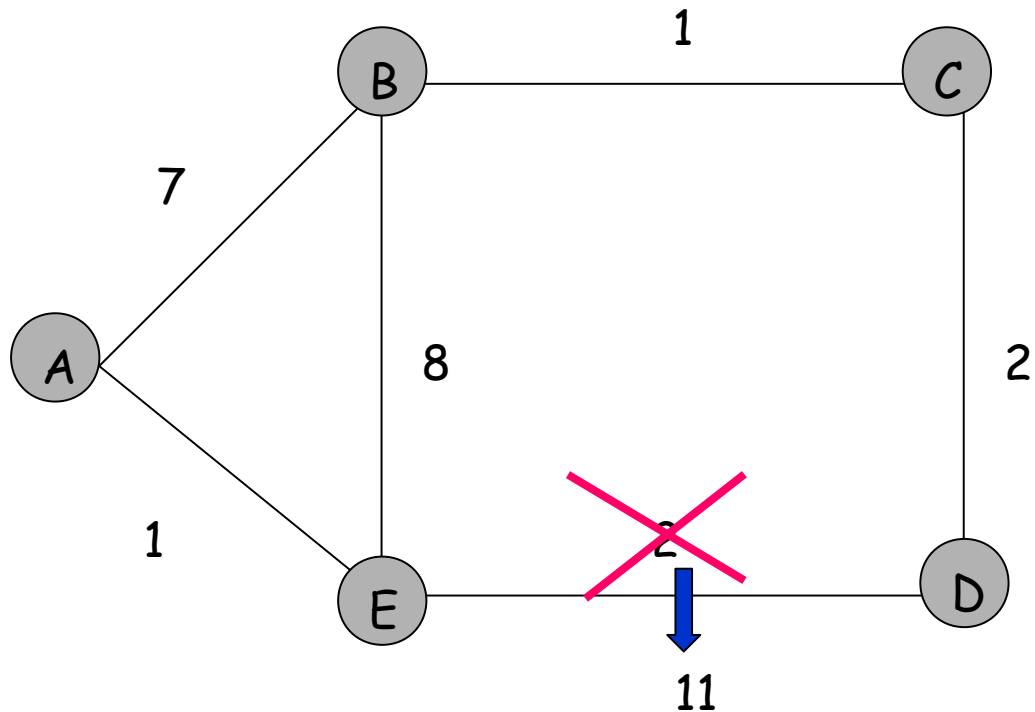
La routing table di E ha una riga per ogni destinazione nella rete e tante colonne quanti sono i nodi adiacenti al nodo stesso

I percorsi di minor costo per la corrispondente destinazione sono indicati in rosso nella *routing table*

# Osservazioni

- **Ci sono circa 1 miliardo di host e milioni di router**
- **E' credibile una tabella che riporti tutti i router di Internet come destinazione?**
- **Come si gestisce nella realtà il problema?**

# Se un link (es., E-D) ha problemi?

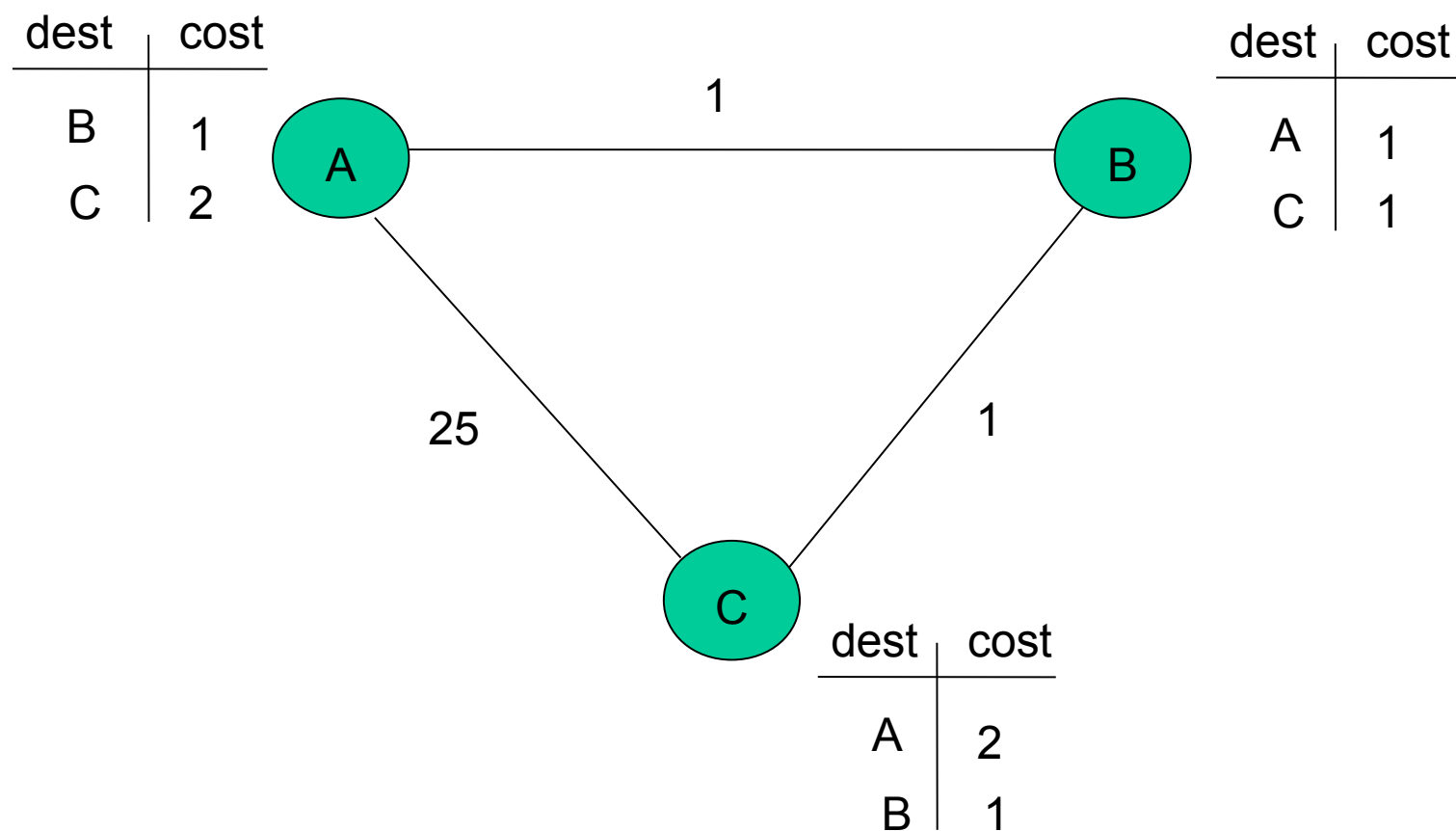


	Distanza dal nodo				
	A	B	C	D	E
A	0	7	8	10	1
B	7	0	1	3	8
C	8	1	0	2	9
D	10	3	2	0	11
E	1	8	9	11	0

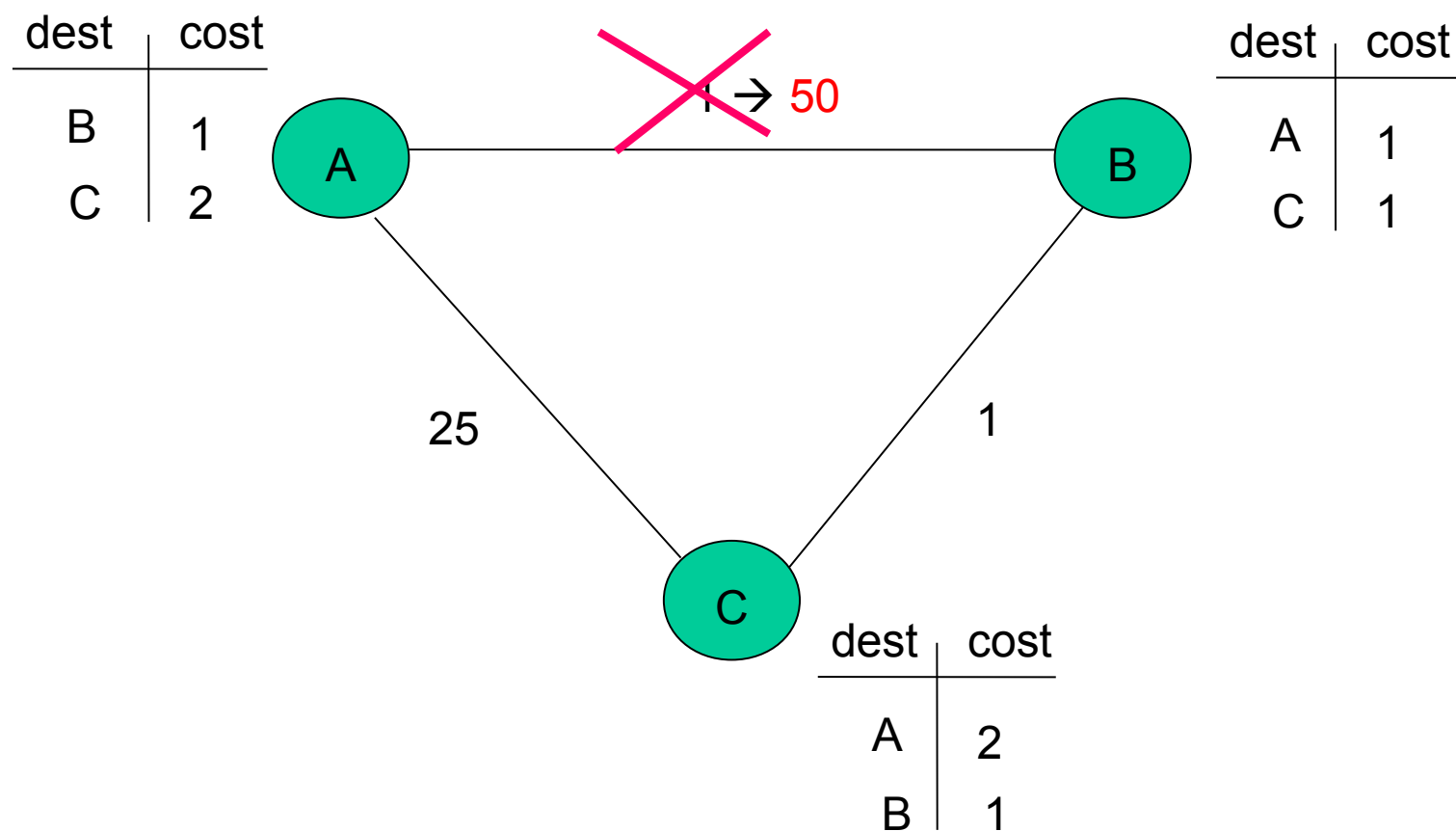
- I nodi che vertono su quel link, ricalcolano il vettore distanza
- Aggiornano la propria routing table e trasmettono il nuovo vettore ai vicini
- Ciascun nodo ricalcolerà il proprio vettore distanza e, iterativamente, lo invierà ai nodi vicini



# Rischio: effetto rimbalzo



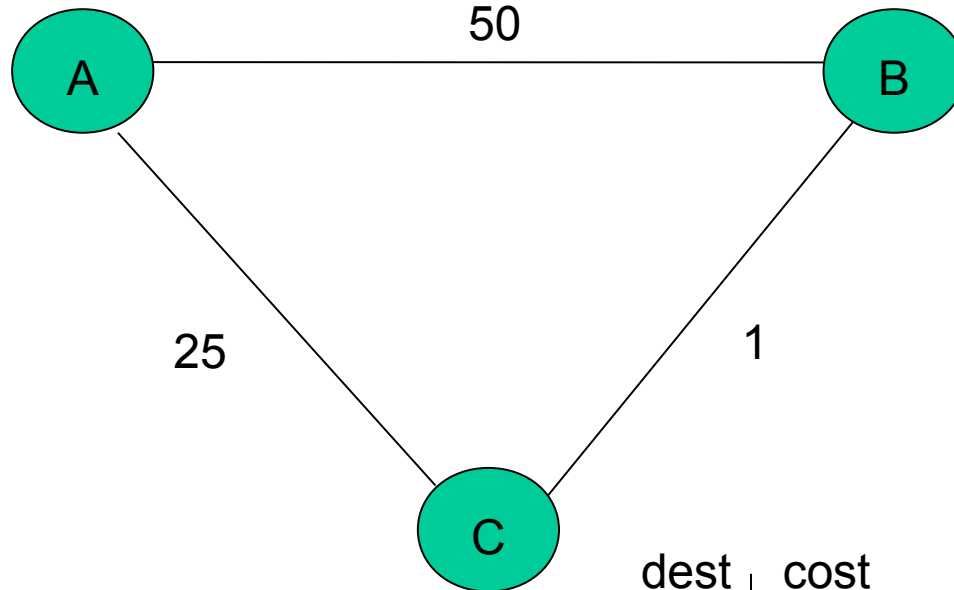
# Esempio: link A-B ha un problema



# B aggiorna le distanze per A

(tenendo conto dei dati di C, che però includeva il passaggio per B)

dest	cost
B	1
C	2

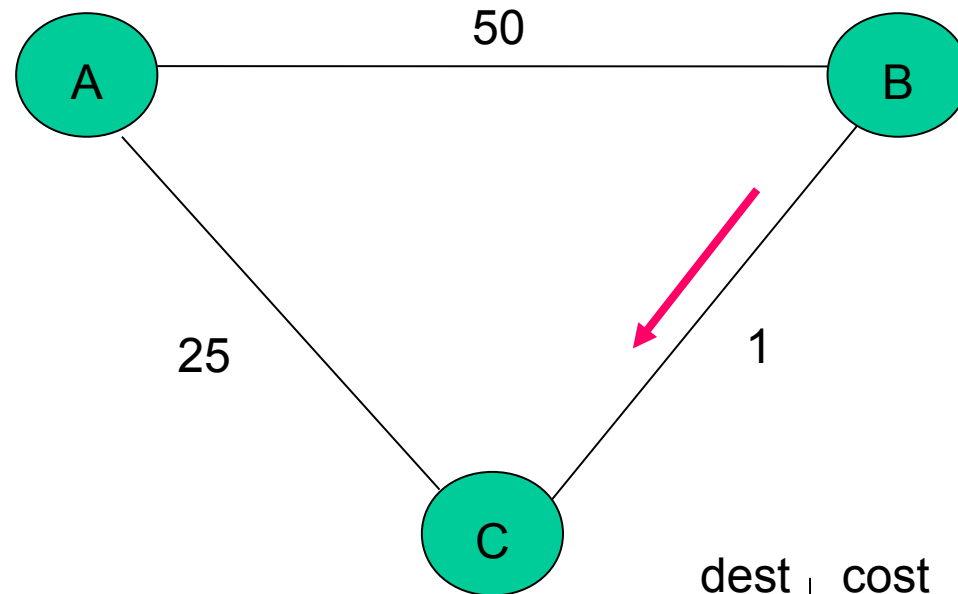


dest	cost
A	3
C	1

dest	cost
A	2
B	1

# B manda il vettore di distanze a C

dest	cost
B	1
C	2

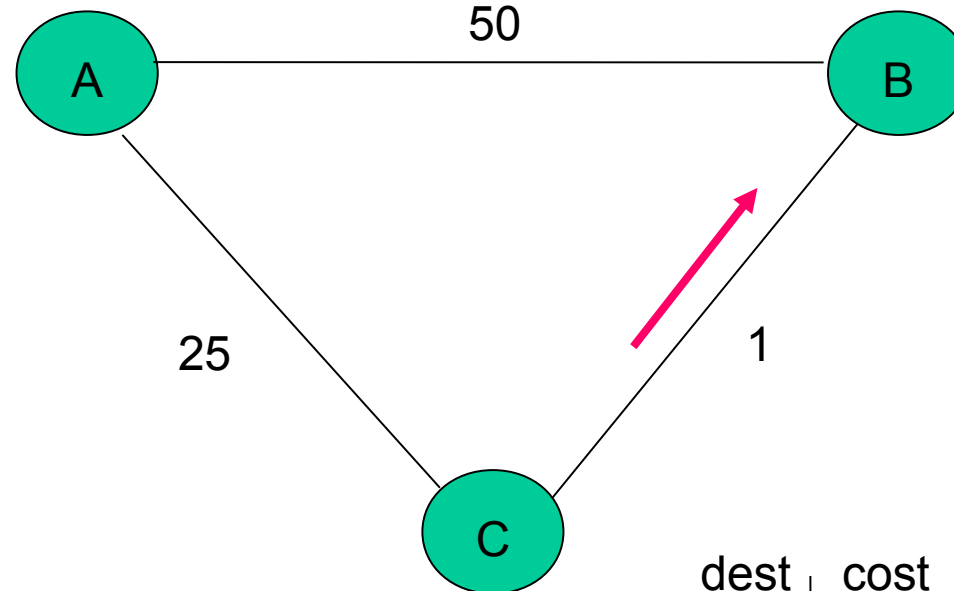


dest	cost
A	3
C	1

dest	cost
A	4
B	1

# C manda il vettore di distanze a B

dest	cost
B	1
C	2



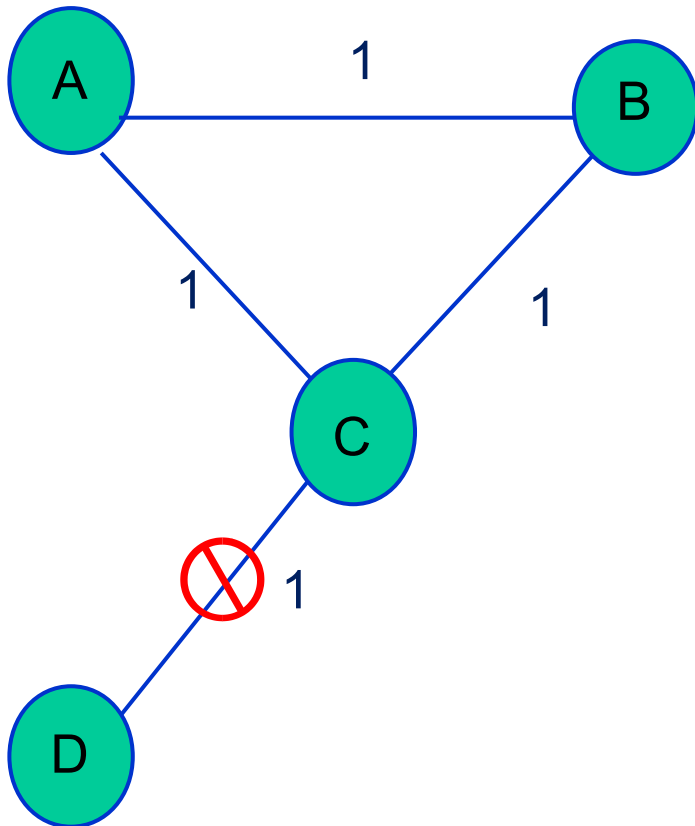
dest	cost
A	5
C	1

dest	cost
A	4
B	1

# *Come si crea l'effetto rimbalzo*

- **La distanza diretta da B verso A cresce molto**
- **Quindi, B sceglie C come prossimo hop per A**
- **Ma..., il percorso implicito da C verso A include B!**
- **Le tabelle di B e C si aggiornano gradualmente, ma si crea un loop che proseguirà fino a quando C considererà il proprio percorso verso A attraverso B minore di 25**
- **Un pacchetto che arrivi a B o a C durante l'esistenza del loop rimbalzerà tra questi due nodi**

# Caso peggiore: non c'è stabilizzazione



- Nel caso in cui il link C-D diventa inutilizzabile, C marca D come irraggiungibile e lo elimina dagli aggiornamenti inviati ad A e B
- Si supponga che A riceva per primo l'aggiornamento. Adesso A considera che il cammino minimo verso D sia attraverso B.
- A dichiara D irraggiungibile a B e a C notifica un costo pari a 3
- C vede D raggiungibile attraverso A a costo 4 e lo notifica a B
- B notifica un costo di 5 ad A che notificherà un costo aggiornato di 6 a C
- Rischio: "count-to-infinity"

# Possibili soluzioni

- **Evitare il “count-to-infinity”**
  - Scegliere una soglia (abbastanza bassa) per “rappresentare” l’infinito. Es., massimo numero di hop necessari = 16
- **Split Horizon**
  - Bisogna differenziare i vettori di distanze inviati ai nodi adiacenti: il vettore di B inviato a C non conterrà le destinazioni raggiungibili tramite C
  - Obiettivo: “Se B raggiunge A attraverso C, non ha senso per C cercare di raggiungere A attraverso B”



# *Possibili soluzioni*

- **Split Horizon with poisoned reverse**
  - Se B raggiunge A attraverso C, B avvertirà C che la sua distanza verso A è infinita (anche se in realtà sa di poter instradare i pacchetti tramite C, il costo risulta troppo alto)
- **Non funzionano per cicli che coinvolgono 3 o più nodi**

# *Evitare l'effetto rimbalzo*

- **Per evitare l'effetto rimbalzo (bouncing effect) si devono selezionare percorsi senza cicli**
- **Un modo per farlo:**
  - Ogni aggiornamento del cammino minimo verso un nodo riporta l'intero percorso
  - Se un router vede se stesso nel percorso, scarta il percorso
- **Problema: la quantità di dati trasmessi è proporzionale alla distanza tra i nodi**

# **Modulo 9b: Link state protocol**

# Algoritmi Link State

- **Gli algoritmi Link State (LS) sono centralizzati**
- **Prevedono che la topologia di rete e i costi di ogni link siano noti (disponibili in input all'algoritmo):**
  - Ogni nodo calcola lo stato dei link ad esso connessi
  - Ciascun nodo periodicamente trasmette identità e costi dei link connessi (link state broadcast)

**(Quindi tutti i nodi hanno una visione identica e completa della rete)**

  - Ciascun nodo calcola i cammini di costo minimo verso tutti gli altri nodi della rete mediante l'Algoritmo di Dijkstra

# ***Pacchetti con informazioni sullo stato dei link (LSP)***

**Periodicamente vengono inviati in broadcast, su tutti i link del nodo, dei pacchetti LSP con le seguenti informazioni:**

- **Node ID**
- **Lista di vicini e costo dei rispettivi link**
- **Informazioni aggiuntive:**
  - Numero di sequenza per accorgersi di errori in caso di delivery out-of-order delle informazioni
  - Time To Live (TTL) per evitare di usare informazioni vecchie e quindi non affidabili

# *Propagazione dei pacchetti LSP*

## **Inoltro con algoritmo di flooding (inondazione)**

**Quando il nodo  $i$  riceve un LSP dal nodo  $j$ :**

- **Se il pacchetto LSP più recente proveniente da  $j$ , viene salvato nel database e una copia viene inoltrata su tutti i link connessi al nodo  $i$  (ad eccezione di quello da cui l'LSP è stato ricevuto)**
- **Altrimenti il pacchetto LSP viene scartato**

# *“Forward search algorithm” di Dijkstra*

- **Algoritmo iterativo: alla k-esima iterazione, il nodo i conosce il cammino di costo minore verso k nodi destinazione**
- **Si definiscono:**
  - $c(i,j)$  costo del link tra nodo i e nodo j
  - $D(v)$  costo minimo del cammino verso il nodo v (minimo per la iterazione corrente)
  - $p(v)$  immediato predecessore di v lungo il cammino a costo minimo verso v
  - N gruppo nodi il cui cammino di costo minore è noto definitivamente

# Algoritmo di Dijkstra - inizializzazione

- **Passo di inizializzazione seguito da un ciclo eseguito una volta per ogni nodo del grafo**
- **Al termine saranno stati calcolati i cammini minimi dal nodo  $u$  verso tutti gli altri nodi**

## Inizializzazione

**$N = \{u\}$**

**Per tutti i nodi  $v$**

**se  $v$  è adiacente a  $u$**

$$\mathbf{D(v) = c(u,v)}$$

**altrimenti  $D(v) = \infty$**



# Algoritmo di Dijkstra - ciclo

## Ciclo

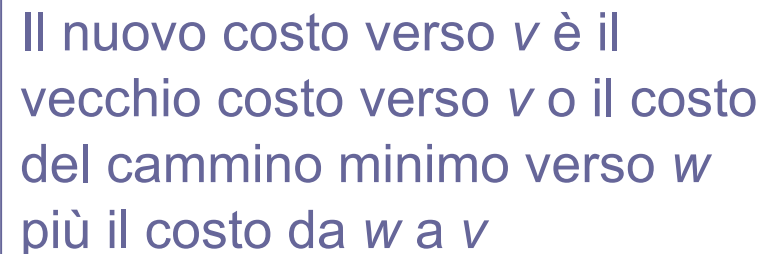
Calcola per tutti i nodi adiacenti  $i$  non in  $N$  il costo  $D(i)$

Aggiungi a  $N$  il nodo  $w$  con il minimo costo  $D(w)$

Aggiorna  $D(v)$  per ciascun nodo  $v$  adiacente a  $w$  e non in  $N$ :

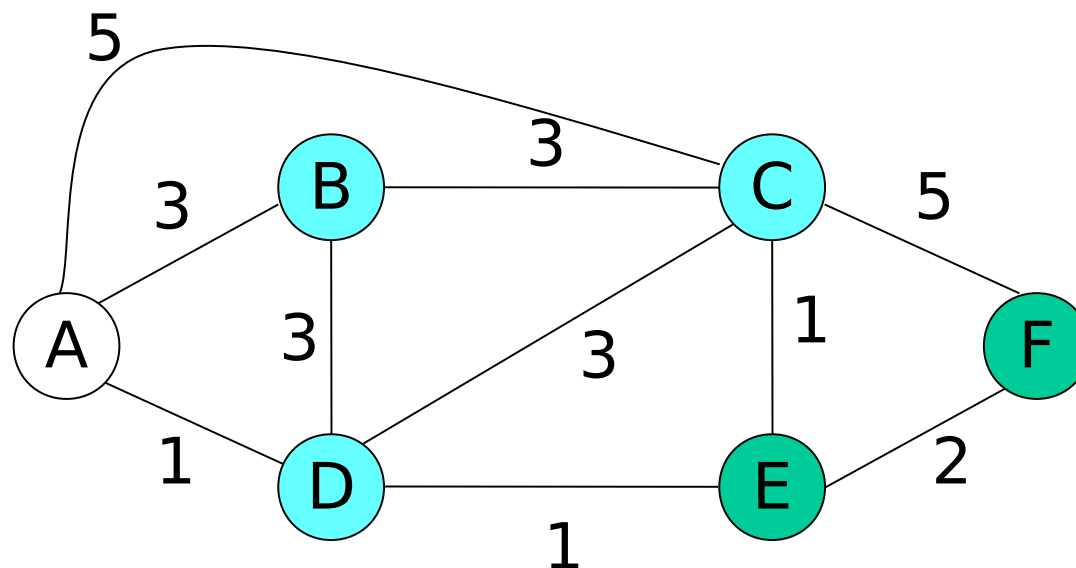
$$D(v) = \min\{ D(v), D(w) + c(w,v) \}$$

Until tutti i nodi del grafo sono nell'insieme  $N$



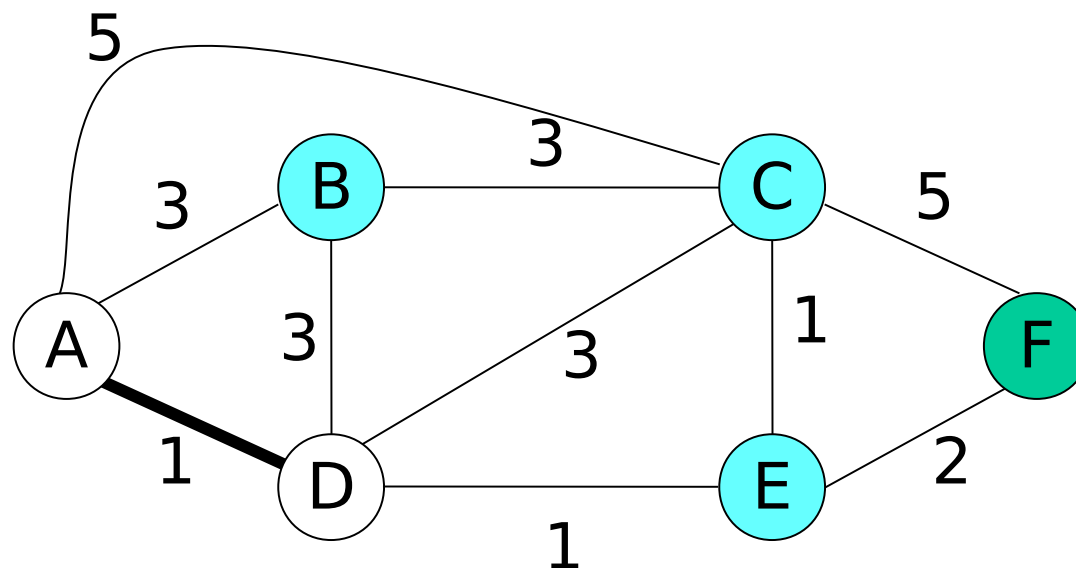
Il nuovo costo verso  $v$  è il vecchio costo verso  $v$  o il costo del cammino minimo verso  $w$  più il costo da  $w$  a  $v$

# Esempio (step 1)



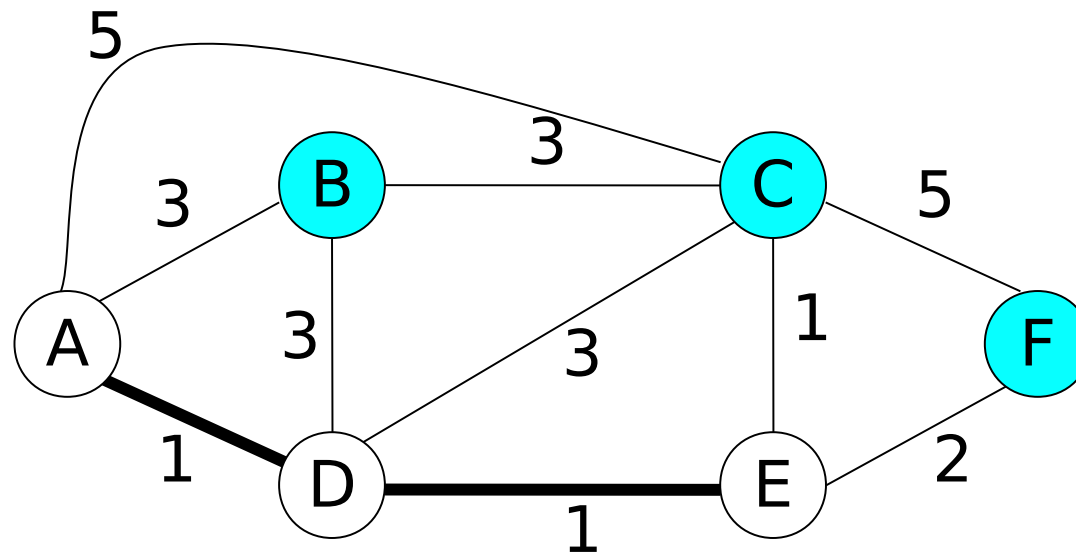
Passo	N	D(B), p(B)	D(C), p(C)	D(D), p(D)	D(E),p(E)	D(F), p(F)
1	A	3, A	5, A	1, A	$\infty$	$\infty$

# Esempio (step 2)



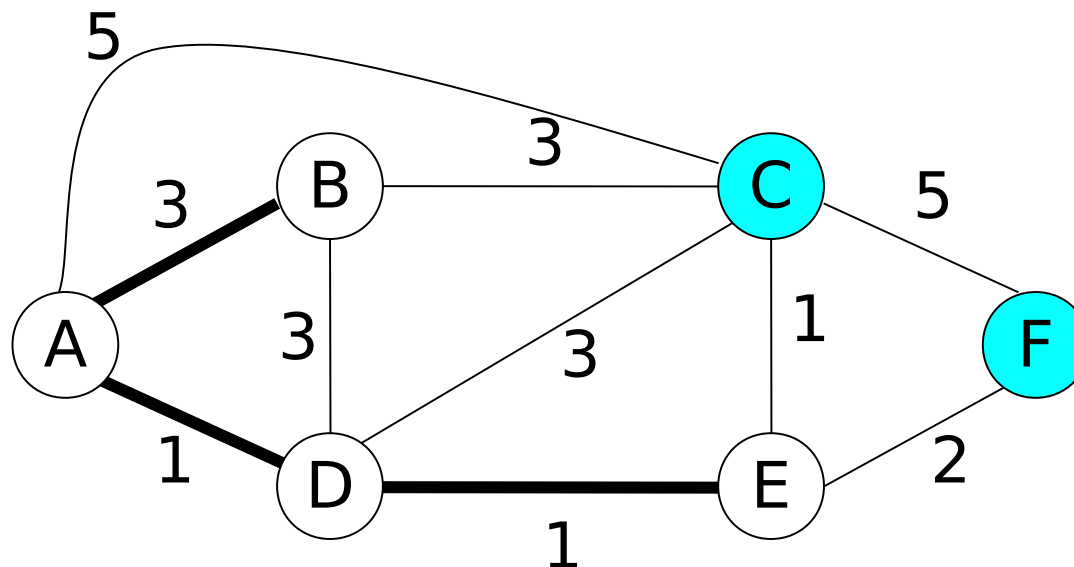
Passo	N	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
1	A	3,A	5,A	1,A	$\infty$	$\infty$
2	AD	3,A	4,D		2,D	$\infty$

# Esempio (step 3)



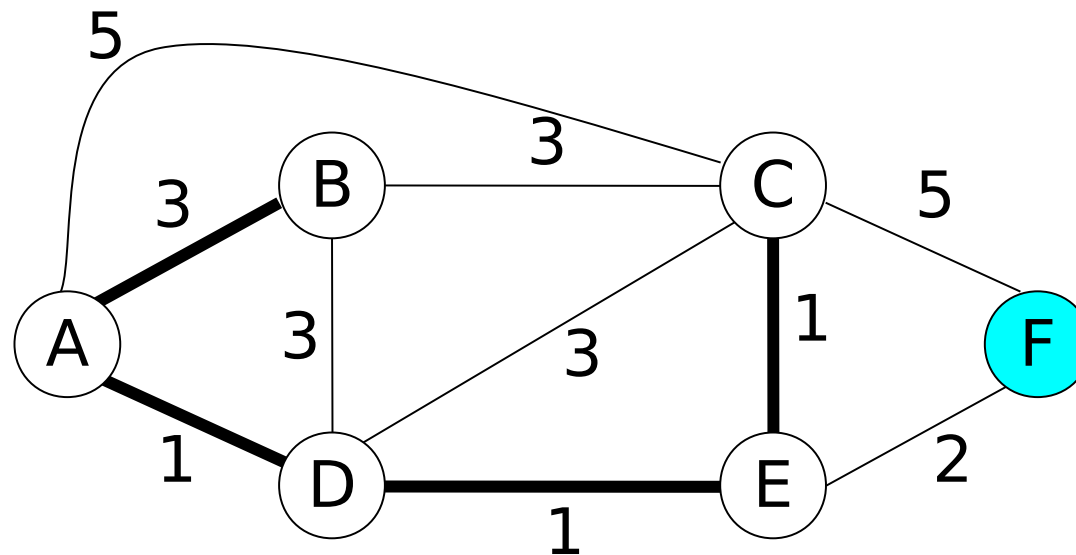
Passo	N	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
1	A	3,A	5,A	1,A	$\infty$	$\infty$
2	AD	3,A	4,D		2,D	
3	ADE	3,A	3,E			4,E

# Esempio (step 4)



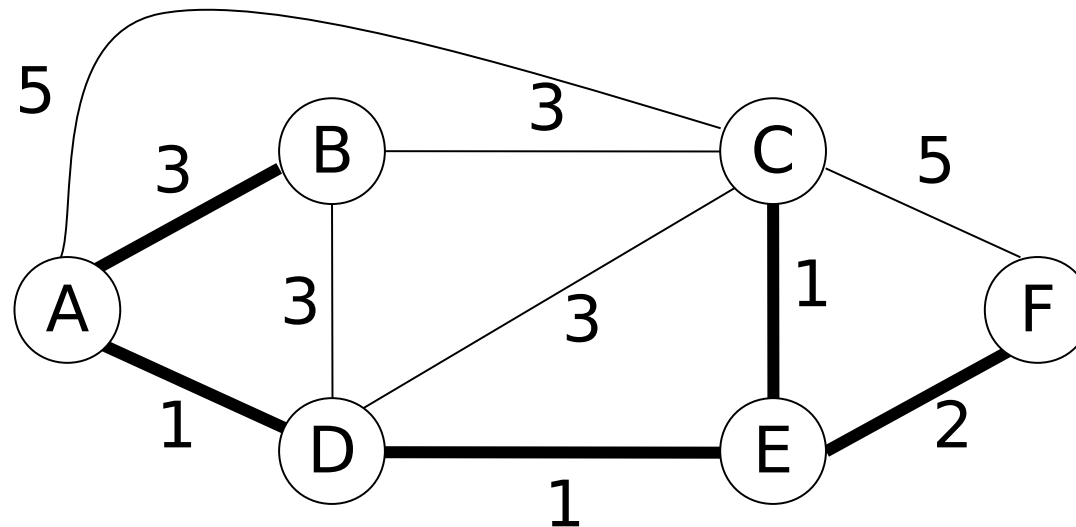
Passo	N	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
1	A	3,A	5,A	1,A	$\infty$	$\infty$
2	AD	3,A	4,D		2,D	
3	ADE	3,A	3,E			4,E
4	ADEB		3,E			4,E

# Esempio (step 5)



Passo	N	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
1	A	3,A	5,A	1,A	$\infty$	$\infty$
2	AD	3,A	4,D		2,D	
3	ADE	3,A	3,E			4,E
4	ADEB		3,E			4,E
5	ADEBC					4,E

# Esempio (step 6)



Passo	N	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
1	A	3,A	5,A	1,A	$\infty$	$\infty$
2	AD	3,A	4,D		2,D	
3	ADE	3,A	3,E			4,E
4	ADEB		3,E			4,E
5	ADEBC					4,E
6	ADEBCF					

**Modulo 9c:  
Distance Vector  
vs. Link State**



# *Distance Vector vs. Link State*

- **DV: tutto quello che si sa è propagato solo ai vicini**
- **LS: le informazioni sui vicini sono passate a tutti**
- **Dimensione dei messaggi**
  - LS: piccola
  - DV: potenzialmente grande
- **Numero di messaggi**
  - LS: molto grande, di tipo  $O(n)$ , dove  $n$  sono i nodi del grafo
  - DV: piccolo in quanto comunicazioni solo ai vicini

# *Distance Vector vs. Link State*

- **Velocità di convergenza**
  - LS: veloce
  - DV: veloce se si usano aggiornamenti periodici abbastanza frequenti (però, troppo frequenti sono a rischio di instabilità)
- **Requisito di memorizzazione**
  - LS: molto alto → si mantiene l'intera topologia del grafo
  - DV: basso → si mantiene solo lo stato dei vicini

# *Distance Vector vs. Link State*

- **Robustezza**

- LS: calcolo dei percorsi effettuato in maniera indipendente da ogni nodo

- **protezione contro guasti ai router**

- DV: calcolo dei percorsi basato sui calcoli degli altri router

- **il calcolo sbagliato di un router può essere propagato a gran parte della rete**

# Conclusione

- **Non c'è un chiaro vincitore tra i due algoritmi:**
  - Distance vector (distribuito) ha dei vantaggi
  - Link state (centralizzato) ha altri vantaggi
- **Gli algoritmi di tipo Link state (centralizzati) tendono ad essere utilizzati all'interno degli AS**
- **Gli algoritmi di tipo Distance vector (distribuiti) sono utilizzati per il routing tra AS**