

ALGORITMI DI ROUTING

ROUTING

- Instradamento

Permette di mettere in comunicazione una sorgente con una destinazione - individuando i percorsi più tra tutte le alternative

Effettuata da un Router

ATTIVITA' DI UN ROUTER

Due step:

1. Ricerca destinazione: interpretare l'indirizzo a cui si deve inoltrare il pacchetto (e quindi individuare la porta di uscita)
2. Inoltro (forwarding) → trasmissione effettiva dei bit nella uscita selezionata

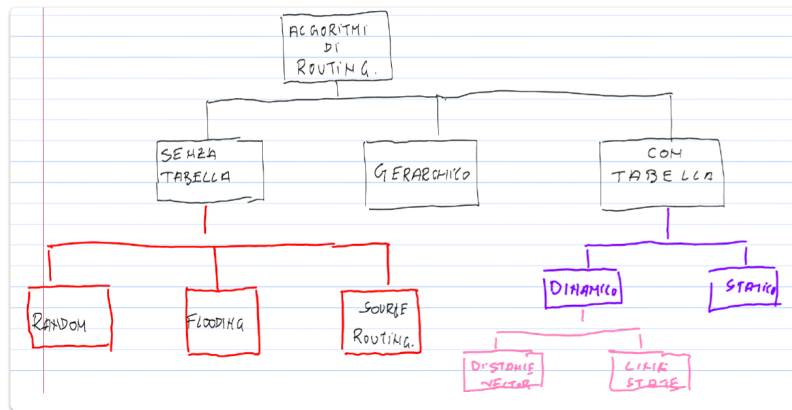
ALGORITMI DI ROUTING

Si cercano metodi/algoritmi che siano:

- semplici da realizzare
- robusti (affidabili)
- stabili
- ottimi (secondo alcuni criteri)

Le tecniche di routing si possono dividere nel seguente modo:

- Implementazione caratteristica
 - Tipologia algoritmo
 - Ulteriori tipologie



- **Senza Tabella** → senza interrogazione del database locale
 - Random
 - Flooding
 - Source Routing
- **Gerarchico** (cfr. rete telefonica)
- **Con Tabella** → il router ha un database dove sono contenute le info utili per il routing
 - Dinamico
 - Distance Vector
 - Link State
 - Statico (stesse organizzazioni)

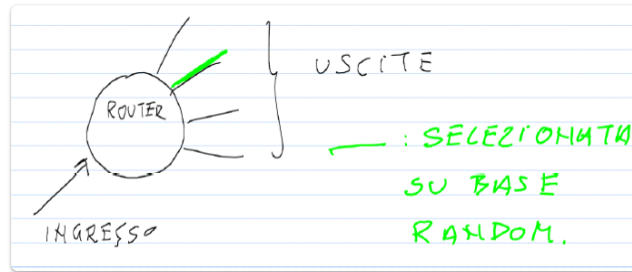
Sistema:

- **Centralizzato**: esiste un dispositivo (anche cloud) che sceglie quale algoritmo usare e collabora con i vari dispositivi
- **Distribuito**: più elementi che gestiscono il routing che cooperano per definire le strategie di routing migliori
- **Isolata**: un router in maniera individuale decide cosa fare (decisioni da prendere tipo algoritmo di routing)

SENZA TABELLA

ALGORITMO RANDOM

Si seleziona su base random su quale porta disponibile nel router andare a ripetere il messaggio ricevuto (escludendo quella da cui il messaggio è arrivato)



- E' robusto perché se si guasta qualche link basta toglierlo dalle possibili scelte, ma l'algoritmo funziona lo stesso 😊
- Non è ottimo perché la scelta è quasi randomica quindi non sarà facile statisticamente trovare quella ottima 😞
- Semplice da realizzare 😊
- Senza tabella: l'azione viene eseguita senza passare dalla consultazione di un database locale (quindi anche veloce) 😊

FLOODING

Si ripete un messaggio su tutte le porte di uscita (escludendo quella da cui arriva)

- Semplice (il più semplice di tutti) 😊
- Rischio di inondazione → allora si imposta un TTL per limitare (cfr. appunti precedenti) 😞
- Sicuro 😊
 - Utilizzato per questo dove si vuol privilegiare la sicurezza che un messaggio arrivi a destinazione rispetto che la velocità/prestazione
 - Ad esempio nelle reti militari (poco estese e necessariamente sicure)

SOURCE ROUTING

Metodo che può essere implementato con due modalità:

- Centralizzata (*path server*): ogni nodo che ha bisogno di inviare un messaggio verso una certa destinazione pone la richiesta al "cloud" (*sink*/apparato centrale). Quest'ultimo ha già costruito tutte le possibili rotte che può gestire e trasferisce al nodo le migliori informazioni per gestire il routing
- Isolata (*path discovery*): direttamente il nodo interessato capisce qual è la rotta da seguire - questo gli è permesso effettuando prima il *discovery* (scoperta) mandando un *pacchetto esploratore* che viene inoltrato nella rete con modalità *floating* con TTL = 1 (massimo una ripetizione) → si scopre il path migliore
 - Viaggiano molte copie del pacchetto esploratore sulla rete: una di queste arriverà alla destinazione cercata
 - Ogni volta che il pacchetto visita un nodo viene scritto sull'*header* il nome del pacchetto che ha visitato
 - Quando il pacchetto arriva alla destinazione c'è la "cronologia dei nodi", quindi basterà ripercorrerla in maniera opposta per scoprire ufficialmente anche dal punto di vista del nodo sorgente il percorso migliore
 - Si considera come rotta migliore quella che arriva prima

ALGORITMI CON TABELLA

Un nodo quando riceve un pacchetto se lo salva/copia nel proprio database locale, dopo aver controllato che in esso non fosse già presente (abbinamenti indirizzo sorgente indirizzo destinazione)

- Più efficiente 😊 ✅
- Più impegnativo (quindi anche più lento) 😞

I due algoritmi che mostreremo sono molto simili tra loro:

- Entrambi arrivano (in modo diverso) alla soluzione ottima
- La differenza principale riguarda la modalità attuativa di realizzare la cooperazione tra i nodi

DISTANCE VECTOR

- Ogni nodo si interfaccia (coopera) con i nodi direttamente connessi (detti "nodi vicini") su base *link-to-link*.

- Ogni nodo trasferisce la propria tabella di routing (condividendola) con i propri vicini

Nella tabella di routing c'è un campo *next-hop*: ovvero il router del passaggio successivo a cui inviare il pacchetto (affinché poi arrivi alla destinazione finale)

Non c'è scritto tutto il percorso, c'è solo il router vicino
(nota: se non ci sono vicini si sceglie un router di default)

L'algoritmo può essere enunciato in questo modo:

Trovare percorsi a costo minimo partendo da un nodo sorgente e selezionando i successivi progressivamente

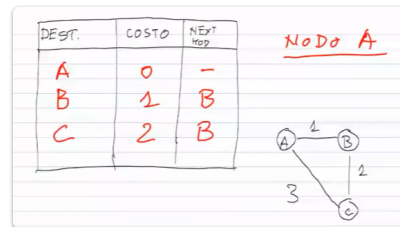
COSTO

Valore di riferimento assegnato a un collegamento con l'obiettivo di costruire un percorso sorgente-destinazione che abbia tale valore minore (costo minore)

- Può essere legato al tempo, al numero totale di nodi, all'affidabilità etc...
- L'importante è stabilirne uno per individuare l'alternativa migliore in un certo contesto

Ad esempio:

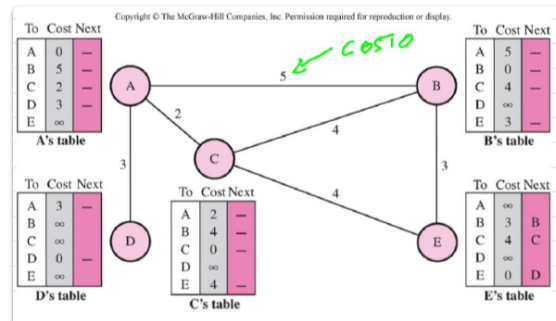
- Per fare $A \rightarrow B$ conviene il collegamento diretto
- Per fare $A \rightarrow C$ non conviene il collegamento diretto ma conviene fare $A \rightarrow B \rightarrow C$



ESECUZIONE

Vediamo l'esecuzione dell'algoritmo con un esempio

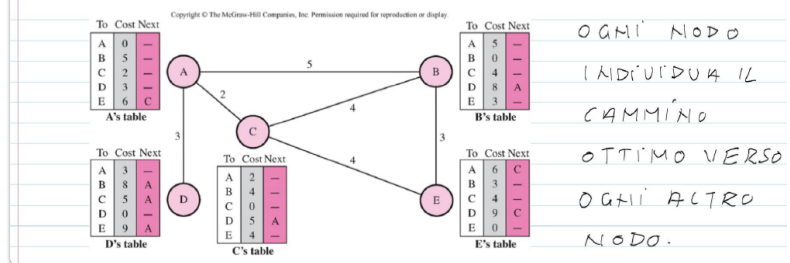
- Sono noti i costi dei links



Passaggi:

- Si sceglie un nodo di partenza, ad esempio A
- Si prende il nodo connesso al nodo di partenza con costo minore, e lo si considera parte di A. In altre parole, in questo caso: $A \leftarrow C$
 - Si ripete lo stesso per ogni nodo, e si condividono le tabelle
- Quando C condivide la sua tabella con A, quest'ultimo acquisisce una visione che prima non aveva (più profonda). In questo modo comincia a verificare se passando da C riesce a migliorare alcuni path verso altri nodi (in termini di costo)
 - In questo caso ad esempio A scopre che può raggiungere E con un costo totale di 6, che probabilmente sarà un costo minimo (ancora però deve verificare tutti i percorsi per la conferma - però intanto segna sulla tabella)
- Si continuano a scambiare tabelle finché queste non subiscono più cambiamenti, ovvero si arriva a una *soluzione stabile*

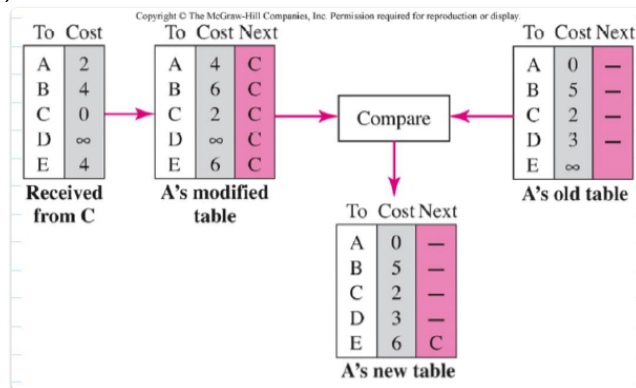
RISULTATO FINALE



ESEMPIO DI AGGIORNAMENTO DI UNA TABELLA

Vediamo la procedura effettiva di scambio (condivisione) delle tabelle:

- Sappiamo che ogni *tot* i nodi si scambiano le tabelle
- Nel seguente caso, C manda ad A l'aggiornamento della propria tabella
- Ora, A deve aggiornare questi valori sommando a tutti costi di raggiungimento relativi $C \rightarrow ?$ (ovvero a tutte le righe della tabella) i costi di $A \rightarrow C$.
 - In questo caso si somma 2 essendo $A \rightarrow C = 2$
- Viene quindi comparata la nuova tabella di A con quella vecchia e se ne crea una "ibrida" composta dai valori migliori relativi a ciascuna riga
 - In questo caso si ha un miglioramento solo nella ultima e quinta riga, che permette ad A di raggiungere E (che prima era irraggiungibile, ovvero con costo ∞) con un costo 6

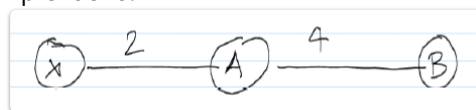


Proseguendo con questo su tutti i nodi si ottiene un *albero* (dei costi minimi)

⚠ CRITICITA' ALGORITMO ⚠

L'algoritmo ha un **problema di instabilità**, alternativamente noto come *problema del conteggio all'infinito*.

Prendiamo un esempio (molto critico) per comprenderlo:



- 3 soli nodi (x, A, B) connessi in cascata
- Supponiamo che per qualche motivo si interrompa il collegamento tra $x \longleftrightarrow A$, ovvero

$$x \not\longleftrightarrow A$$

- Pertanto, il collegamento assume costo infinito, cioè

$$x \overset{\infty}{\longleftrightarrow} A$$

- Supponiamo che questo evento accada **dopo** che B ha condiviso con A la propria tabella di instradamento che prevede raggiungibilità di x attraverso A con costo totale di 6
- La situazione ora è la seguente:
 - A ha compreso che la distanza con x è ∞
 - B non lo sa: non è stato informato dell'evento suddetto, quindi ancora non ha inviato ad A la propria tabella aggiornata
- Allora A consulta nuovamente la tabella e scopre che in realtà per quanto c'è scritto può raggiungere x passando da B con costo 10. Quindi aggiunge la tabella.
- A sua volta B aggiorna la propria tabella sapendo che può raggiungere x soltanto passando da A, quindi a sua volta aggiorna il costo e lo porta a 14

- Il procedimento continua: B comunica ad A che può raggiungere x costo 14. Quindi A aggiorna il costo (dato che il diretto $A \rightarrow x$ costa ∞) e lo pone a 18
- A un certo punto entrambi raggiungono un valore di cammino per x molto grande, e quindi comprendono che tale nodo non è raggiungibile \rightarrow *spreco di tempo ed energie*

✓ Soluzioni al problema

Sono state trovate quindi delle soluzioni a questo problema. In particolare 3 metodi:

① Infinito finito

Si stabilisce un valore massimo per il costo del collegamento, superato il quale il collegamento converge alla situazione di "improponibile/impossibile", ovvero equivale trattare costo ∞

😊: riduce il disservizio, 😡 non lo elimina del tutto (specie se il numero che impostiamo è alto)

✍ Split Horizon

Orizzonte diviso: permette di evitare di mandare aggiornamenti in quei percorsi che si sa essere attuati passando dal nodo verso il quale si sta mandando l'aggiornamento.

Nell'esempio proposto, B sa che raggiunge x attraverso A quindi non manda ad A l'aggiornamento del suo percorso verso x (al limite manda gli altri, come $B \rightarrow A$)

😡 A livello di come sono implementati i dispositivi tuttavia, si sa che se un dispositivo (nel nostro caso A) non riceve notizie (aggiornamenti) da un altro (ad esempio B) per un certo lasso di tempo, allora il primo può ritenere il secondo come "inattivo", quindi escluderlo dai suoi "vicini". Per questo è stato ideato il terzo metodo...

“ Metodo Ibrido: Split Horizon with Poison Reverse

Si mandano aggiornamenti periodici per *tutti* i cammini, anche quelli esposti nel secondo metodo, assegnandoli un valore *alto*, così che poi non saranno mai effettivamente scelti.

In questo modo anche i dispositivi "critici" continuano a essere considerati attivi

LINK STATE (ALGORITMO)

- Converge all'ottimo con una maniera diversa: i nodi in questo caso si "aiutano a vicenda", ma ognuno **non** manda in rete la tabella completa, ma bensì unicamente *l'informazione riguardo i percorsi che può gestire con i costi relativi* (cioè non manda la tabella in blocco ma solo i singoli componenti in modo individuale e solo quando il nodo percepisce cambiamenti rispetto a situazioni precedenti)
 - Quindi non si mandano tutti, e quei singoli si mandano solo in casi di cambiamento (non periodicamente "a dritto")
 - Risolve il problema del conteggio all'infinito 😊

Passi:

- Si accende un nodo
- Esso scopre i propri vicini e manda l'informazione a ciascuno riguardo il percorso con il relativo costo
- I vicini lo ricevono, aggiornano la tabella ed eventualmente se ci sono stati effettivamente degli aggiornamenti li comunicano a tutti gli altri

Mentre prima l'informazione veniva trasferita solo fino ai nodi direttamente connessi, in questo caso l'informazione di un cammino che cambia costo arriva a tutti i nodi della rete. È quindi più pervasiva, non è cioè più limitata a un hop

Quindi nel caso precedente l'informazione arrivava solo ai vicini (hop). In questo caso invece arriva (influenza) a tutti

Possiamo esporre l'algoritmo come segue:

DISTANCE VECTOR E LINK STATE A CONFRONTO

I due algoritmi utilizzano approcci diversi al riempimento della tabella di routing:

- Nell'algoritmo distance vector ogni nodo comunica solo con i nodi a lui connessi per scambiarsi informazioni inerenti al costo del percorso tra di loro mentre nell'algoritmo link state ogni nodo comunica con tutti i nodi della rete per avere una visione globale dello stato.

In caso di cambio di costo tra due nodi con l'algoritmo link state tutta la rete viene inondata con un messaggio che informa di tale variazione, se poi ciò implica un cambio nella tabella di routing dipende da nodo a nodo. Invece nell'algoritmo distance vector, il cambio del costo verrà notificato a tutti i nodi della rete solo se esso introduce un cambio del percorso a minor costo per uno dei nodi collegati a quel link.

- L'algoritmo link state è più robusto rispetto al distance vector, poiché ogni nodo, in modo indipendente rispetto agli altri, calcola la propria tabella di routing. Questo vuol dire che se un nodo comunica una metrica sbagliata, ciò non influisce su tutta la rete. Invece nell'algoritmo distance vector, ogni ri-calcolo (sbagliato) dei costi viene poi passata ai nodi adiacenti dunque l'errore può propagarsi in tutta la rete.

vedi esercizio sulle slide

✍ Vediamo ora una serie di algoritmi che derivano dai precedenti che abbiamo visto

RIP (ALGORITMO - BASATO SU DISTANCE VECTOR)

- Utilizza come (metrica di) costo dei collegamenti il numero di *hop* nel percorso sorgente-destinazione

CARATTERISTICHE

- Essendo distance vector, può soffrire di conteggio all'infinito 😞
 - La soluzione proposta è quella dell'infinito finito 😊
 - Il numero fissato a tal proposito è 15
 - Buono perché ci si ferma presto, ma se ci fossero collegamenti più distanti di 15 *hop*?
 - Quindi, l'efficienza dell'algoritmo è dipendente al numero di dispositivi 😞
- Per l'aggiornamento delle tabelle, si prevede uno scambio con un rate di 30 secondi
 - La procedura (che manda un datagramma IP) prende il nome di RIP *advertisement*
 - Può essere inviata a massimo 25 destinazioni 😞
 - Se un nodo non manda niente per 180s viene considerato inattivo e quindi escluso dalla rete 😞
 - Esistono anche messaggi specifici per aggiornare l'algoritmo
- I messaggi contenenti l'istruzioni di routing (aggiornamenti e non solo), che sono ovviamente "molto delicati", vengono inviati tuttavia con protocollo UDP (e non TCP) 😞
 - Oltre al tipo di protocollo utilizzato, ci chiediamo soprattutto: dato che il router lavora a livello 3, perché dobbiamo utilizzare un protocollo di livello superiore (4) per il trasferimento invece che il protocollo IP?
 - Questo perché in realtà *questi algoritmi sono applicazioni* che risiedono nel relativo piano applicazioni appunto, e quindi comandano lo stato TCP che è a livello inferiore.

OSPF - BASATO SU LINK STATE

Open Shortest Path First (scegliere prima il percorso più corto)

- Usa l'algoritmo di *Dijkstra*
- Il costo non è un valore univoco - è specifico della relativa rete di riferimento

CARATTERISTICHE


- Gli aggiornamenti avvengono in modalità *flooding*: i nodi mandano gli aggiornamenti dei cammini ai vicini → può succedere che un nodo possa ricevere messaggi di aggiornamento da uno stesso nodo generati in momenti diversi. Il nodo deve tener conto solo dell'ultimo di essi (che è quello più aggiornato).

- Per questo, in ogni messaggio c'è un *time-stamp* che viene associato.
 - Ogni nodo che riceve un aggiornamento quindi controlla se nel proprio DB ha una copia più recente dello stesso aggiornamento. In caso affermativo, esclude il messaggio e lo rimanda indietro, con allegato il *time-stamp* dell'aggiornamento più recente che possiede. Altrimenti si considera il messaggio e si aggiorna il DB
 - Se non avvengono aggiornamenti, ogni 30 minuti il router procede a diffondere un messaggio di default con lo stato dei dispositivi
- Quando un nodo viene acceso per la prima volta si presenta ai propri vicini con un messaggio di HELLO
 - Prevede di autenticare la fonte che fornisce le informazioni di aggiornamento → livello di *sicurezza alto* dell'algoritmi
 - Permette l'utilizzo dei percorsi che hanno lo stesso costo in modo indistinto
 - Permette di utilizzare modalità di *instradamento diverse* da quelle previste da una certa sorgente-destinazione
 - In dettaglio, tra le altre:
 - Prevede il *multicast*: invio di un pacchetto da una sorgente a più destinazioni;
 - Prevede il *broadcast*: invio di un pacchetti da una sorgente a tutti gli altri nodi della rete
 - ogni nodo lavora in modo indipendente costruendosi la sua tabella (in questo modo se qualcuno sbaglia, l'errore non si propaga nella rete)

[algoritmi gerarchici: non li facciamo]

NUOVA (MODERNA) TECNICA DI INSRADAMENTO: MPLS

Multi Protocol Label Switch

- Tecnica di instradamento (switching) multiprotocolli basata su *label* (etichette)
- Si impegna a migliorare le tecniche preesistenti:
 - : alta velocità di trasferimento di switching/router (un po' come introdurre il telepass in autostrada: si ha una velocità di percorrenza ai caselli più veloce)
- Ha migliorato le precedenti reti x25 e ATM, con la differenza che stavolta si lavora a livello di *etichetta* invece che di indirizzo:
 - Più indirizzi possono essere associati a una stessa etichetta: questo alleggerisce le operazioni di routing, perché scoprendo/elaborando un' unica etichetta si gestiscono più indirizzi rendendo quindi più veloci le operazioni
 - (associa più flussi informativi a una stessa etichetta)
- Permette di *poter differenziare le rotte in relazione ai requisiti di servizio a cui il flusso informativo è associato* - questo permette di associare a etichette diverse anche flussi destinati a una stessa destinazione
 - In questo modo **etichette diverse ↔ gestioni diverse inoltro messaggio in rete**
 - Si viola da un certo punto di vista il senso d'instradamento classico che abbiamo, in cui si associa in maniera *biunivoca* la sorgente dalla destinazione (questo non avviene più)
 - Gli algoritmi precedenti associano un unico percorso a una destinazione, trovando un percorso ottimo
 - Stavolta una stessa destinazione può essere raggiunta da una sorgente *con percorsi diversi* a seconda del tipo di traffico che viene mandato
 - Apre le discussioni alla *Traffic Engineering*
- E' un *middleware*, livello intermedio tra il livello collegamento e il livello IP
- Modalità a commutazione di circuito (CC)

COMPOSIZIONE

Il campo MPLS comprende 4 byte, ed è così composto:

LABEL	EXP	S	TTL
20 bit	3 bit	1 bit	8 bit

- Label: vedi sopra
- EXP: Sperimentale (quasi mai usato)
- S: flag che indica se c'è una o più etichette scritte nel campo *label*
- TTL: Time To Live

Dove si mette questo campo di 32 bit?

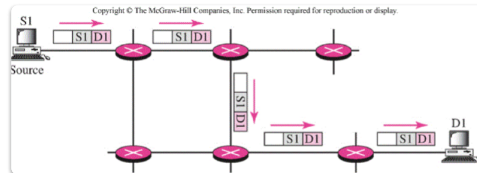
Deve precedere la testa IP, quindi va inserita nel campo di informazione di livello 2

Va letta prima della testata IP per capire se si può ridurre i tempi

ROUTING MULTICAST E BROADCAST

Fin ora abbiamo considerato solo casi di *routing unicast*, ovvero collegamenti tra una sorgente e una destinazione, del tipo

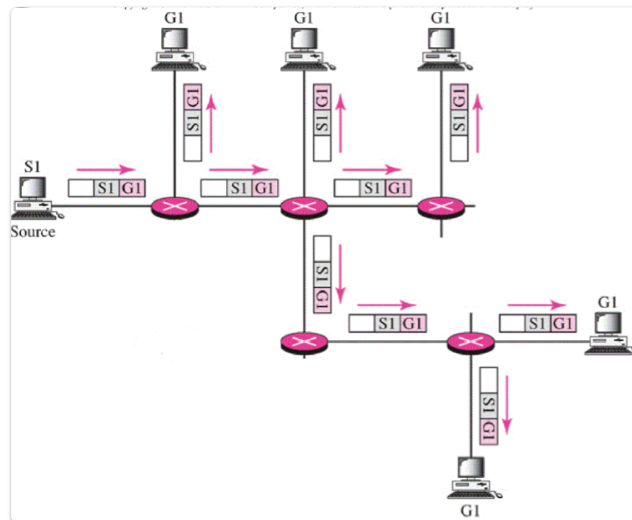
1 : 1



Studiamo ora le modalità *multicast* (uno a molti) e *broadcast* (uno a tutti):

$1 : n$, $1 : N$ $1 < n < N \in \mathbb{N}$

Si è pensato a queste nuove modalità per rispondere a servizi specifici soprattutto a livello applicazione - ad esempio e-learning (multicast - un host che trasmette a tanti)



- si nota come in questa modalità *ogni router effettua tante copie del pacchetto quante sono le porte*, e le inoltra in tutte queste

L'alternativa si chiama *multi-unicast*: nelle ipotesi di conoscere quante sono le destinazioni da raggiungere, il nodo sorgente inoltra altrettante copie del pacchetto, e i vari router ne trattengono una e smistano le rimanenti agli altri

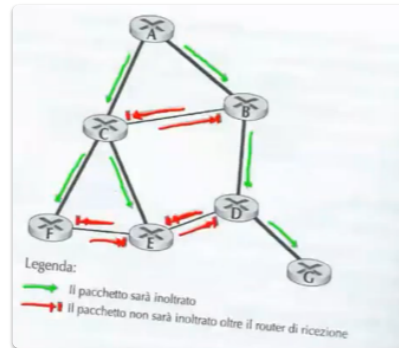
- 😊: piuttosto sicuro
- 😞: molto pesante da implementare - quasi mai usato

TECNICHE DI ROUTING BROADCAST

Ci chiediamo in che modo si può mandare un pacchetto a tutti gli utenti della rete:

1. *Flooding Broadcast* (già visto): ogni nodo che riceve un pacchetto broadcast lo inoltra su tutte le porte tranne quella da cui è arrivato
 - 😊: semplice
 - 😞: possibile inondazione 😊 si risolve o comunque si limita impostando $TTL = 1$
 - 😞: inefficiente e talvolta pesante
2. *Reverse Path Forwarding*: si ripete a molti (multicast) & *Reverse Path Broadcast*** (si ripete a tutti)

- Percorso inverso: ogni nodo, che ha la propria tabella di routing, quando riceve un pacchetto controlla se la porta da cui è arrivato è la porta che corrisponde al cammino (inverso) più breve che lo connette con quella sorgente
 - In caso affermativo, processa il pacchetto
 - Altrimenti no, non fa niente - vuol dire che esiste un percorso migliore



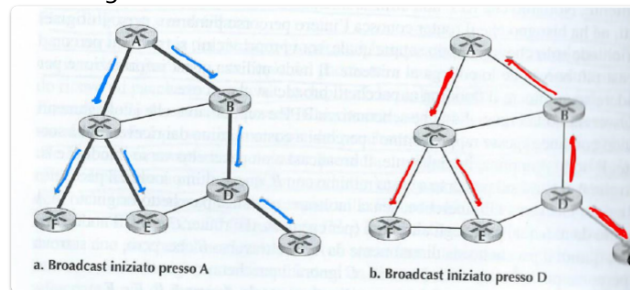
- Esempio: a C gli arriva un pacchetto da A ; poco dopo gli arriva lo stesso che è passato da B , ovvero: $A \rightarrow B \rightarrow C$. Il nodo C processa quindi solo quello che gli è arrivato da A

MODALITA'

Esistono due modalità per attuare tale procedura:

- si elegge un coordinatore del gruppo multicast (o broadcast), ad esempio il nodo A . In questo modo, quando un $nodo \neq A$ vuole mandare un pacchetto a tutti non lo manda direttamente, ma lo manda ad A attraverso il collegamento migliore (che risiede nella propria tabella).
 - A questo punto A lo inoltra con modalità Reverse Path Forwarding (Broadcast)

In alternativa, il nodo sorgente (in questo caso A) può costruire innanzitutto il proprio albero di collegamenti multicast (o broadcast), in questo modo quando arriva una richiesta sa già dove smistarla e come



Come avviene l'adesione di un nodo a un albero?

Un nodo invia la propria richiesta di appartenere all'albero (multicast per esempio) all'amministratore (nel nostro caso A) → procedura detta innesto

- La richiesta $nodo \rightarrow A$ deve attraversare tutto l'albero quindi comporta **ritardo**
 - Allora per velocizzare, la richiesta di adesione **si ferma al primo nodo che appartiene al gruppo multicast** - esso è già connesso al coordinatore A , quindi si impegnerà lui a gestire la richiesta
 - Vale lo stesso per la fase di **dissociazione**, e prende il nome di **potatura**