Routing IP

Tecnica

La funzione fondamentale dell'**instradamento** (*routing*) consiste nell'inoltro (*forwarding*) di pacchetti ed avviene generalmente in modalità *store-and-forward* (memorizza ed inoltra). La necessità di ricevere completamente il pacchetto prima di ritrasmetterlo introduce un tempo di latenza pari al tempo di trasmissione.

Le tecniche fondamentali di inoltro, che differiscono per il metodo di analisi del problema instradamento, sono le seguenti:

- *Routing by network address*. L'indirizzo di un sistema, che deve essere univoco sulla rete, è scritto direttamente nel pacchetto. Gli **IS** (*Intermediate System*) usano tale indirizzo come chiave di ricerca nella loro tabella di instradamento e determinano lungo quale cammino il pacchetto debba essere ritrasmesso. Tale tecnica è usata nei ***transparent-bridge*** (livello OSI 2), e in IP. È in generale adottata dai protocolli non connessi.
- *Label swapping*. È generalmente usata nei protocolli connessi e trova applicazioni in **ATM**. Ogni pacchetto è marcato con una *label* che serve come chiave in una tabella di instradamento sull'IS. L'IS, prima di ritrasmettere il pacchetto, sostituisce la *label* con una nuova *label*. Le *label* devono quindi essere univoche solo all'interno di un dato link. Se il protocollo è connesso, le *label* altro non sono che gli identificativi delle connessioni.
- *Source routing*. È una tecnica usata tramite una opzione del protocollo IP (per esempio, dai *bridge Token Ring*). Nel *source routing* la lista degli IS da attraversare, è scritta nel pacchetto dal nodo mittente, che lo chiede ad un IS o lo scopre con meccanismi di "*route location*".

La tecnica presa in esame in questa trattazione sarà la prima, poiché è quella adottata negli schemi di instradamento IP, e quindi integrata nei protocolli e nei *router* IP.

Definizioni

Con la dicitura rete fisica si indica un insieme di calcolatori aventi le interfacce di rete attestate su una stessa sottorete, in cui una particolare tecnologia di trasporto assicura la connessione. Una rete logica è l'insieme delle interfacce, a cui è stato assegnato lo stesso indirizzo di ***'subnet'***, che possono comunicare senza dover passare attraverso un ***router*** (instradatore).Tale condizione viene detta di *routing* implicito. IP assumeva originariamente una corrispondenza biunivoca tra reti fisiche e logiche; realizzazioni più moderne ammettono anche più reti logiche nella stessa rete fisica. Il *routing* tra reti logiche diverse è esplicito ed è gestito dai *router* tramite tabelle di instradamento.

IP adotta i concetti di destinazioni dirette e indirette nella sua logica di *routing*. Un *host* diretto è una stazione collegata direttamente alla rete ed al *router* della rete, mentre un *host* indiretto è un *host* di destinazione situato su una rete diversa da quella dell'*host* di origine; questo significa che il **datagramma** deve essere inviato ad un *router* intermedio prima di essere consegnato all'*host* di destinazione.

Il modo in cui IP gestisce gli indirizzi e decide i percorsi di *routing*, richiede che una macchina esamini solo la parte di indirizzo di rete dedicata all'indirizzo di destinazione, per determinare se l'*host* di destinazione è collegato direttamente o indirettamente alla rete dell'*host* di origine: in altri termini, la macchina verifica la corrispondenza della parte 'rete' dell'indirizzo di destinazione e sceglie se effettuare un *forwarding* diretto o *forwarding* indiretto.

Forwarding diretto: la trasmissione di un datagramma IP tra due *host* connessi su una singola rete logica IP (stesso *netid*): non coinvolge i *router*. Il trasmettitore incapsula il datagramma nel frame fisico e lo invia direttamente all'*host* destinatario.

Forwarding indiretto: i datagrammi passano da un *router* all'altro finché non raggiungono un *router* che può trasmetterli direttamente. I *router* realizzano l'interconnessione tra le diverse reti.

Classificazione

Gli **algoritmi di *routing*** possono essere classificati per tipo:

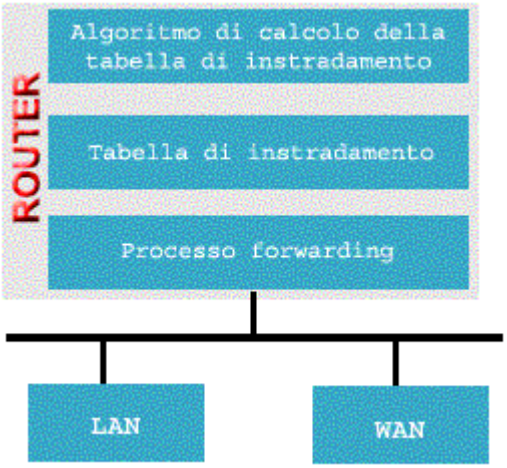
- ***Statici*** o ***dinamici***: negli algoritmi statici, le tabelle di *routing* che vengono memorizzate sono compilate da una persona (amministratore di rete) e i valori di tali tabelle non cambiano per nessun motivo fino a quando l'amministratore di rete non li cambia, mentre negli algoritmi dinamici le tabelle vengono continuamente aggiornate e cambiate a seconda dei cambiamenti della rete (caduta di una rete, inserimento di una rete).
- *Gerarchici*: i *router* gerarchici hanno funzioni diverse da quelli che non lo sono, in quanto vengono suddivisi più nodi in gruppi logici chiamati domini di *routing*, *autonomous system* o aree. Solo alcuni di questi *router* possono interagire con ulteriori *router* di altri domini di *routing*, mentre altri possono interagire con *router* appartenenti allo stesso dominio.
- ***Link-State*** o ***Distance-Vector***: *link-state* (conosciuto anche come *shortest path first*) trasferisce tutte le informazioni di *routing* a tutti i nodi: ogni *router* invia solo la porzione di tabella che descrive lo stato dei suoi link. Gli algoritmi del tipo *distance-vector* inviano tutta o parte della tabella ai soli router vicini. Quindi *link-state* spedisce piccoli aggiornamenti a tutti, *distance-vector* spedisce grossi aggiornamenti ma solo ai *router* vicini: i *link-state* richiedono più risorse *hardware* (CPU e memoria) rispetto ai *distance-vector*, ma sono meno propensi ai *routing loop*.

Tabella di instradamento

Ogni *router* contiene una tabella di instradamento, visto che se un pacchetto viene destinato al *router* questo dev'essere instradato. Ogni riga nella tabella deve contenere almeno i seguenti tre elementi:

- un indirizzo di destinazione: il *router* può avere più di un percorso per la stessa destinazione.
- l'interfaccia su cui inoltrare i pacchetti.
- il costo per raggiungere la destinazione sul percorso, che inizia con l'interfaccia indicata nella riga.

Il costo consentirà all'IS di scegliere tra eventuali percorsi alternativi; l'unità di misura di questo costo dipende dal protocollo utilizzato. Si indicano genericamente con il termine *route* le informazioni presettate su una riga della tabella di *routing*.



Quando il *router* deve inoltrare un pacchetto, scorre la tabella per individuare la riga corrispondente al destinatario del pacchetto stesso. Mediamente il tempo di ricerca (*table lookup*) è pari alla metà del numero di righe. Considerando che tale operazione viene eseguita ogni volta che si deve inoltrare un pacchetto, diventa molto critica la complessità della tabella ai fini delle prestazioni dell'apparato.

Routing

Nel ***routing by network address***, la ricerca non verrà basata sull'intero indirizzo del destinatario, ma su un prefisso, molto spesso di lunghezza variabile. La ricerca dovrà essere eseguita nei confronti di quella riga che specifica il *route* con più lungo prefisso comune all'indirizzo del destinatario (*longest prefix matching*).

Affinché i pacchetti arrivino a destinazione è indispensabile che le tabelle nei vari IS siano coerenti tra di loro, al fine di evitare l'invio di pacchetti in percorsi ciclici (*routing loop*). In tal caso i pacchetti girerebbero a vuoto, consumando inutilmente risorse computazionali e trasmissive dei vari *router*.

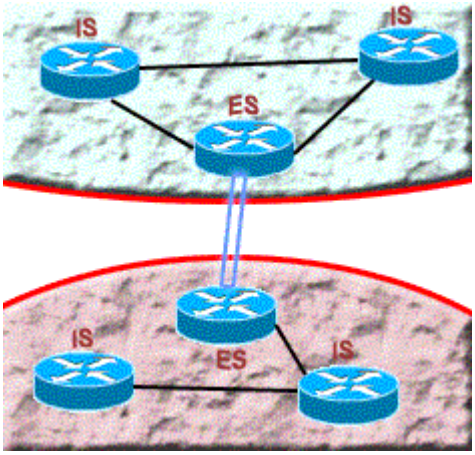
Dal percorso lungo il quale un pacchetto viene inoltrato, dipendono il ritardo che esso subirà, la probabilità che venga scartato a causa di eventuali congestioni del IS e il fatto che esso raggiunga o no la destinazione. Inoltre se la rete contiene maglie, una destinazione potrà essere raggiunta attraverso una o più percorsi alternativi; in presenza di guasti, la scelta di un percorso che eviti nodi o collegamenti non funzionanti consentirà alla rete di continuare a recapitare dati.

Dunque la scelta del percorso, cioè il ***routing***, sarà un fattore chiave per il buon funzionamento della rete e per la sua robustezza (*fault-tolerance*).

Neighbor greetings

Un altro problema è rappresentato dal ***neighbor greetings***, cioè del fatto che gli IS collegati ad una **LAN** devono conoscere gli ES collegati alla stessa LAN e viceversa. Questo è indispensabile per due motivi:

- gli IS devono conoscere gli ES per inserirli nelle tabelle di instradamento e propagare l'informazione della loro raggiungibilità agli altri IS;
- gli ES devono conoscere gli IS presenti sulla LAN per sapere a chi inviare i messaggi non destinati a nodi collegati alla stessa LAN.



La soluzione a quest'ultimo problema deve essere tale da ammettere LAN prive di *router*, LAN con un solo *router* o LAN con più *router*.

Routing statico

Il *routing* statico prevede che i percorsi di inoltro dei pacchetti siano determinati ed eventualmente cambiati dall'amministratore della rete tramite la configurazione degli apparati di *internetworking*. Quindi la tabella di *routing* è sotto la responsabilità dell'amministratore che deve gestire l'inserimento, la modifica e l'eliminazione delle righe.

Principalmente si ha lo svantaggio della mancanza di reattività ai cambiamenti topologici della rete, ed i percorsi non saranno automaticamente adattati alle variazioni dello stato di funzionamento di nodi e collegamenti.

Il ***fixed directory routing*** e il ***flooding***, sono le principali tecniche di *routing* statico.

Routing statico: fixed directory routing

Il *fixed directory routing* prevede che ogni nodo abbia una tabella di instradamento che metta in corrispondenza il nodo da raggiungere con la linea da usare, e che tale tabella sia scritta manualmente dal gestore della rete nel *router* tramite un'operazione di *management*.

Il gestore ha il totale controllo dei flussi di traffico sulla rete, ma è necessario un suo intervento manuale per il reinstradamento di detti flussi in presenza di guasti. Questo approccio è spesso utilizzato in TCP/IP per le parti non magliate della rete e le regole di instradamento specificate su ogni singolo *router* prendono il nome di *route* statiche.

Esiste una variante, detta quasi-statica, che adotta tabelle con più alternative da scegliere secondo un certo ordine di priorità, in funzione dello stato della rete. Questo approccio, che consente di avere cammini alternativi in caso di guasto, è adottato, ad esempio, dalla rete SNA.

Occorre comunque evidenziare che la gestione manuale delle tabelle risulta molto complessa e difficoltosa, soprattutto per reti di grandi dimensioni.

Routing statico: flooding

Il *flooding* è un altro algoritmo non adattativo, in cui ciascun pacchetto in arrivo viene ritrasmesso su tutte le linee, eccetto quella su cui è stato ricevuto. Concepito per reti militari a prova di sabotaggio, se realizzato nel modo sopra descritto massimizza la probabilità che il pacchetto arrivi a destinazione, ma induce un carico elevatissimo sulla rete.

Si può cercare di ridurre il carico utilizzando tecniche di selective *flooding*, in cui i pacchetti vengono ritrasmessi solo su linee selezionate. Un primo esempio, senza applicazioni pratiche, è l'algoritmo *random walk* che sceglie in modo pseudo-casuale su quali linee ritrasmettere il pacchetto. Una miglioria più efficace si ha scartando i pacchetti troppo vecchi, cioè quelli che hanno attraversato molti *router*: a tal scopo nell'***header*** del pacchetto viene inserito un *age-counter*.

Un'ultima miglioria, ancora più significativa, consiste nello scartare un pacchetto la seconda volta che passa in un nodo: in tal modo si realizza una tecnica per trasmettere efficientemente la stessa informazione a tutti i nodi, qualsiasi sia la topologia. Lo svantaggio è che bisogna memorizzare tutti i pacchetti su ogni nodo per poter verificare se sono già passati.

Routing dinamico

Negli algoritmi di instradamento dinamico (adattativo) le tabelle dipendono dalle informazioni raccolte sulla topologia della rete, sul costo dei cammini e sullo stato degli elementi che la compongono. Gli algoritmi adattativi possono essere **centralizzati**, **isolati** o **distribuiti**.

Routing dinamico centralizzato

Il *routing* centralizzato è quello che più si avvicina al *fixed directory routing*. Presuppone l'esistenza di un **RCC** (**R**outing **C**ontrol **C**enter) che conosce la topologia della rete, riceve da tutti i nodi informazione sul loro stato e su quello dei collegamenti, calcola le tabelle di instradamento e le distribuisce.

È un metodo che consente una gestione della rete molto accurata, in quanto permette di calcolare le tabelle anche con algoritmi molto sofisticati, ma implica l'esistenza di un unico gestore, ipotesi questa oggi molto spesso non realistica.

Il RCC, per ragioni di affidabilità, deve essere duplicato e la porzione di rete intorno ad esso è soggetta ad un elevato volume di traffico di servizio: informazioni di stato che arrivano al RCC e tabelle di instradamento che escono dal RCC.

In caso di guasti gravi possono verificarsi situazioni in cui il RCC perde il contatto con una parte periferica della rete e si verificano quindi degli aggiornamenti parziali di tabelle che possono determinare situazioni di *loop*.

Routing dinamico isolato

Ogni IS calcola in modo indipendente le tabelle di instradamento senza scambiare informazioni con gli altri IS. Esistono due algoritmi di *routing* isolato riportati in letteratura: "*hot potato*" e "*backward learning*".

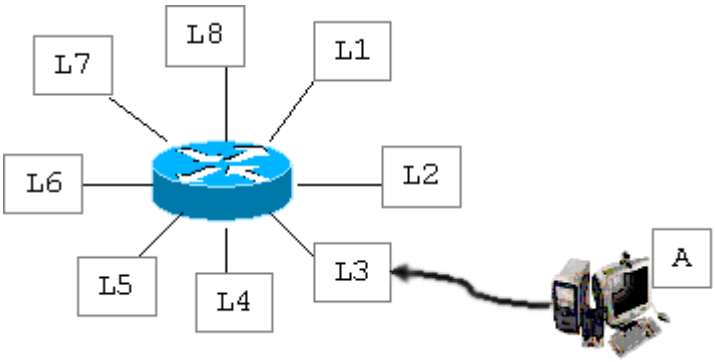
hot potato

Ogni IS considera un pacchetto ricevuto come una patata bollente e cerca di liberarsene nel minor tempo possibile, ritrasmettendo il pacchetto sulla linea con la coda di trasmissione più breve.

backward learning

L'IS acquisisce una conoscenza indiretta della rete analizzando il traffico che lo attraversa: se riceve un pacchetto proveniente dal nodo A sulla linea L3, il *backward learning* impara che A è raggiungibile attraverso la linea L3. è possibile migliorare il *backward learning* inserendo nell'*header* del pacchetto un campo di costo inizializzato a zero dalla stazione mittente ed

incrementato ad ogni attraversamento di un IS. In tale modo gli IS possono mantenere più alternative per ogni destinazione, ordinate per costo crescente.



Il limite di questo metodo consiste nel fatto che gli IS imparano solo le migliorie e non i peggioramenti nello stato della rete: infatti se cade un link e si interrompe un cammino, semplicemente non arrivano più pacchetti da quel cammino, ma non giunge all'IS nessuna informazione che il cammino non è più disponibile. Per tale ragione occorre limitare temporalmente la validità delle informazioni presenti nelle tabelle di instradamento: ad ogni *entry* viene associata una validità temporale che viene inizializzata ad un dato valore ogni volta che un pacchetto in transito conferma l'*entry*, e decrementata automaticamente con il passare del tempo. Quando la validità temporale di un'*entry* giunge a zero, questa viene invalidata ed eliminata dalla tabella di instradamento. Qualora ad un IS giunga un pacchetto per una destinazione ignota, l'IS ne fa il *flooding*. Il *backward learning* può generare *loop* su topologie magliate, per cui, ad esempio nei *bridge*, lo si integra con l'algoritmo di *spanning tree* per ridurre la topologia magliata ad un albero.

Questo metodo metodo è utilizzato per calcolare le tabelle di instradamento nei *bridge* conformi allo standard IEEE 802.1D.

Routing dinamico distribuito

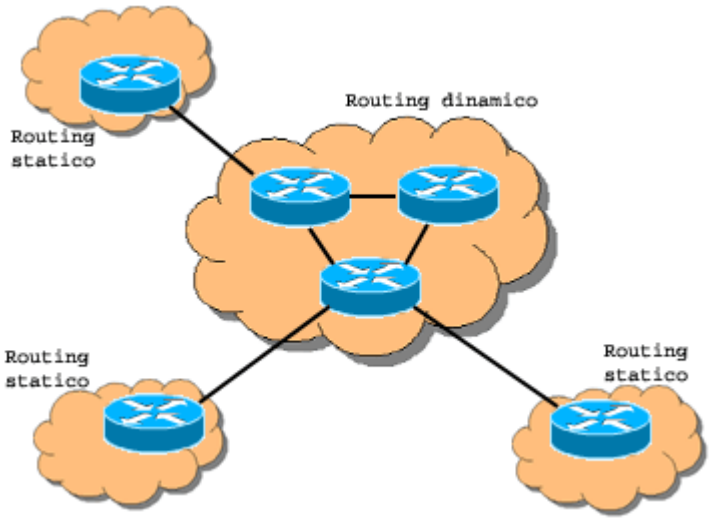
Il *routing* distribuito si pone come una scelta di compromesso tra i due precedenti: non esiste un RCC, ma le sue funzionalità sono realizzate in modo distribuito da tutti gli IS della rete, che a tal scopo usano un protocollo di servizio per scambiare informazioni tra loro ed un secondo protocollo di servizio per scambiare informazioni con gli ES. Tali protocolli vengono detti di servizio in quanto non veicolano dati di utente, che sono gestiti da un terzo protocollo, ma solo informazioni utili al calcolo delle tabelle di instradamento e al ***neighbor greetings***.

Le tabelle di instradamento vengono calcolate a partire dai due **parametri di ottimalità** (costo e *hop*). Il costo di ciascuna linea di ciascun *router* è un parametro che viene impostato dal *network manager* tramite il *software* di gestione dei *router* stessi.

Gli algoritmi di *routing* distribuito sono oggi adottati da DECnet, TCP/IP, OSI, APPN, ecc., e si suddividono ulteriormente in due famiglie: algoritmi ***distance vector*** e algoritmi ***link state*** *packet*.

Routing statico e dinamico: dove impiegarli

Dovendo trarre delle conclusioni, il ***routing* statico** meglio si adatta a situazioni periferiche di dimensione ridotta; dove cioè sia semplice gestire l'intera rete. Al contrario dove siano possibili frequenti cambi di topologia, variazioni di prestazioni in dipendenza da carico di lavoro del *router* e dei link, guasti, percorsi magliati ... allora sarà bene utilizzare ***routing* dinamico**.



Algoritmi e protocolli di routing: criteri di ottimalità

Per la scelta di un algoritmo di instradamento esistono più criteri di ottimalità spesso contrastanti, ad esempio minimizzare il ritardo medio di ogni pacchetto o massimizzare l'utilizzo delle linee. A tal fine si dovrà disporre di una serie di parametri misurabili in base ai quali le caratteristiche di un percorso possano essere confrontate, per scegliere il migliore tra due cammini alternativi.

Gli unici due parametri universalmente accettati sono:

- il numero di salti effettuati (*hop*), cioè il numero di IS attraversati lungo un percorso;
- il costo, cioè la somma dei costi di tutte le linee attraversate lungo un percorso.

Entrambi questi parametri sono di demerito, in quanto il costo di una linea è assegnato in modo inversamente proporzionale alla velocità della linea stessa, e gli *hop* indicano *router* attraversati e quindi ritardi introdotti.

Il metodo appena introdotto, non tiene conto dei problemi di carico dinamico della rete. Le tecniche più moderne consentono di operare un bilanciamento del traffico (*load splitting*) tra cammini paralleli, eventualmente attivando circuiti commutati, quali quelli forniti da una rete alternativa in presenza di un guasto (ad esempio, funzionalità di *backup* di un **CDN**) o per gestire un eccesso di traffico su di un link (traffico di trabocco).

La scelta dell'algoritmo di instradamento ottimale è anche complicata dalle limitate risorse di memoria e CPU disponibili oggi sui *router*, specialmente se confrontate con la complessità delle reti ed in particolare con l'elevato numero di nodi collegabili con una topologia qualsiasi. Algoritmi troppo complessi, operanti su reti molto grandi, potrebbero richiedere tempi di calcolo inaccettabili.

Algoritmi e protocolli di routing: caratteristiche degli algoritmi

Riassumendo, le caratteristiche che in generale si richiedono ad un algoritmo di *routing* sono:

- **Semplicità**: l'algoritmo deve essere funzionalmente efficiente con un minimo *software* e una bassa utilizzazione delle risorse *hardware*, poiché i *router* hanno CPU e memoria finite e devono impiegare la maggior parte del loro tempo a instradare pacchetti, e non a calcolare nuove tabelle di instradamento.
- **Robustezza**/**adattabilità**: di fronte a guasti *hardware*, variazioni di topologia, alto traffico, l'algoritmo deve continuare a lavorare.
- **Ottimizzazione**: è l'abilità dell'algoritmo a scegliere la strada migliore. La strada dipende dalla metrica (unità di misura per calcolare la lunghezza del percorso).

- **Stabilità**: quando ad esempio una rete diviene irraggiungibile, i *router* distribuiscono messaggi di aggiornamento di tale cambiamento a tutta la rete nel più breve tempo possibile, perché in caso contrario si potrebbero verificare dei "*routing loop*". Inoltre l'algoritmo deve sempre convergere velocemente ad un instradamento stabile, cioè non deve modificare le tabelle di instradamento se non a fronte di una variazione di topologia.
- **Equità**: nessun nodo deve essere privilegiato o danneggiato.

Gli algoritmi di *routing* non adattativi (**statici** e deterministici) utilizzano criteri fissi di instradamento mentre gli algoritmi adattativi (**dinamici** e non deterministici) calcolano le tabelle di instradamento in funzione della topologia della rete e dello stato dei link.

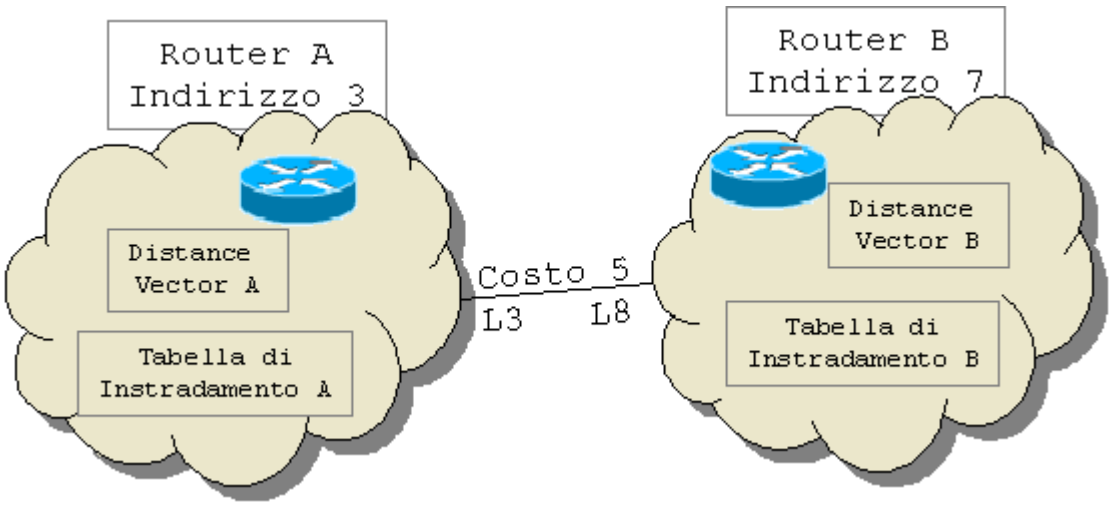
Algoritmi e protocolli di routing: Distance Vector (Bellman-Ford)

L'algoritmo *distance vector* è anche noto come algoritmo di *Bellman-Ford*. Abbiamo visto che ogni *Intermediate System* mantiene una tabella di instradamento, che gli indica attraverso quale interfaccia raggiungere la destinazione collegata. Il *router*, per realizzare tale algoritmo, gestisce una ulteriore struttura dati, detta vettore delle distanze (*distance vector*), per ogni linea.

Il vettore delle distanze è una struttura dati composta da:

- indirizzo di destinazione;
- minimo costo associato ad un *route* verso la destinazione.

L'algoritmo prevede che l'*Intermediate System* invii su tutte le proprie interfacce l'elenco delle destinazioni che è in grado di raggiungere e la distanza da esse. Il *distance vector* associato a ciascuna linea nella tabella di instradamento, contiene informazioni ricavate dalla tabella del *router* collegato all'altro estremo della linea (si veda figura).



Un apparato riceve un *distance vector* da ognuno dei suoi vicini e lo memorizza dopo aver sommato alle distanze annunciate la distanza tra sé e il vicino che ha inviato il vettore. Nel caso in figura, le distanze verranno sommate al costo del percorso tra il *router* A e B, ossia 5.

A partire da tale conoscenza l'apparato costruisce la propria tabella di *routing* con una semplice operazione di fusione (*merge*) dei vettori.

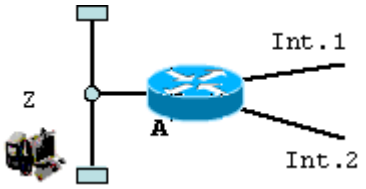
Algoritmi e protocolli di routing: Distance Vector - merge

Quando un *router* memorizza un *distance vector* nella sua struttura dati locale, verifica se sono

presenti variazioni rispetto al *distance vector* precedentemente memorizzato: in caso affermativo ricalcola le tabelle di instradamento fondendo (*merge*) tutti i *distance vector* delle linee attive. Analoga operazione di ricalcolo avviene quando una linea passa dallo stato ON allo stato OFF o viceversa. Alcune implementazioni di protocolli *distance vector* inviano anche i *distance vector* periodicamente, ad esempio il RIP, invia il *distance vector* ogni 30 secondi.

La fusione avviene secondo il criterio di convenienza del costo: a parità di costo secondo il minimo numero di *hop* e a parità di *hop* con scelta casuale. Se la tabella di instradamento risulta variata rispetto alla precedente, il *distance vector* relativo viene inviato ai *router* adiacenti.

Con 'Ind', identifichiamo un IS on un ES, senza alcuna distinzione. Quindi le lettere identificano *host* o *router* della rete.



Interfaccia 1		**Interfaccia 2**	
Ind	Costo	Ind	Costo
A	9	A	10
B	11	B	0
C	0	C	11
D	2	D	9
E	7	E	4
F	8	F	6
X	3	X	10
Y	10	Y	11
W	10	W	8
Z	13	Z	10
Costo Int.1		Costo Int.2	
+9		+10	

Nella tabella qui sopra sono riportati i vettori delle distanze ricevuti dal *router* A su due sue interfacce. La numero uno ha un costo di 9 e la due un costo di dieci.

Nella fase di *merge*, si compone una tabella di *routing* come di seguito visualizzata.

Ind	**Costo**	**Interfaccia**
A	0	se stesso
B	10	2
C	9	1
D	11	1
E	14	2
F	16	2

X	12	1
Y	19	1
W	18	2
Z	20	2

La fusione, in particolare, tiene conto anche di percorsi fittizi, quali le destinazioni collegate direttamente e l'apparato stesso. Nell'ipotesi in cui la destinazione 'Z' sia direttamente collegata all'apparato, nella tabella risultante, troveremo scritto un valore 0 come costo.

Al termine della fusione, se questa implica la modifica della tabella precedentemente memorizzata sul *router*, avverrà la trasmissione su tutte le interfacce del nuovo vettore delle destinazioni.

Il modo di operare fin qui esaminato, se esaminato su ampia scala, genera un fenomeno simile a quello generato da una pietra che cade in uno stagno d'acqua:

- La pietra è una variazione dello stato della rete,
- lo stagno è la rete,
- le onde generate dalla caduta della pietra sono i *distance vector* che si dipartono dal luogo di impatto, arrivano ai bordi della rete, si specchiano e tornano verso il centro e ancora verso la periferia e poi verso il centro, con un moto che si ripete più volte prima di giungere a stabilità (acqua ferma).

Algoritmi e protocolli di routing: Distance Vector - cold start

Si supponga di produrre una inizializzazione contemporanea di tutti i nodi, un *cold start*. Durante questo stato, ciascuno dei nodi è caratterizzato unicamente da una conoscenza locale, il che significa che ciascun nodo conosce il proprio indirizzo, ma ignora totalmente la topologia della rete. La tabella di *routing* sarà quindi minima, e conterrà il route dell'apparato stesso e di tutti gli *End System* ad esso collegato. La conoscenza degli ES è dovuta, con IP, a configurazione manuale, o a specifici protocolli di ***neighbor greetings***.

Dalla tabella minimale, sarà estratto il vettore delle distanze che verrà trasmesso a tutti i vicini. All'atto della ricezione, gli apparati sono in grado di costruire una tabella comprendente un numero maggiore di destinazioni e di trasmettere vettori delle distanze sempre più ricchi.

Con il susseguirsi di questi scambi, ogni apparato della rete raccoglie informazioni sulla raggiungibilità delle destinazioni di tutta la rete. La convergenza dei vari distance-vector in transito, porta alla stabilizzazione.

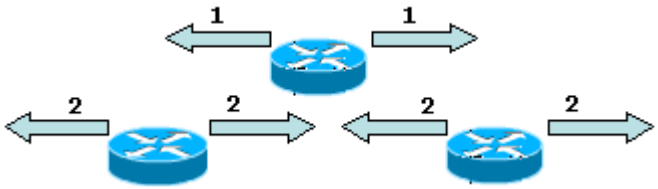
Solo ora le informazioni contenute nelle tabelle di instradamento sono da ritenersi corrette.

Algoritmi e protocolli di routing: Distance Vector - updates

Si ripresenta la similitudine con l'effetto generato da una pietra che cade in uno stagno d'acqua (in questo caso per la propagazione delle modifiche topologiche). Nell'ipotesi di un guasto ad un'interfaccia, il *router* dovrà ricalcolare la tabella di instradamento dai vettori delle distanze delle interfacce rimaste funzionanti. In caso di una tabella risultante diversa da quella precedentemente calcolata, si dovrà optare per la ritrasmissione a tutti della nuova tabella.

In questo modo, anche gli altri apparati, dovranno verificare le novità apportate. La verifica non tiene conto della topologia, bensì calcola semplicemente l'unione delle tabelle come precedentemente mostrato (***merge***).

Analogamente al malfunzionamento, anche un cambio della topologia volontario, provoca una reazione a catena che comincia nella zona circostante al cambiamento e si propaga sino ai bordi della rete, e contemporaneamente anche nella direzione opposta.



Il modo in cui l'algoritmo reagisce ad un cambiamento topologico, produce le seguenti conseguenze:

- un IS può ricalcolare più di una volta la propria tabella di *routing* a seguito di un cambiamento topologico.
- non si può considerare raggiunta la convergenza, fino a che tutti gli IS non abbiano terminato di calcolare le proprie tabelle, determinando che non è più necessario ritrasmettere il vettore delle distanze ai vicini.

In ultima analisi, si determina che la velocità di convergenza è limitata dall'apparato e dai collegamenti più lenti della rete.

Algoritmi e protocolli di routing: Distance Vector - sommario

L'algoritmo *distance vector* è fatto di meccanismi estremamente semplici, che ne hanno determinato la facile distribuzione e lo sviluppo sugli apparati di instradamento. Al contempo, dimostra una serie di limiti tra i quali:

- Gli apparati per *internetworking* non sono in grado di rendersi conto se stanno cercando di inoltrare il pacchetto su di un percorso chiuso (*loop*).
- La quantità di informazioni di servizio che gli IS si scambiano è considerevolmente alta e sottrae risorse al traffico dati. Va ricordato che il vettore delle distanze contiene informazioni su tutte le destinazioni che esso è in grado di raggiungere.
- La complessità dell'algoritmo, fa sì che la convergenza del *routing* si raggiunga solo dopo un numero di operazioni pari al quadrato o al cubo dei nodi della rete. Questa differenza tra valore minimo e massimo dipende dal numero di collegamenti che la rete dispone. Ad esempio una rete che cresce di 10 volte avrà complessità in aumento da 100 a 1000 volte.

Algoritmi e protocolli di routing: Link State

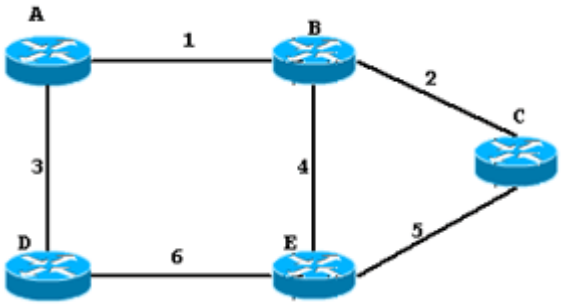
I protocolli di tipo *Link State* sono basati sul concetto di mappa distribuita:

- tutti i nodi posseggono una copia della mappa della rete, che viene regolarmente aggiornata.

Gli IS, dalla copia locale della mappa della rete, eseguono un calcolo completo dei migliori percorsi. La mappa è contenuta in un *database*, dove ciascun *record* rappresenta un link nella rete.

Ciascun *record* contiene un identificatore di interfaccia e le informazioni che descrivono lo stato del link (la destinazione e la distanza o metrica). Con queste informazioni ogni nodo può facilmente calcolare il percorso più breve da sé stesso a tutti gli altri nodi. Con la velocità degli attuali *computer*, è necessario solo un brevissimo tempo, tipicamente una frazione di secondo se la rete non è molto estesa, per calcolare tali percorsi.

Poiché tutti i nodi contengono lo stesso database ed eseguono lo stesso algoritmo di *route-generation*, i percorsi sono coerenti e non si verificano *loop*.



Da	A	Link	Distanza
A	B	1	1
A	D	3	1
B	A	1	1
B	C	2	1
B	E	4	1
C	B	2	1
C	E	5	1
D	a	3	1
D	E	6	1
E	B	4	1
E	C	5	1
E	D	6	1

Si consideri l'invio di un pacchetto dal nodo A al nodo C nella rete riportata in figura, e ci si basi sui calcoli tra i nodi A e B. Il nodo A può rilevare, tramite il database, che il percorso più corto verso il nodo C passa attraverso il nodo B e quindi invia il pacchetto sul link numero 1. Il nodo B, a sua volta, invierà il pacchetto sul link numero 2.

Algoritmi e protocolli di routing: Link state - LSP, adiacenze e flooding

Ai fini della costruzione della mappa della rete, i *router* invieranno e riceveranno speciali pacchetti di servizio detti ***L**ink **S**tate **P**acket* (**LSP**).

Il *Link State Packet* (LSP) contiene:

- Stato di ogni link connesso al *router*.
- Identità di ogni vicino connesso all'altro estremo del link.
- Costo del link.
- Numero di sequenza per il LSP
  (a seguito di frequenti variazioni di topologia un *router* può ricevere un LSP vecchio dopo uno nuovo, quindi deve essere in grado di valutare il più recente).
- ***Checksum***.
- *Lifetime*
  (la validità di ogni LSP è limitata nel tempo; diversamente un errore sul numero di sequenza potrebbe rendere un LSP valido per anni).

La generazione del LSP avviene su base periodica, oppure quando viene rilevata una variazione nella topologia locale (adiacenze), ossia:

- Viene riconosciuto un nuovo vicino.
- Il costo verso un vicino è cambiato.
- Si è persa la connettività' verso un vicino precedentemente raggiungibile.

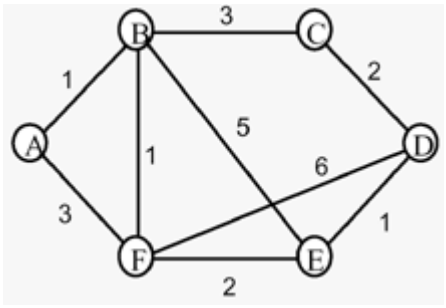
Il LSP è trasmesso in ***flooding*** selettivo su tutti i link del *router* e tutti i *router* del dominio di *routing* ricevono il LSP. All'atto del ricevimento di un LSP il *router* compie le seguenti azioni:

- Se non ha mai ricevuto LSP da quel *router* o se il LSP è più recente di quello precedentemente memorizzato (campo *Sequence Number*), memorizza il pacchetto e lo ritrasmette in *flooding* su tutte le linee eccetto quella da cui l'ha ricevuto.
- Se il LSP ha lo stesso numero di sequenza di quello posseduto non fa nulla.
- Se il LSP è più vecchio di quello posseduto trasmette al mittente il pacchetto più recente.

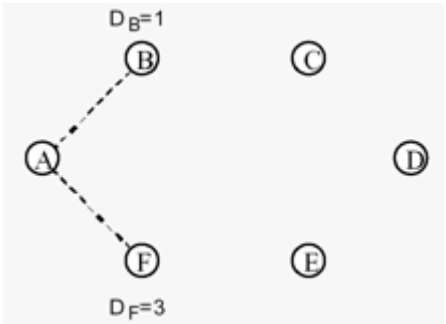
Algoritmi e protocolli di routing: Link state - algoritmo Dijkstra e SPF

Ogni nodo ha a disposizione il grafo pesato della rete ed assegna a tutti gli altri nodi un'etichetta che rappresenta il costo massimo per la raggiungibilità del nodo in esame; l'algoritmo di calcolo modifica tali etichette cercando di minimizzarle e di renderle permanenti.

L'*intermediate System* che voglia calcolare il proprio elenco di percorsi più brevi, si basa sui principi dell'algoritmo Dijkstra. Si visita ogni nodo della rete a partire da se, un nodo alla volta; il successivo nodo visitato è quello più vicino a uno dei nodi già visitati (da ciò il nome **SPF**, ***S**hortest **P**ath **F**irst*: prima il percorso più breve).

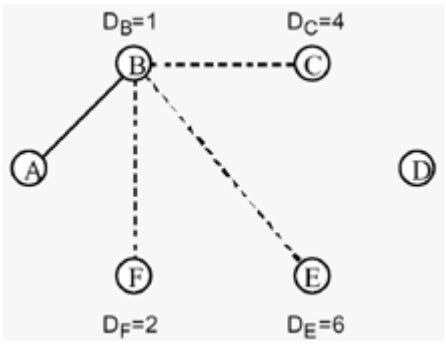


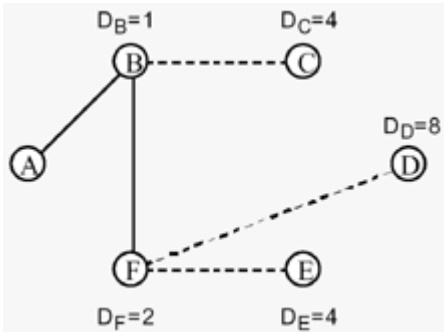
Si consideri l'insieme di nodi a fianco mostrato.



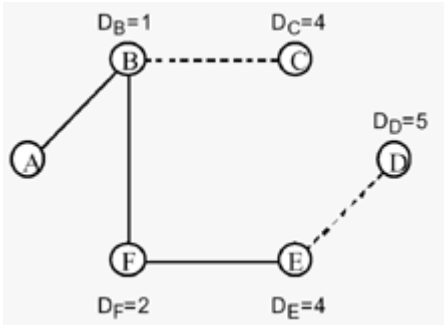
Il punto di partenza arbitrariamente designato, è il nodo A. Quindi sarà analizzato il grafo pesato a partire dal nodo A. Notare il costo del percorso evidenziato sopra ogni nodo di destinazione.

Si procede scegliendo sempre per primi i percorsi meno costosi. In questo caso il passo meno costoso, sta nel passaggio per B.

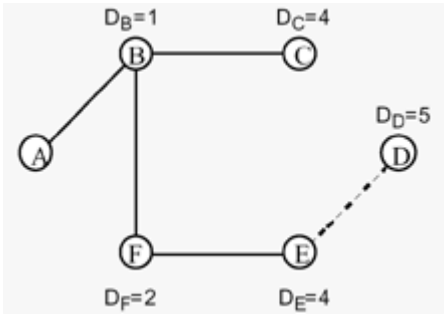




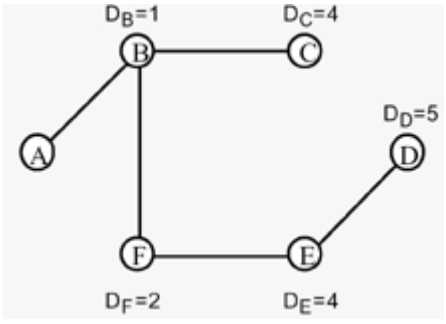
Sempre con il medesimo principio viene scelto in nodo F.



Nodo scelto: E



Nodo scelto: C



Nodo scelto: D
Alla fine, tutti i nodi sono raggiungibili senza maglie (*loop*).

Algoritmi e protocolli di routing: Link state - proprietà

Il *routing link-state* è caratterizzato da una elevata stabilità, cioè brevi tempi di convergenza e bassissima inclinazione a creare dei *loop* di instradamento. Questi pregi derivano da:

- il numero di passi che portano allo *shortest path* è dell'ordine di C log N; dove C è il numero di collegamenti e N è il numero di nodi.

Diversamente da ***Bellman Ford***, dove si cresceva in passi da compiere quadraticamente o cubicamente, in questo caso si avrà aumento lineare nei confronti della complessità topologica e logaritmico della dimensione della rete.
- avvenendo il calcolo dello *shortest path* in maniere indipendente tra gli apparati, la lentezza di calcolo di un singolo non pregiudicherà le prestazioni degli altri.
- i tempi di propagazione degli LSP non pregiudicano la consegna dei pacchetti paganti, poiché tanto più gli apparati sono lontani dalla zona dove è avvenuto un ipotetico cambiamento, tanto più sarà probabile che non porti alla modifica delle tabelle di *routing*.

Va aggiunto il non indifferente vantaggio della ridotta dimensione dei pacchetti LSP; in tali pacchetti di servizio verranno incapsulate solo informazioni relative ai collegamenti con i vicini, piuttosto che con il vettore distanza dove è necessario inserirvi tutta la tabella di instradamento.

Autonomous System

Ogni *Router* vede crescere la propria tabella di *routing* in maniera lineare rispetto al numero di reti da instradare direttamente. Nei primi anni ottanta, Internet era considerata come una *single network* (SN) dalle dimensioni geografiche molto estese. L'introduzione di sistemi che permettessero ai *router* di adattarsi dinamicamente al mutare della tipologia di connessione costituiva una soluzione parziale al problema.

Se da una parte diminuiva il tasso di errore e la necessità di coordinamento umano, aumentando contemporaneamente la tolleranza ai guasti delle singole connessioni, dall'altra ci si rese conto che questa scelta non risolveva il problema della crescita delle tabelle di *routing*.

La decisione intrapresa fu quella di abbandonare il modello SN per suddividere Internet in un certo numero di *Autonomous System* (**AS**) costituiti da un insieme di *router* e LAN sotto la medesima amministrazione.

Questa definizione, non molto flessibile, è stata in seguito sostituita da una più funzionale che prevede che i *router* all'interno di ogni AS siano reciprocamente raggiungibili. Le informazioni di raggiungibilità sono, preferibilmente, scambiate mediante uno o più protocolli adattivi detti ***Interior Protocol***. Inoltre si prevede che gli AS si scambino informazioni sulla rispettiva raggiungibilità utilizzando un protocollo opportuno genericamente designato come ***Exterior Protocol***.

La definizione classica di un sistema autonomo è quella di un insieme di *router* sotto una singola organizzazione ed amministrazione tecnica, utilizzante un IGP e metriche ad *hop* per gestire l'instradamento interno all'AS stesso, e un EGP per gestire il *routing* verso altri AS.

In molte monografie il termine *Autonomous System* viene utilizzato parallelamente alla dizione Dominio di *routing*, per intendere una zona sottoposta alla gerarchia di un *router* principale (*Master*).

Se un *gateway* deve instradare un messaggio a un altro *gateway* appartenente allo stesso AS, avrà nella propria tavola di *routing* l'informazione di raggiungibilità idonea. Se al contrario è necessario raggiungere un *gateway* che appartiene ad un differente AS il messaggio viene inviato attraverso una coppia di *router* particolari detti *exterior router*, almeno uno per AS.

Ciascun *exterior router* conosce le reti raggiungibili utilizzando i link che lo collegano agli altri *exterior router* ma non conosce il modo in cui queste reti sono di fatto connesse all'interno dei rispettivi AS (secondo un modello tipicamente gerarchico).

Autonomous System: IGP - la famiglia di protocolli inter-dominio

Il protocollo IGP (*Interior Gateway Protocol*) maggiormente utilizzato oggi su Internet è senza dubbio il protocollo RIP.

RIP è l'acronimo di ***R**outing **I**nformation **P**rotocol* ed è un protocollo relativamente semplice appartenente alla famiglia di protocolli di tipo *distance vector*.

Sviluppato dalla *Xerox* per XNS, nel 1982 il RIP è stato adattato per il TCP/IP con lo *UNIX* BSD.

Si tratta di un protocollo di *routing* intradominio basato su un algoritmo di tipo *distance vector*, ed è stato adottato dall'IETF nello RFC 1058 (1988) e nello RFC 1388 (1993). È inoltre utilizzato in diverse implementazioni proprietarie tra cui lo RTMP di *AppleTalk*.

È il protocollo di *routing* IP più vecchio ancora in uso: la versione IP esiste in due versioni, la seconda versione aggiunge nuove funzionalità a questo protocollo. RIPv1 è di tipo *classfull* mentre RIPv2 è *classless*.

L'algoritmo ***distance vector*** è stato sviluppato da *Bellman*, *Ford* e *Fulkerson* nel lontano 1969, poi è stato integrato dalla *Xerox Network System* nei suoi protocolli come XNS RIP. Tale protocollo è stato precursore di comuni protocolli di *routing*come *Novell* IPX RIP, *AppleTalk Routing Table Maintenance Protocol* (RTMP) e naturalmente IP RIP. Nella versione 4.2 del *Berkley UNIX* rilasciata nel 1982 il RIP è implementato come un processo demone chiamato *routed*: ancora molte versioni di *UNIX* implementano il RIP.

Autonomous System: IGP - RIP (caratteristiche)

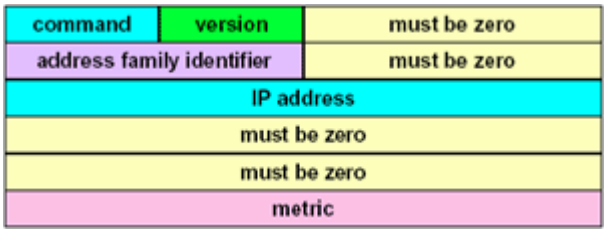
Gli indirizzi presenti nelle tabelle RIP sono indirizzi Internet a 32 bit. Una voce (*entry*) nella tabella di *routing* può rappresentare un *host*, una rete o una sottorete. Non sono presenti specifiche sul tipo di indirizzo nei pacchetti RIP; è compito dei *router* analizzare l'indirizzo per capire di cosa si tratta. Questi, prima separano la parte di rete dalla parte sottorete + *host* in funzione della classe dell' indirizzo (A, B o C). Se la parte sottorete+*host* è nulla, l'indirizzo rappresenta una rete, viceversa può rappresentare sia una sottorete che un *host*. Al fine di discriminare tra queste 2 possibilità, è necessario conoscere la *subnet mask*; se la parte *host* è nulla, si tratta dell'indirizzo di una sottorete, di un *host* viceversa.

Di *default*, RIP utilizza una **metrica** molto semplice: la distanza (*hop count*) è il numero di links che vengono attraversati per raggiungere la destinazione. Questa distanza è espressa come un numero intero variabile tra 1 e 15; 16 rappresenta una distanza infinita.

RIP supporta sia i links punto a punto che le reti di tipo *broadcast* come *Ethernet*. I pacchetti RIP vengono impacchettati nei pacchetti **UDP** e IP; i processi RIP utilizzano la **porta 520** sia per la trasmissione che per la ricezione. Utilizzando una porta specifica, minore di 1024, si è in linea con le protezioni di sistema di BSD-*Unix*.

I pacchetti normalmente sono inviati in modalità *broadcast*, ovvero saranno ricevuti da tutti i *routers* connessi alla rete. Normalmente i pacchetti vengono inviati ogni 30 secondi, o meno, nel caso di aggiornamenti alle tabelle. Se una *route* non viene aggiornata entro 3 minuti, la distanza viene fissata ad infinito e l'*entry* verrà successivamente rimossa dalle tabelle. Allo scopo di evitare aggiornamenti troppo frequenti , questi vengono regolati da un *timer* casuale, che può variare tra 1 e 5 secondi.

Command
request = 1 utilizzato in fase di inizializzazione per



richiedere un *distance vector*
response = 2 n utilizzato normalmente per distribuire i *distance vector*

Come osservabile dal pacchetto, il protocollo RIP prevede un comando di richiesta e uno di risposta o aggiornamento. Normalmente RIP opera in risposta, a intervalli regolari di 30 secondi, o in seguito a richieste di aggiornamento delle tabelle di *routing*. Il processo RIP, in seguito alla ricezione di un messaggio di risposta, aggiorna la propria tabella. Ogni voce della tabella sarà al limite composta da:

- Indirizzo di destinazione.
- Metrica associata con la destinazione.
- Indirizzo del *next router*.
- Un *recently updated flag*.
- Numerosi *time*.

Autonomous System: IGP - RIP

Elaborando le risposte in arrivo, il *router* esaminerà le voci una ad una ed eseguirà una serie di controlli, quali la verifica della validità dell'indirizzo e l'appartenenza ad una delle classi A, B o C, che il numero identificante la rete non sia 127 (*loop-back*) o zero (ad eccezione dell'indirizzo di *default* 0.0.0.0), che la parte *host* non sia un indirizzo *broadcast* e che la metrica non sia maggiore di 15 (infinito). In ogni caso voci non corrette vengono ignorate.

Se la metrica in arrivo risulta diversa da infinito, viene incrementata di 1 per il successivo *hop*, quindi la tabella di *routing* viene scandita per una voce corrispondente alla destinazione e viene quindi eseguito il generico processo **distance vector**, di seguito illustrato.

- Se la e la voce non è presente e la sua metrica nel messaggio ricevuto non è infinito, la aggiunge alla tabella, inizializzando la metrica al valore ricevuto ed il *next router* al mittente del messaggio, prima di avviare un *timer* per quella voce.
- Se la voce è presente con una metrica più grande, aggiorna i campi della metrica e del *next router* e riavvia il *timer* per quella voce.
- Se la voce è presente ed il *next router* è il mittente del messaggio di risposta, aggiorna la metrica se questa differisce dal valore memorizzato e, in tutti i casi, riavvia il *timer*.
- In tutti gli altri casi, il messaggio ricevuto è ignorato.

Se la metrica o il *next router* cambiano, l'*entry* viene marcata come aggiornata. Un messaggio di risposta viene inviato ad intervalli regolari di 30 secondi o può essere attivato in seguito ad un aggiornamento alle tabelle di *routing*. Questo può essere causa di eccesso di traffico sulla rete e l' RFC-1058 specifica una serie di precauzioni da adottare a questo preciso scopo. La risposta non dovrebbe essere inviata immediatamente in seguito alla ricezione dell' aggiornamento ma, piuttosto, dopo un piccolo intervallo *random*, variabile tra 1 e 5 secondi. Questo permette ai relativi aggiornamenti provenienti dai nodi vicini di venire riassunti nel successivo messaggio, limitando così il carico di rete.

Un messaggio di risposta separato viene preparato per tutte interfacce connesse. L'informazione può variare in seguito al processo di *split horizon*. Il messaggio normalmente include le coppie indirizzo e metrica per tutte le voci della tabella ma, se il messaggio è inviato come un aggiornamento, non deve necessariamente includere tutte le voci, ma solo quelle che sono state aggiornate rispetto

all'ultima trasmissione. Il massimo formato del pacchetto è 512 *bytes*, che permette di avere sino a 25 voci per messaggio. Nel caso di un maggiore numero di voci, RIP invierà più pacchetti. L'indirizzo sorgente del messaggio dovrebbe sempre coincidere con l'indirizzo IP associato con l'interfaccia alla quale il messaggio è inviato.

I processi RIP possono anche ricevere messaggi di richiesta. Una richiesta viene normalmente inviata quando un *router* inizia le operazioni allo scopo di ottenere dai suoi vicini il valore iniziale delle tabelle di *routing*. Esistono 2 possibili forme di richiesta, quella per una lista completa delle tabelle di *routing* o quella per sole specifiche *routes*.

Una delle richieste di lista completa delle tabelle di *routing* si ha specificando solo le coppia indirizzo + metrica per l'indirizzo di *default* 0.0.0.0 con una metrica pari ad infinito. In questo caso, il *router* replicherà con una tipica risposta, simile a quelle inviate periodicamente nelle normali operazioni del protocollo, incluso il processo di *split horizon*.

Qualsiasi altra forma di richiesta prevede la lista delle sole voci specificate. La risposta verrà inviata in modalità *point to point* al richiedente e conterrà una copia esatta dell'informazione di distanza nelle tabelle di *routing*, senza eseguire il processo di *split horizon*. Questa forma di richiesta ha poco significato per le normali operazioni, mentre ne assume molto per scopi di *debugging*.

Autonomous System: IGP - RIP (parametri)

Timers

Il protocollo RIP gestisce i seguenti *timers*:

- *Routing update timer* (*default* 30 s): intervallo di tempo per l'invio degli annunci.
- *Route invalid timer* (*default* 90 s): intervallo di tempo dopo il quale una *route* è dichiarata irraggiungibile (distanza posta ad infinito).
- *Route flush timer* (*default* 270 s): intervallo di tempo dopo il quale la *route* è cancellata dalla *routing table*.
- *Triggered updates*: sono inviate con un ritardo casuale compreso tra 1 e 5 secondi, per evitare intasamenti della rete e per far sì di poter eventualmente comunicare più cambi di *route* con un messaggio solo.

Stabilità

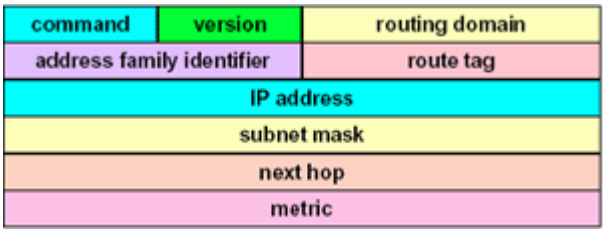
Al fine di assicurare una buona stabilità e, quindi, di evitare situazioni di *loop* e, di conseguenza, possibili congestioni della rete, RIP utilizza una serie di tecniche, di seguito descritte:

- *Triggered updates*: si tratta di messaggi di *routing update* fatti anzitempo, causati da variazioni di connettività. Nel caso vengono inviate solo le informazioni relative alle *route* che sono cambiate (non viene trasferito il *distance vector* completo). Evitano alla rete di trovarsi in uno stato incoerente, fino allo scadere del *Routing Update Timer*, quando i *distance vector* sarebbero inviati comunque.
- *Hop Count Limit*: fa sì che le destinazioni più distanti di 15 siano considerate irraggiungibili e permette quindi di evitare il *count to infinity problem*.
- *Hold Downs*: Il *router* mette in quarantena le *route* che utilizzavano il link guasto. Inoltre, il *router* che ha rilevato il guasto non può partecipare ad alcun *loop*, almeno fino alla scadenza dell'*Hold Down timer*.
- *Split Horizon*: Se A raggiunge la destinazione X attraverso B, non ha senso per B cercare di raggiungere X attraverso A. Previene il *loop* tra 2 nodi ma non elimina i *loop* con 3 nodi
- *Split Horizon with Poisonous Reverse*: Migliora lo *Split Horizon* puro. Nel momento in cui si verifica una variazione di *route*, lo *Split Horizon* semplice non comunica più la *route* (che

continua però a valere sino alla scadenza del *timer*). Viceversa, con il *Poisonous Reverse* la *route* viene comunicata con costo infinito, quindi gli altri *router* apprendono immediatamente che non possono usare la *route* (non devono aspettare lo scadere del *route invalid timer*). Le *route* non più valide non vengono rimosse dagli annunci ma sono annunciate con metrica 15 (*infinity*).

Autonomous System: IGP - RIPv2

A causa di alcune limitazioni è stato sviluppato un nuovo RIP descritto nella RFC 1723. Questa nuova versione consente l'interoperabilità con RIPv1 ed ha in più la possibilità di trasferire anche le *netmask* e quindi di instradare anche su sottoreti differenti.



RIPv1 non è un protocollo sicuro. Qualsiasi *host* che invia pacchetti dalla porta UDP 520 verrebbe considerato un *router* dai propri vicini, mentre invece solo un utente privilegiato dovrebbe avere il diritto di utilizzare questa porta.

A questo scopo, RIPv2 prevede una procedura di autenticazione che specifica che la prima *entry* in un pacchetto può essere rimpiazzata da un *authentication segment*. Il pacchetto conterrà quindi l'*header* iniziale a 32 bit, seguito da un segmento di autenticazione composto da:

- Il campo **AFI** (***A**ddress **F**amily **I**dentifier*) settato a 0xFFFF.
- Un campo *Authentication Type* (2 *bytes*) che identifica il tipo di algoritmo di autenticazione in uso.
- 16 *bytes* di dati di *autentication*.
- 24 coppie di campi destinazione-metrica.

I *routers* RIPv1 semplicemente rilevano che il campo AFI non risulta pari a 2 e quindi non considerano l'*authentication segment*, procedendo con le 24 rimanenti *entries*. Alla ricezione del pacchetto, il *router* RIPv2 verifica la presenza dei campi di autenticazione e, in caso affermativo, ne rivela l'origine.

Esistono svariati algoritmi di autenticazione, definiti attraverso il campo *Authentication Type*; in comune vi è la protezione dei messaggi di aggiornamento attraverso una *password* cifrata.

Autonomous System: IGP - IGRP

IGRP (***I**nternal **G**ateway **R**outing **P**rotocol*) nasce come un'evoluzione del RIP. Alcune delle modifiche apportate al protocollo originale sono sotto il *copyright* della *Cisco*. Alla base dello IGRP vi è un aggiornamento periodico inviato in *multicast* delle informazioni possedute da ciascun *gateway*. Ogni informazione di *routing* è composta da quattro attributi principali destinati al calcolo del miglior cammino sorgente-destinazione:

- Ritardo (R): indica la somma di tutti i ritardi accumulati nel cammino intrapreso. Il valore può essere calcolato staticamente in base al tipo di reti attraversate o modificato dall'amministratore di sistema.
- Banda (B): rappresenta la banda disponibile sul più lento link attraversato. Il calcolo di questo

valore avviene quasi sempre staticamente.
- Affidabilità (A): rappresenta una stima dell'errore medio presente su ogni connessione fisica ed è calcolato dinamicamente monitorando, istante per istante, le condizioni presenti sulla rete.
- Carico (C): indica il valore di picco registrato sul *router* più occupato attraversato nel cammino sorgente-destinazione.

Durante l'applicazione del classico algoritmo *distance-vector*, la scelta tra due cammini che conducono alla medesima destinazione si effettua in base ad una funzione di metrica che tiene conto dei quattro parametri appena descritti e di alcune costanti che condizionano il risultato finale. Si ricordi che il RIP, usa una metrica molto meno complessa e basata esclusivamente sulla distanza intesa come numero di link attraversati. Uno dei principali problemi evidenziati dal RIP è la difficoltà di rilevare la creazione di un ciclo nel caso di caduta di un link.

A *counting to infinity* il RIP affianca i metodi *Split Horizon*, *Trigger Updtate* e *Poisonous Reverse*. IGRP utilizza, al posto del *Poisonous Reverse*, uno fra i due metodi descritti di seguito:

Path HoldDown si basa su una semplice ma efficace osservazione: la creazione di *loop* si verifica se e solo se un *router*, che ha già rilevato la caduta di un link, riceve l'informazione contraria (per errore o per ritardi di propagazione delle informazioni) da un *router* che non ha rilevato la mutata condizione topologica, prima di riuscire ad immettere sulla rete questo nuovo stato. Partendo da quest'osservazione *Path HoldDown* non appena rileva la caduta di un link l impone un periodo di quarantena durante il quale non è accettato alcun *update* esterno relativo ad l.

Route Poisonous: osserva che la tecnica *counting to infinite*, in presenza di creazione di un *loop*, fa crescere progressivamente la metrica associata ai cammini coinvolti. Se questa condizione si verifica improvvisamente, IGRP considera in modo conservativo il cammino inutilizzabile sino al prossimo *trigger update*. Un effetto collaterale di questa strategia è la creazione di problemi temporanei quando la topologia fisica della rete viene modificata intenzionalmente (ad esempio aggiungendo nuovi *router*).

Oltre ai metodi di prevenzione dei *loop*, una nuova caratteristica del protocollo è la possibilità di mantenere nelle tavole di *routing* non solo il miglior cammino verso una specifica destinazione ma anche percorsi ausiliari da utilizzare come *backup*. L'algoritmo del RIP è leggermente modificato per consentire la memorizzazione anziché lo scarto di un link con metrica maggiore, in caso di presenza di un link con metrica minore.

Autonomous System: IGP - EIGRP

EIGRP è la risposta *Cisco* ad OSPF. La casa americana, pur avendo implementato nei propri *router* quest'ultimo protocollo, sostiene che ci siano dei validi motivi per non abbandonare la tecnologia *distance-vector*. L'*Enhanced* IGRP è essenzialmente un algoritmo *distance-vector* cui è affiancato il metodo conosciuto come *Diffusion Update Algorithm* (DUAL) sviluppato da J.J.Garcia-Luna-Aceves sulla base di un precedente algoritmo di E.W. Dijkstra.

Supposto che ogni *router* h mantenga per ciascuna destinazione j la metrica d (k, j). La metrica è propagata attraverso ciascuno dei k *router* direttamente connessi con h. Sia inoltre l (h, k) il costo associato al link che collega lo stesso h con k.

In condizioni normali per raggiungere j, il *router* h seleziona il *router* direttamente connesso x tale che sia minimizzata la funzione di costo [FC] d (h, j) = l (h, x) + d (x, j).

In altre parole, h privilegia il *router* x che è raggiunto con minore costo rispetto sia al link di connessione sia alla distanza per la destinazione j.

Supponiamo che h, dopo aver ricevuto come update dal *router* y i due valori d1(y, j) ed l1(h, y), verifichi che la nuova somma risulta minore del valore precedentemente calcolato (analiticamente l1(h, y) + d1(y, j) < d(h, j)). Il *router* h selezionerà semplicemente come *next hop* y al posto di x. Al di là delle notazioni, il funzionamento è fino a questo momento semplice e, sorprendentemente, vi sono poche possibilità di creare dei *loop* in caso di caduta di un link. Vediamo perché: se viene effettuato un *update*, che incrementa il costo di un link (come nel caso di creazione di un ciclo), non si modifica la tavola di *routing* perché in ogni caso l'algoritmo impone la scelta del cammino di costo minimo. La condizione precedente è vera con l'eccezione del caso in cui l'*update* incrementa il costo del *router* x correntemente selezionato, in altre parole del cammino con costo di per sé già minimo. In tal caso EIGRP è più complicato poiché cerca nuovamente l'esistenza di un *router* vicino k tale che d (k, j) < d (h, j) dove d (h, j) è il vecchio valore di metrica che ha permesso di selezionare il *router* x prima dello *update* che ha incrementato il costo verso x. Se la selezione da luogo ad un insieme I di *router*, si sceglie quello che minimizza FC. Se non esiste nessun *router* che soddisfi la condizione richiesta la tavola di *routing* per la *entry* relativa alla destinazione j è messa in quarantena fotografando la situazione precedente all'*update* incriminato. Da quel momento viene eseguita la parte *Diffusion* dell'algoritmo. Si noti che in questa fase, in modo poco conservativo, si considera il link verso x come valido anche se, di fatto, esso potrebbe essere interrotto.

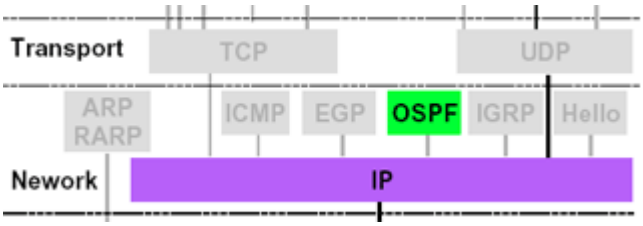
Un *router* entrato in modo *Diffusion* invia una *query* a tutti i vicini con l'esclusione del *router* x, segnalando la nuova distanza d1(h, j) = l1(h, x) + d1 (x, j). Chi riceve il messaggio, supponiamo il *router* z, ha due comportamenti: se z non possiede alcun'informazione circa la raggiungibilità della destinazione j o, se al contrario, non solo possiede informazioni su j ma è stato in grado di selezionare un vicino per raggiungere x risponde alla *query* immediatamente con la propria tavola di *routing*. In caso le condizioni precedenti non siano vere anche z invia una *query* ai propri vicini passando in modo *Diffusion*.

Un *router* ritorna in modo normale non appena riceve una risposta ad una sua *query*, preoccupandosi di propagare la stessa a tutti quei vicini da cui aveva eventualmente ricevuto richieste. A quel punto la tavola è tolta dalla quarantena ed aggiornata con le nuove informazioni acquisite.

E-IGRP non è semplice. Tuttavia, esso rappresenta, a ragion veduta, lo stato dell'arte per ciò che riguarda la classe d'algoritmi *vector distance*. Semplificando, il suo obiettivo è quello di minimizzarne gli effetti negativi ricorrendo in caso d'insuccesso ad una forma di elaborazione distribuita.

Autonomous System: IGP - OSPF

OSPF (***O**pen **S**hortest **P**ath **F**irst*) adopera pacchetti di servizio specifici di tipo IP ed è definito dall'IETF con la RFC 1247 (1991) e la RFC 1583 (1994).



Contrariamente ai protocolli *distance-vector*, OSFP è basato su una differente tecnica detta *link state* che, in media, richiede minore tempo per convergere ad una soluzione stabile. Semplicemente ciascun *router* R deve:

- Scoprire l'indirizzo di ogni *router* vicino Ri.

- Misurare il costo necessario per raggiungere ciascun Ri usando una funzione idonea allo scopo.
- Inviare un messaggio a tutti i *router* diffondendo le informazioni acquisite.
- Calcolare, in locale, il cammino minimo verso ogni altro *router* utilizzando un algoritmo sviluppato da Dijkstra.

La maggiore differenza rispetto agli algoritmi *vector distance* è che gli algoritmi *link state* richiedono che ciascun *router* sia informato circa la completa topologia e i ritardi presenti nella rete.

Sulla base di queste informazioni ogni *router* calcola in locale il cammino minimo verso ogni destinazione conosciuta.

Scoperta dei vicini e misura dei costi di raggiungimento

Per conoscere l'intera topologia di una rete sono necessari alcuni accorgimenti: in primo luogo conoscere da ciascun punto l'insieme di *router* direttamente connessi. Per far questo non appena un *router* è inserito, invia un pacchetto di *HELLO* su tutte le connessioni *point-to-point* disponibili e riceve come risposta dai *router* direttamente connessi il relativo identificativo. Per calcolare il costo di connessione con i *router* vicini, è utilizzato un pacchetto *ECHO* misurando il tempo necessario ad ottenere la relativa risposta.

Diffusione delle informazioni ed algoritmo di *flooding*

Una volta che le informazioni necessarie allo scambio sono state collezionate, ciascun *router* costruisce un pacchetto contenente l'identità di chi invia, un numero di serie e la lista dei vicini (ciascuno con la stima di costo associata). L'algoritmo di diffusione serve a fare conoscere ad ogni ***gateway*** la completa topologia di rete a cui si è stati connessi. Esso si basa su di una tecnica di **flooding** in cui il numero di serie è usato per verificare la consistenza delle informazioni. Supponiamo che un pacchetto Psr (s indica il numero di serie ed r l'origine) giunga su di un *router* R:

- Se il pacchetto non è mai stato acquisito o se, al contrario, già esiste un pacchetto Ptr (con numero di serie t e con origine r) tale che s > t allora Psr viene forwardato su tutte le connessioni *point-to-point* con l'eccezione di quella da cui è arrivato.
- Se s = t il pacchetto è duplicato e si procede al suo scarto.
- Se il numero di serie del pacchetto s è inferiore di t si procede egualmente allo scarto per obsolescenza.

In base a questo semplice sistema, dopo un certo numero di iterazioni, ogni *gateway* conosce la topologia di rete cui è stato connesso.

Autonomous System: IGP - OSPF cammino minimo con algoritmo di Dijkstra

Non appena tutti i *router* hanno acquisito la topologia della rete, possono costruire un grafo pesato G che rappresenta le connessioni: ciascun link fisico è rappresentato da una coppia d'archi di direzione opposta e con pesi eventualmente differenti. Su G applicano l'algoritmo per il calcolo dei cammini minimi su grafo sviluppato da Dijkstra. Riportiamo di seguito i passi principali dell'algoritmo:

Dato l'insieme di tutti i nodi nella rete (i *router* che utilizzano OSPF) si definisca E l'insieme dei nodi già considerati ed R l'insieme di quelli rimanenti. Poniamo inizialmente E={r} (con r nodo rappresentante il *router* locale). Sia O l'insieme dei cammini uscenti da r; poniamo inizialmente O pari all'insieme dei cammini uscenti da r di lunghezza uno. Si ordini per metrica crescente O. Se O è vuoto o contiene soltanto cammini con metrica infinita, allora si possono marcare tutti i nodi in R

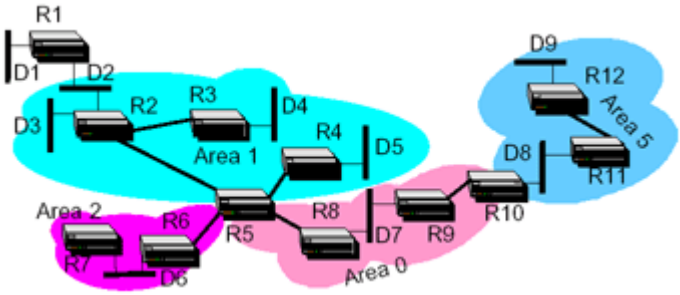
come irraggiungibili. L'algoritmo termina.

Si esamini il cammino di minore lunghezza P contenuto in O e lo si rimuova da questo insieme. Sia v l'ultimo nodo di P. Se v è già in E si ritorna al passo precedente altrimenti si è certi che P sia il più breve cammino da s ad v. Si sposta, quindi, v dall'insieme R all'insieme E. Si costruisca un nuovo insieme NP di j cammini candidati, collegando a P ciascun nodo fra gli nj adiacenti al nodo v. Il costo associato è pari al costo di P sommato al costo di nj. L'insieme NP è unito all'insieme O mantenendo l'ordinamento per costi crescenti. Si continua dal secondo passo.

L'algoritmo appena descritto si chiama *Shortest Path* (da cui il nome *Open Shortest Path First*) perché costruisce in modo incrementale l'insieme dei cammini minimi. Ad ogni passo si prova a verificare se sono soddisfatte le condizioni di *Bellman* (vedi riquadro). L'ordinamento dei cammini è realizzato mantenendo i nodi in una coda di priorità. Pur senza approfondire il perché, può essere utile citare il fatto che la complessità computazionale dell'algoritmo è pari ad O(MlogM) con M pari al numero di link di connessione contenuti nel *network* in esame. A titolo di paragone ricordo che i protocolli ***distance vector***, basandosi sull'algoritmo di *Bellman-Ford*, convergono in O(MN) dove N è il numero di *router* presenti nel *network*.

Autonomous System: IGP - gerarchia OSPF e pacchetti

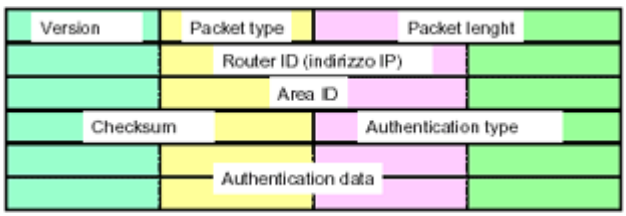
OSPF ha il concetto di gerarchia, dispone infatti del concetto di AS, il quale è ulteriormente suddiviso due livelli: *local area* (gruppo di reti contigue) e ***backbone*** (area particolare non necessariamente contigua).



Gli avvertimenti *Link-state* non lasciano rispettive aree. Segue un elenco dei termini comunemente menzionati nel contesto OSPF:

- *Backbone*: area di transito tra le altre aree
- *Backbone router*: *router* che è nel *backbone*
- *Area border router* (ABR): *backbone router* che si affaccia su più aree esegue una copia dell'algoritmo per ogni area
- *Internal router*: *router* che fa parte di un area diversa dal *backbone*.

Pacchetti OSPF



Version: specifica la versione del protocollo

Packet Type: identifica il tipo del pacchetto.

1. *Hello*
2. *Database Description*
3. *Link state Request*
4. *Link State Update*
5. *Link State Acknowledgement*

router-id:
un numero su 32 bit che identifica univocamente il *router* che ha generato il pacchetto

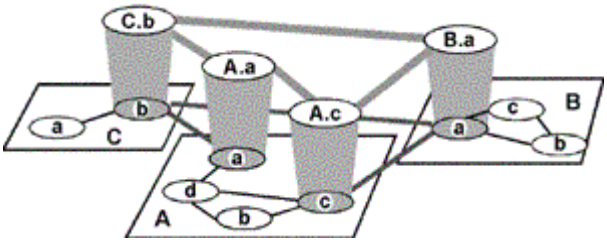
Area-id:
un numero su 32 bit che identifica a quale area appartiene il pacchetto

Authentication Type:
0 nessuna *authentication*
1 password in chiaro
altri valori riservati per usi futuri

Autonomous System: EGP - protocolli inter-dominio

Internet consiste di Sistemi Autonomi (***Autonomous Systems* AS**) interconnessi fra loro:

- *Stub* AS: piccolo
- *Multihomed* AS: grande (no *transit*)
- *Transit* AS: *provider*



BGP (*Border Gateway Protocol*) è un protocollo di *routing* tra domini, correntemente utilizzato sul *backbone* di Internet ed è in pratica il successore del protocollo **EGP** (*Exterior Gateway Protocol*).

In effetti questo protocollo viene usato soprattutto su Internet dove diversi AS sono collegati a questa grande rete attraverso strutture chiamate *Internet Service Provider* (ISP).

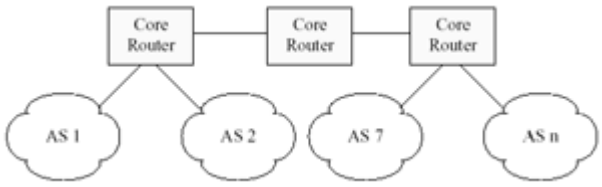
Il protocollo BGP costruisce un grafo di *autonomous system* basato sulle informazioni che si scambiano i *router*: questo grafo viene chiamato anche albero in cui ciascun AS viene identificato con un numero univoco. La connessione tra due AS si chiama percorso e una collezione di percorsi forma a sua volta un percorso che viene utilizzato per raggiungere la destinazione.

Autonomous System: EGP - exterior gateway protocol

L'***E**xterior **G**ateway **P**rotocol* è il primo **EGP** ad essere stato ampiamente utilizzato all'interno della rete Internet. Specificato con RFC 904 nell'aprile 1984 è oggi ampiamente disponibile su tutti i *router*, anche se è ormai considerato un protocollo obsoleto e Internet lo sta sostituendo con il BGP.

EGP è simile ad un algoritmo *distance vector*, ma invece del concetto di costo specifica solo se la

destinazione è raggiungibile oppure no. Questo ne impedisce il funzionamento su topologie magliate.
Esiste il concetto di un *core system* formato da una interconnessione di *core router*.



EGP genera dei pacchetti di *routing update* che contengono informazioni di *network reachability*, cioè annunciano che certe reti sono raggiungibili attraverso certi *router*. I pacchetti di *routing update* sono inviati ai *router* vicini ad intervalli di tempo regolari e raggiungono tutti i *router* EGP. L'informazione in essi contenuta è utilizzata per costruire le tabelle di instradamento.

I limiti di EGP sono molti e gravi: EGP non ha una metrica associata alle linee e quindi basa le sue decisioni esclusivamente sulla raggiungibilità; EGP non ammette la presenza di magliature nella topologia, e tutti gli AS devono essere collegati in modo stellare ad un *core system*; i pacchetti di *routing* update possono essere molto grandi.

Autonomous System: EGP - la famiglia BGP

Il ***B**order **G**ateway **P**rotocol* (**BGP**) è un *exterior gateway protocol* pensato per rimpiazzare il protocollo EGP ormai obsoleto. Il BGP è specificato per la prima volta dal RFC 1105 nel 1988, rispecificato come BGP-2 nel RFC 1163 nel 1990 e rispecificato ancora come BGPv3 nel RFC 1267 del 1991.

I *router* BGP comunicano tra loro utilizzando un livello di trasporto affidabile. Il BGP è un algoritmo di tipo *distance vector*, ma invece di trasmettere il costo di una destinazione, trasmette la sequenza di *autonomous system* da attraversare per raggiungere quella destinazione. Ogni *router* calcola il suo instradamento preferito verso una data destinazione e lo comunica ai *router* BGP adiacenti tramite un *distance vector*. La politica con cui tale calcolo avviene è configurabile su ogni *router* BGP.

Autonomous System: EGP - BGPv4 Funzionamento di base

Due sistemi danno luogo ad una connessione TCP tra di loro, dopodiché si scambiano messaggi per aprire e confermare le modalità di connessione.

All'inizio inviano entrambi nel flusso dei dati la loro intera tabella di instradamento. Man mano che si presentano variazioni dinamiche della tabella, inviano aggiornamenti al *peer*. BGP non richiede periodici invii dell'intera tabella, quindi è necessario che ogni dispositivo che comunica mediante BGP gestisca e mantenga in memoria tutta la tabella di ognuno dei suoi *peer*.

Vengono comunque impiegati dei sistemi di *timeout* propri di BGP con dei messaggi di tipo *KeepAlive* per assicurare il mantenimento della connessione. Inoltre vi sono degli altri tipi di notifica nel caso vi siano condizioni di errore o altri particolari eventi.

È importante sottolineare che secondo l'RFC del protocollo, non è detto che entrambi i capi della connessione debbano per forza essere *router*. È infatti possibile che uno dei due sia un *host* che voglia solo controllare o gestire la propria tabella in maniera dinamica, o magari che faccia da punto di unione tra un segmento di rete di un AS ed il *router* perimetrale di un altro AS, senza per questo fungere da *gateway* del suo sistema autonomo.

a) Le *route*

Una *route* viene definita come una coppia che abbini ad una destinazione gli attributi del percorso per quell'indirizzo. Queste vengono emanate come messaggi BGP di tipo *UPDATE*: la destinazione è quel o quei sistemi i cui indirizzi IP sono riportati nell'NLRI o *Network Layer Reachability Information*, ed il percorso è l'informazione riportata nei campi di attributi della *route* che vengono riportati all'interno dello stesso *UPDATE*.

Le *route* vengono inserite e mantenute all'interno del RIB o *Routing Information Bases*. BGP permette anche di poter cancellare una *route* precedentemente immessa in un RIB remoto mediante 3 modi differenti:

- Inserendo il prefisso IP della destinazione nel campo *WITHDRAWN ROUTES* di un messaggio *UPDATE*;
- emanando una *route* alternativa che contenga lo stesso campo NLRI;
- terminando la connessione TCP, fatto che comporta la rimozione di tutte le *route* che i due *peer* avevano emanato l'un l'altro.

b) *Routing Information Bases* (RIB)

Questo consta di tre parti:

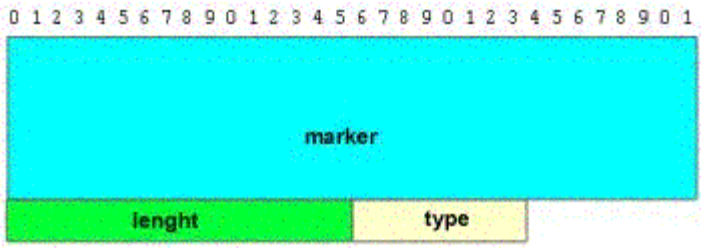
- *Adj-RIBs-In*: contiene le *route* ricevute mediante messaggi remoti di *UPDATE*. Queste verranno prese in considerazione dal processo decisionale di BGP.
- Loc-RIB: contiene le *route* presenti localmente che il processo decisionale BGP ha scelto tra quelle contenute in *Adj-RIBs-In*.
- *Adj-RIBs-Out*: contiene le *route* che il processo ha selezionato per essere emanate ai propri *peer*.

In pratica, *Adj-RIBs-In* contiene informazioni ricevute mediante *UPDATE* e non ancora processate, Loc-RIB quelle scelte come facenti parte della tabella di *routing*, mentre *Adj-RIBs-Out* quelle da inviare mediante propri messaggi di *UPDATE*.

Autonomous System: EGP - BGPv4 formato dei messaggi

Vediamo ora gli *header* BGP ed i tipi di messaggio che i *router* scambiano tra di loro per iniziare una connessione BGP e mantenere il loro RIB. I messaggi possono essere lunghi fino ad un massimo di 4096 ottetti e non più corti di un *header* BGP, o 19 ottetti.

Nella figura seguente viene mostrato l'*header* BGP:



Il *marker* contiene un valore di 128 bit che può essere predetto dal *peer*. Infatti nel caso di un messaggio di tipo *OPEN*, o se un messaggio *OPEN* non contiene alcuna informazione relativa ai meccanismi di autenticazione, allora sarà composto da soli 1. Altrimenti deve contenere un valore computabile secondo gli algoritmi di autenticazione usati. Questo viene usato non solo per

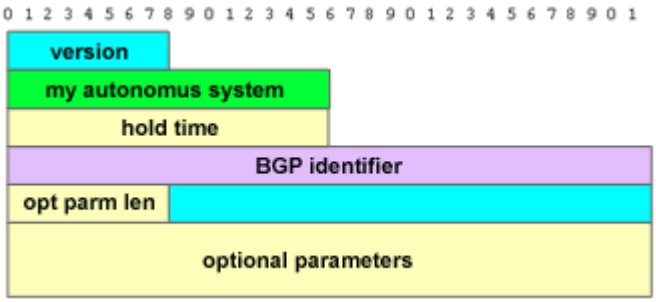
autenticare i messaggi in arrivo, ma anche per stabilire stati di mancata sincronizzazione tra i *peer* BGP.

La lunghezza (*length*) indica mediante un *unsigned* int la lunghezza totale del messaggio in *byte*, incluso l'*header* (minimo 19, massimo 4096, non deve contenere *padding*). Il tipo (*type*) di messaggio può essere uguale a:

- *OPEN*.
- *UPDATE*.
- *NOTIFICATION*.
- *KEEPALIVE*.

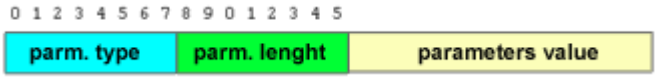
[1] Messaggio di tipo *OPEN*

Dopo una connessione TCP, il primo messaggio inviato da entrambi i *peer* è un *OPEN*. Nel caso sia accettabile, viene inviato un *KEEPALIVE* di conferma. Dopo che la sequenza *OPEN* è stata portata a termine, sarà possibile utilizzare i messaggi di tipo *UPDATE, NOTIFICATION* e *KEEPALIVE*. Oltre all'*header* BGP, sono presenti anche i seguenti dati:



La versione (*version*) è un *unsigned* int che identifica la versione del protocollo. Ora è uguale a 4. Dopo, un *unsigned* int di 16 bit contiene il valore dell' AS (*my autonomos system*). *Hold Time* contiene il valore in secondi entro cui bisogna inviare un messaggio *UPDATE* o *KEEPALIVE*, pena la caduta della connessione BGP. Impostando 0, non ci saranno *timeout*. L'RFC specifica un valore di almeno 3 secondi come base del valore al di fuori di 0.

L'identificativo BGP (BGP *identifier*) è in pratica un indirizzo IP associato ad una interfaccia utilizzata per l'invio dei messaggi. Ad esempio *Cisco* calcola questo valore usando il numero IP maggiore associato al *router*. Opt Parm Len indica la lunghezza in *byte* degli eventuali Parametri Opzionali. I parametri opzionali (*optionals parameters*) vengono rappresentati da una tripletta che contiene:



il tipo di parametro, la sua lunghezza ed il suo valore.

L'RFC di BGP4 specifica un solo parametro opzionale:

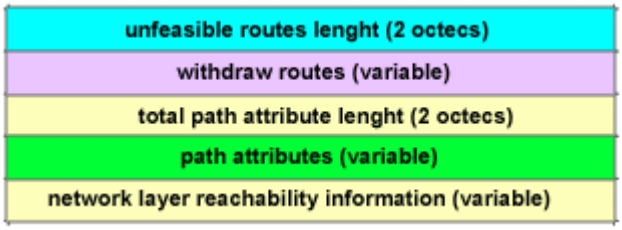
a) *Authentication Information* (*Parameter Type* 1)
Il campo *value* del parametro opzionale in questo caso contiene un codice di autenticazione seguito da un campo variabile contenente i dati necessari alla autenticazione:

È stato anche proposto l'uso di un'opzione TCP come la MD5 *Signature Option* per la gestione sicura dei segmenti TCP nel corso di sessioni BGP per poter ovviare a problemi di *hijack* e *man-in-the-middle*.

Il messaggio di tipo *OPEN* deve essere lungo almeno 29 *byte* compreso l'*header* del messaggio.

[2] Messaggio di tipo *UPDATE*

Questo messaggio viene usato per emanare o ritirare una singola *route* dal servizio. Può farlo anche insieme specificando differenti *route*. Contiene sempre l'*header* BGP e può anche contenere i seguenti campi opzionali:



I primi due ottetti contengono la lunghezza totale in *byte* del campo successivo e deve permettere di calcolare la lunghezza del NLRI [vedi sotto]. Un valore uguale a 0 indica che nessuna *route* viene ritirata e che quindi il campo *WITHDRAWN ROUTES* non è presente nel messaggio *UPDATE*.

Il campo successivo contiene una lista di prefissi IP codati come una coppia <lunghezza, prefisso> secondo questo schema:

Length (1 *octet*) - *Prefix* (*variable*)

la lunghezza indica il valore in bit del prefisso IP. Se uguale a 0 specifica tutti gli indirizzi IP. Ad esempio <24, 192.168.1.3> sta per 192.168.1.0/24 secondo la notazione CIDR, o *Classless Internet Domain Routing*, il prefisso invece l'indirizzo IP da associare ai bit di mascheramento.

Segue il campo di lunghezza totale degli attributi di percorso che indica in *byte* la lunghezza del campo che lo segue. Deve permettere di calcolare anche il NLRI. Un valore uguale a 0 indica che non c'è NLRI.

Gli attributi di percorso sono presenti in ogni messaggio di tipo *UPDATE*. Ogni attributo consta di una tripletta <tipo, lunghezza, valore>.

Il tipo consta di due ottetti, uno per delle *flag*, e l'altro per il codice vero e proprio:

Attr. *Flags* (1 *octect*) - Attr. *Type Code* (1 *octect*)

Il primo bit delle *flag* è il bit delle opzioni. Se uguale a 1 il parametro è opzionale, se uguale a 0 è invece riconosciuto. Il secondo bit identifica invece se un parametro debba essere propagato o meno, ovvero se sia transitivo, con valore uguale a 1, o non transitivo, con valore uguale a 0. Se il primo è uguale a 0, il secondo è sempre uguale a 1. Ovvero i parametri riconosciuti sono sempre

transitivi. Il terzo bit specifica se i parametri opzionali e transitivi (11) sia parziale, se uguale a 1, o totale, se uguale a 0. Il quarto specifica se la lunghezza degli attributi sia di un solo ottetto, uguale a 0, o di due, se uguale a 1. Questo quarto bit può essere usato solo in caso di lunghezza superiore a 255 *byte*. Gli ultimi 4 bit non sono utilizzati e devono essere settati a 0. Se il 4 bit è uguale a 0, allora la lunghezza sarà specificata nel terzo ottetto della tripletta degli attributi; nel quarto se invece sarà uguale a 1.

I codici di attributo sono:

ORIGIN (*Type Code* 1) Contiene un valore per identificare la origine del'attributo di percorso.

0. ottenuto mediante IGP;
1. ottenuto mediante EGP;
2. INCOMPLETO - ottenuto altrimenti.

AS_PATH (*Type Code* 2) contiene triplette per indicare le sequenze di percorso in segmenti della rete di AS. La tripletta prende la forma <tipo, lunghezza, valore>. Il tipo è un *long* con questi valori:

1. *AS_SET* : set disordinato di AS.
2. *AS_SEQUENCE* : set ordinato.

La lunghezza indica il numero di AS, mentre il valore contiene i numeri di AS percorsi.

NEXT_HOP (*Type Code* 3) Indica il prossimo *hop* da usarsi per arrivare alle destinazioni listate nell'NLRI.

MULTI_EXIT_DISC (*Type Code* 4) Opzionale non transitivo serve per il processo decisionale BGP per scegliere tra diversi punti di uscita verso il vicino AS.

LOCAL_PREF (*Type Code* 5) Serve per far conoscere il grado di preferenza del *router* mittente per una certa *route*.

ATOMIC_AGGREGATE (*Type Code* 6) Di lunghezza 0, serve per informare i *peer* che il sistema locale ha scelto una *route* non specifica, o meno, scartandone una più specifica contenuta in essa.

AGGREGATOR (*Type Code* 7) Opzionale transitivo, contiene il numero di AS che ha aggregato una *route*, seguito dal numero di *router* che ha aggregato.

COMMUNITY (*Type Code* 8) Implementato da *Cisco*, permette di definire la propagazione delle *route* al di fuori di comunità virtuali tra AS o *router*.

ORIGIN ID (*Type Code* 9) Implementato da *Cisco*, serve per identificare il *router* originante di una *route* dopo una sua riflessione in un AS per evitare doppioni nel caso venisse riflessa anche all'origine.

CLUSTER LIST (*Type Code* 10) Implementato da *Cisco*, contiene una lista di ID attraversati durante la riflessione di una *route* al di fuori del *cluster* di *client*.

Dopo segue l'NLRI. La sua lunghezza viene calcolata sulla base della lunghezza degli altri campi del messaggio *UPDATE*, secondo questo algoritmo:

Lunghezza del messaggio *UPDATE* - 23 - *Path Attributes* - *Withdrawn Routes* dove 23 è la lunghezza dell'*header* BGP, e dei due campi che contengono la lunghezza degli attributi e dei

WITHDRAWN.

Le informazioni sulle destinazioni sono contenute mediante coppie di valori <lunghezza, prefisso>:

Length (1 *octet*) - *Prefix* (*variable*)

questi due valori sono gli stessi utilizzati anche per rimuovere le *route*, come visto in precedenza. La lunghezza minima è di 23 *byte* come visto prima, 19 per l'*header* BGP, 2 per *Unfeasible Routes Length* e 2 per *Total Path Attribute Length*.

Ogni messaggio *UPDATE* può al massimo emanare una sola *route*, come riportato nel campo dell' NLRI, e può descriverla con una serie di attributi. Può invece ritirare più di una *route* con il campo *Withdrawn Routes*.

[3] Messaggio di tipo *KEEPALIVE*

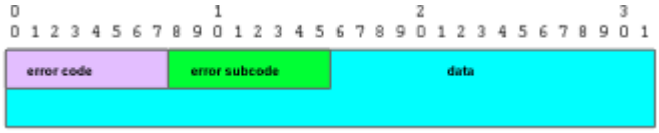
BGP non usa il sistema di *timeout* e *retransmission* proprio di TCP per gestire e controllare la propria connessione tra *peer*, ma utilizza questo messaggio per valutare lo stato delle *route* emanate da ogni *peer*, prima che il livello di trasporto rilevi dei problemi di rete.

I messaggi di tipo *KEEPALIVE* non devono essere inviati più velocemente di uno per secondo, ma devono comunque raggiungere il *peer* prima che il suo *Hold Timer* raggiunga la fine. Un valore ragionevole è un terzo del *Timer* remoto. Se il valore è stato negoziato uguale a 0, allora i messaggi non devono essere inviati.

Questo messaggio consiste del solo *header* BGP di 19 *byte*.

[4] Messaggio di tipo *NOTIFICATION*

Questo messaggio viene inviato appena venga rilevata una condizione di errore. Dopodiché la connessione BGP viene conclusa. Oltre all'*header* BGP, questo messaggio contiene i seguenti dati:



il codice di errore è un *unsigned* int che può contenere i seguenti valori:

1. *Message Header Error*.
2. *OPEN Message Error*.
3. *UPDATE Message Error*.
4. *Hold Timer Expired*.
5. *Finite State Machine Error*.
6. *Cease*.

il sottocodice contiene dei valori associati ad ognuno dei codici per meglio definire il tipo di errore [come succede nel caso dei pacchetti ICMP]. Se non contiene alcun valore deve essere settato a 0.

Ecco i sottocodici assegnati dall'RFC:

Message Header Error subcodes:

1. *Connection Not Synchronized*.
2. *Bad Message Length*.
3. *Bad Message Type*.

OPEN Message Error subcodes:

1. *Unsupported Version Number*.
2. *Bad peer* AS.
3. *Bad BGP Identifier*.
4. *Unsupported Optional Parameter*.
5. *Authentication Failure*.
6. *Unacceptable Hold Time*.

UPDATE Message Error subcodes:

1. *Malformed Attribute List*.
2. *Unrecognized Well-known Attribute*.
3. *Missing Well-known Attribute*.
4. *Attribute Flags Error*.
5. *Attribute Length Error*.
6. *Invalid ORIGIN Attribute*
7. *AS routing Loop*.
8. *Invalid NEXT_HOP Attribute*.
9. *Optional Attribute Error*.
10. *Invalid Network Field*.
11. *Malformed AS_PATH*.

il campo dati invece contiene elementi necessari per diagnosticare l'errore. Dipende strettamente dal codice e dal sottocodice di errore. Per maggiori informazioni controllare l'RFC di BGP4.

Autonomous System: EGP - Negoziazioni e UPDATE

Negoziazioni

I dispositivi paritari BGP dispongono della possibilità di negoziare durante il messaggio *OPEN* la versione d BGP supportata da entrambi. Nel caso di una versione non supportata, ogni *router* avrà la versione richiesta dal *peer* con l'*OPEN*, quella da lui tentata, quelle ricevute dal *peer* mediante un *NOTIFICATION* e quelle disponibili localmente. In questo modo saranno in grado di scegliere una versione comune per la connessione BGP. D'altra parte è ovvio che future versioni di BGP, per poter supportare la negoziazione della versione, dovranno mantenere l'uso e la conformazione dei messaggi *OPEN* e *NOTIFICATION*.

Elaborazione dei Messaggi *UPDATE*

Dopo che la connessione TCP sia stata effettuata, i messaggi *OPEN* e *KEEPALIVE* scambiati, il *timer* inizializzato, è possibile ricevere messaggi di tipo *UPDATE*.

Se il messaggio contiene un campo *WITHDRAWN ROUTES* non vuoto, allora le *route* precedentemente emanate con quei prefissi IP saranno eliminate da *Adj-RIB-In*. Dopodiché il processo decisionale penserà a cancellare la *route* dal *Local-RIB* o a sostituirle con altre possibili.

Se il messaggio contiene una *route* accettabile allora succederà questo:

1. se la nuova è identica ad una già presente in *Adj-RIB-In*, allora quella vecchia verrà sostituita e quindi cancellata, costringendo il processo BGP a sostituirla in *Local-RIB*.
2. se la nuova *route* fa parte di una precedente contenuta in *Adj-RIB-In* allora il processo decisionale farà prevalere la nuova in quanto più specifica di quella precedente
3. se la nuova ha uguali attributi ma è più specifica allora non servono altre azioni
4. se la nuova ha un NLRI non presente nelle vecchie *route* verrà inserita subito nel *Adj-RIB-In*
5. se la nuova è una meno specifica di una precedente allora il processo decisionale valuterà solo il set di IP raggiungibili con la nuova *route*.

Come funziona questo processo decisionale? Esso è suddiviso in tre fasi distinte. Nella prima BGP deve calcolare un livello di preferenza per ogni *route* disponibile in *Adj-RIB-In*. Per *route* interne all'AS spesso il solo parametro *LOCAL_PREF* verrà usato, altrimenti i parametri di percorso saranno valutati secondo il PIB o *Policy Information Base*, i cui dati sono a cura di ogni AS.

La seconda fase serve per scegliere la *route* considerata più efficiente tra quelle disponibili per ogni destinazione. Dopo averla scelta, per miglior livello di preferenza, od in quanto nuova ed unica *route* per una destinazione, la porrà in *Local-RIB*. Esistono comunque delle regole specifiche nel caso diverse *route* siano in parità per quanto riguarda la preferenza.

Nella terza fase BGP si occupa di valutare *Local-RIB* per poter ottimizzare secondo le *policy* dell'AS il *Adj-RIB-Out*. Nel momento in cui la valutazione di *Local-RIB* ed il processo di *Adj-RIB-Out* siano completi, allora potrà avere luogo l'invio di messaggi *UPDATE*.