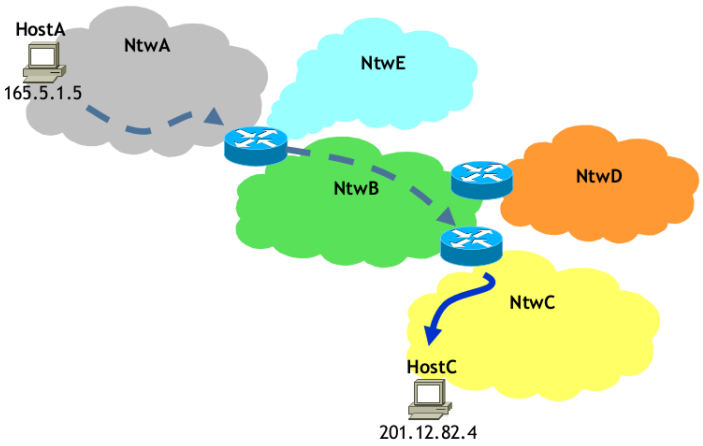
1 Livello di Rete

Nel modello TCP/IP, il livello di rete viene detto **livello internet** oppure *livello di internetworking* e si occupa della consegna *point-to-point* delle informazioni; in altre parole permette la trasmissione dei dati attraverso reti anche di diversa natura. A questo livello é necessario conoscere solamente l'indirizzo sorgente e destinazione degli host, in quanto tutta la semantica della comunicazione tra processi remoti ed eventuale controllo di flusso viene gestita dal livello superiore.

Sono necessari quindi dei meccanismi che permettano al livello di rete di scoprire la quali host sono connessi al proprio tratto di rete, in modo da sapere il percorso *"migliore"* per il **Routing** *(Instradamento)* dei dati, e un modo univoco di identificare tutte le interfacce di rete sulla stessa, per essere in grado di eseguire il **Forwarding** *(Inoltro)* correttamente alla destinazione.

Il protocollo utilizzato nella IPS che implementa tutte le funzionalitá appena descritte é chiamato **IP** *(Internet Protocol)*. Inoltre definisce lo standard per gli apparati di rete che hanno lo specifico compito di connettere le reti tra loro e di inoltrare il traffico in modo ottimale a seconda della destinazione del pacchetto, questi *relay system* vengono chiamati **Router** *(Commutatori)*.

Piú precisamente la PDU del livello di rete viene chiamata **Datagram** *(datagramma)*.



Come si vede dall'immagine soprastante, `HostA` vuole comunicare con `HostC` ma appartengono a reti diverse. In questo caso il livello di rete di `HostA` sa che tutti i dati con una destinazione diversa da `NtwA`, dovranno essere consegnati al router piú vicino, in modo che ci penserá lui ad inoltrarlo correttamente a `NtwC` (consegna *indiretta*). Ora il router deciderá quale percorso utilizzare per inoltrare correttamente il datagramma, utilizzando le tabelle di routing precedentemente calcolate, fino a raggiungere il router che *"conosce"* la rete del destinatario originale della comunicazione, `HostC`. L'ultimo passo é di consegnare direttamente i datagrammi a `HostC` (consegna *diretta*).

1.1 Riassumendo

Riassumendo, le funzioni fondamentali del livello di rete:

- **Routing:** calcolare il percorso *"migliore"* dalla sorgente alla destinazione utilizzando degli appositi algoritmi di Routing
- **Forwarding:** lo spostamento dei datagrammi da una linea in ingresso linea d'uscita
- **Indirizzamento:** l'identificazione in modo univoco di ogni host sullo stesso tratto di rete
- **Tunnelling:** incapsulamento ulteriore dei datagrammi per permettere il loro transito su determinati tipi di rete.

1.2 Internetworking

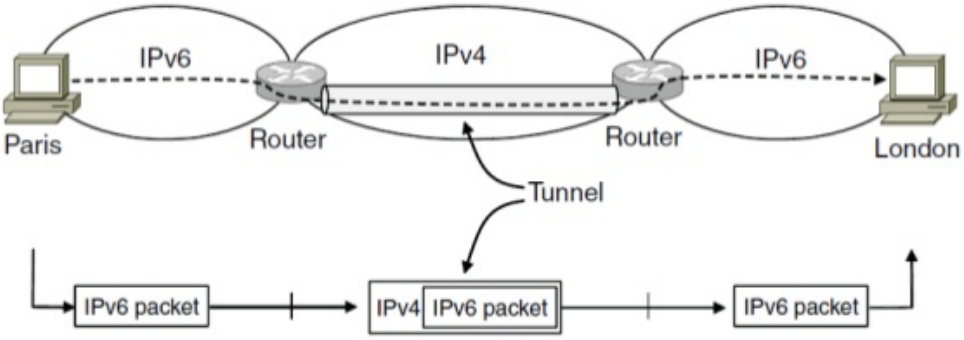
Di seguito verranno analizzati i problemi che sorgono quando si collegano tra loro due o piú reti che supportano protocolli e/o richiedono caratteristiche differenti.

Per connettere reti diverse le possibilitá sono due: si possono costruire sistemi che traducano i datagrammi per un dato tipo di rete oppure aggiungere un livello comune a tutte. La seconda soluzione é quella adottata nella IPS con il protocollo IP.

1.2.1 Tunneling

Nel caso in cui due host vogliano comunicare tra di loro, ma sono separati da un tratto di rete che non supporta gli stessi protocolli della rete del sorgente e della destinazione, il **Tunneling** *(fare da tunnel)* riesce ad incapsulare tutti i datagrammi della sorgente nel payload dei datagrammi che possono transitare su quel tratto di rete, quindi estrarli e ricomporli una volta arrivati dall'altro cado della rete.

L'esempio seguente si riferisce al tunneling dei datagrammi IPv6 attraverso una rete che supporta solo IPv4.



1.2.2 Frammentazione dei datagrammi

Ogni rete impone una dimensione massima ai datagrammi che possono circolare al suo interno. Questo puó dipendere da vari fattori come: hardware degli apparati di rete, sistemi operativi, standard internazionali ecc...

La tendenza é di trasmettere datagrammi di grandi dimensioni per ridurre l'overhead, ma é possibile che debbano transitare attraverso una rete che non li supporta, ovvero che la rete consente una **MTU** *(Maximum Transfer Unit)* minore della precedente. In questo caso il datagramma deve essere suddiviso in frammenti, inviati poi come un nuovi datagrammi a livello di rete.

Il problema principale nell'utilizzo di questo approccio é proprio corretta gestione dei frammenti da parte del router, quindi si possono attuale le seguenti scelte:

- frammentare del datagramma all'inizio della rete con MTU piú piccola e ricostruzione all'uscita: questo comporta un peggioramento delle prestazioni perché é possibile che si creino coni di bottiglia
- non ricomporre il datagramme e continuare a trasmettere i frammenti lungo il tratto di rete rimasto: in questo modo non vengono sprecate risorse per la ricomposizione ma, necessita che il protocollo in uso preveda la frammentazione tramite opportuni campi nell'header.

Per risolvere definitivamente il problema é necessario non frammentare i datagrammi alla sorgente, infatti IP prevede il protocollo **MTU Path Discovery** per scoprire la dimensione minima della MTU lungo il tratto di rete tra sorgente e destinazione. Il suo funzionamento é molto semplice: un host invia un datagramma impostato come non frammentabile alla destinazione, se un tratto di rete intermedio non supporta quella dimensione risponderá con un datagramma di errore contenente la sua massima MTU supportata.

Fermo restando che il percorso preso dai datagrammi non é necessariamente detto che rimanga tale per tutta la durata della trasmissione.

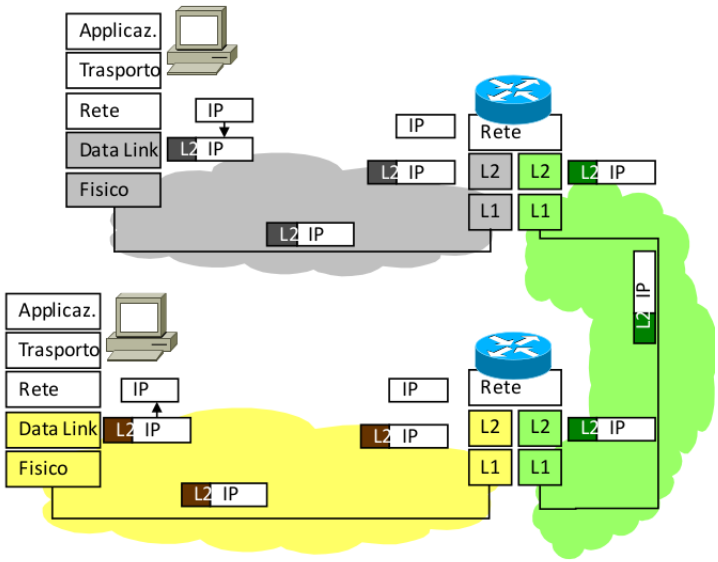
2 Router

I router, o *commutatori*, sono dei relay system che lavorano esclusivamente a livello di rete, non hanno bisogno dei livelli soprastanti che gestiscano connessioni e perdita di pacchetti, in quanto il loro compito é di permettere l'inoltro dei datagrammi attraverso sottoreti diverse. Per riuscire in questo é necessario che il router abbia:

- una interfaccia di rete per ogni sottorete a cui é collegato[1]
- la conoscenza dei suoi router *"vicini"*
- un algoritmo decisionale per l'inoltro dei datagrammi attraverso il percorso *"migliore"*

2.1 Commutazione di Pacchetto - Store and Forward

Tutti i router implementano la tecnica chiamata **Store and Forward** *(Salva e Inoltra)* che permette di salvare temporaneamente il pacchetto in memoria per controllare il checksum e l'indirizzo di destinazione, e successivamente decidere qual'é il prossimo **Hop** *(Salto)*, ovvero quale router, inoltrarlo.



In figura si nota come i router conoscano a quali sottoreti sono collegati e, in base all'indirizzo di rete associato all'indirizzo IP di destinazione e la maschera di sottorete, riescano a capire a quale router inoltrare il datagramma. Il router in alto conosce che per tutti i datagrammi in entrata dalla sottorete *grigia* verso la sottorete *gialla*, dovrá inoltrarli al router sulla sottorete *verde*, in quanto lui non ha conoscenza di come raggiungere direttamente la sottorete *gialla*.

[1] oppure una sola interfaccia divisa logicamente a livello software

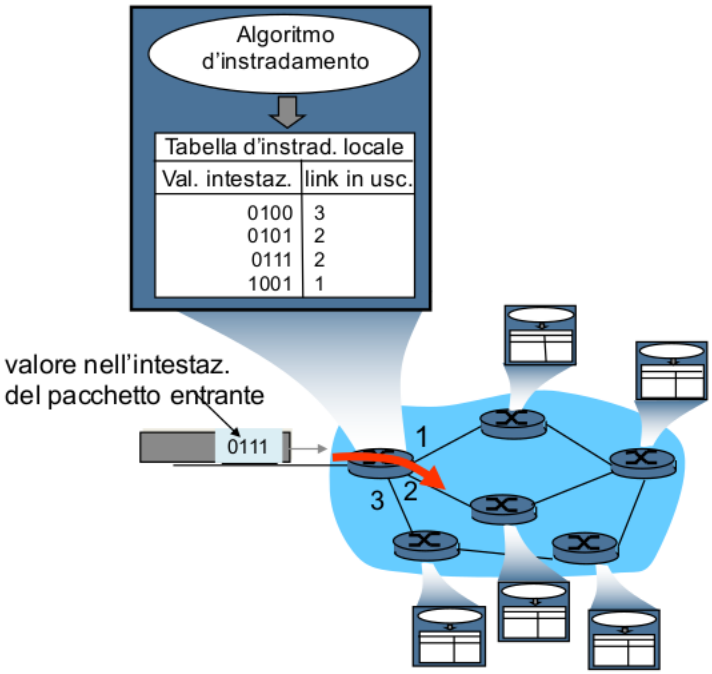
2.2 Instradamento - Routing

Il meccanismo di **routing** *(instradamento)* dei pacchetti, é necessario per non creare una connessione fisica tra sorgente e destinatario per ogni coppia di host in internet, in quanto permette al commutatore di decidere quale percorso é piú *"conveniente"* far percorrere ad un dato datagramma. A tale scopo si rende necessario la realizzazione di software adatto a scegliere il miglior percorso, o la migliore direzione, per minimizzare la latenza ed impedire la congestione degli apparati di rete.

Un **Algoritmo di Routing** é, quindi, quella parte del software di rete che si occupa capire a quali sottoreti é collegato il commutatore per poi prendere decisioni sull'inoltro dei datagrammi ingresso.

La parte cruciale del software é la capacitá di adattarsi ai cambiamenti della rete: congestione, perdita di un collegamento fisico oppure aggiunta di un nuovo commutatore. A fronte dei cambiamenti, ogni router deve mantenere aggiornato una tabella di "costi" per ogni sottorete a cui é collegato, dove per "costi" si intende una unitá di misura in base alla quale puó prendere una decisione sull'inoltro dei datagrammi. Questa tabella viene chiamata **Routing Table** *(Tabella di Inoltro)*.

É facile immaginare che non é pensabile impostare una tabella di routing statica, per ogni commutatore in internet, in quanto, per stessa definizione di internet, é una rete estremamente dinamica, con collegamenti che si aggiungono e si tolgono continuamente. Quindi i router devono adattarsi a questa caratteristica intrinseca della rete.



Come si vede dall'immagine ogni router ha calcolato la propria tabella di instradamento in funzione delle sottoreti a lui collegate. Nel momento in cui riceve un datagramma in ingresso, viene interrogata per sapere in quale link in uscita inoltrarlo.

2.3 Distribuzione delle Tabelle di Routing

Il problema dell'instradamento dei pacchetti si riduce essenzialmente a come diffondere le informazioni dei link locali dei router a tutti gli altri, in modo tale che ogni commutatore conosca ogni possibile destinazione. Queste problematiche vengono affrontate in vario modo dai protocolli e dagli algoritmi seguenti.

2.3.1 Distance Vector Routing

Il primo algoritmo é basato sul *Vettore delle Distanze* dove ogni router misura, il *"costo"* dei collegamenti tra lui e tutti i suoi vicini tramite il numero di **hop** *(salti)*, producendo cosí il vettore delle distanze. Ovviamente verrá mantenuta l'associazione `vicino-costo`. Per mettere a conoscenza gli altri router che attraverso di lui si possono raggiungere alcune destinazioni ad un certo costo, invia a tutti i suoi vicini il suo vettore delle distanze.

A questo punto, il procedimento avviene contemporaneamente per tutti i router collegati, quindi é necessario decidere come aggiornare il proprio vettore delle distanze in funzione di quello ricevuto. Quindi Possono accadere i seguenti casi:

- una destinazione `X` che prima era irraggiungibile (costo infinito) ora é raggiungibile tramite un vicino `Y` a costo `k`, quindi aggiorno il mio vettore nella posizione della destinazione `X` scrivendo `Y,k+n`, dove `n` é il costo per andare da `X` a `Y`
- una destinazione `X` é raggiungibile attraverso il vicino `Y` con un costo `k` inferiore a quello che ho attualmente, quindi aggiorno il mio vettore alla posizione della destinazione `X` scrivendo `Y,k+n`, dove `n` é il costo per andare da `X` a `Y`
- una destinazione `X` non é raggiungibile, oppure ha un costo superiore al mio attraverso il vicino `Y`, quindi mantengo la mia distanza

Ogni aggiornamento del vettore delle distanza locale al router provoca la sua trasmissione a tutti i vicini.

É ragionevole pensare che questo tipo di algoritmo non termini mai, ovvero aggiornamenti provocano ritrasmissioni, che provocano aggiornamenti ecc... Questo tipo di problema prende il nome di *count to infinity* ma fortunatamente accade solo in determinate circostanze e normalmente *converge* ad una situazione stabile anche se lentamente.

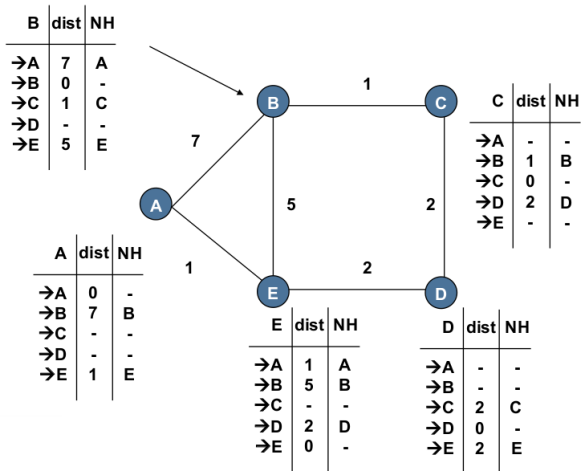
Ció nonostante é stato previsto che i pacchetti di aggiornamento vengano inviati con il campo `TTL` pari a 15, limitando il diametro massimo della rete.

Un'implementazione dell'algoritmo basato sul vettore delle distanze é **RIP** *(Routing Information Protocol)*.

Riassumendo

- ogni router ha una visione ristretta della topologia della rete
- non necessita di una nozione condivisa di distanza
- costruzione del percorso minimo partendo dalla destinazione e andando verso le sorgenti in modo additivo.

Esempio Di seguito viene mostrato un grafo i quali nodi sono i router e gli archi sono i collegamenti tra essi. La tabella accanto ai nodi raffigura la routing table ad ogni iterazione del protocollo ed é cosí composta: `X` é la colonna destinazione dove `X` é il nome del nodo, `dist` é la distanza, `NH` é il next hop.



B	dist	NH
→A	7	A
→B	0	-
→C	1	C
→D	-	-
→E	5	E

C	dist	NH
→A	-	-
→B	1	B
→C	0	-
→D	2	D
→E	-	-

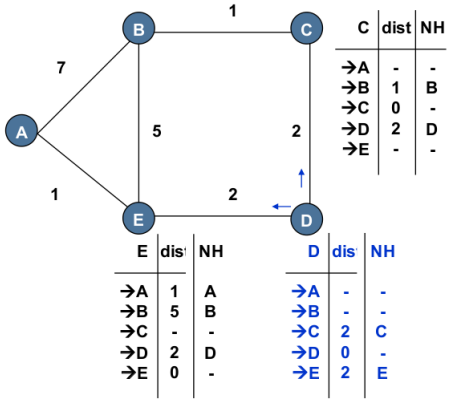
A	dist	NH
→A	0	-
→B	7	B
→C	-	-
→D	-	-
→E	1	E

E	dist	NH
→A	1	A
→B	5	B
→C	-	-
→D	2	D
→E	0	-

D	dist	NH
→A	-	-
→B	-	-
→C	2	C
→D	0	-
→E	2	E

I protocolli di tipo distance vector partono dalla seguente situazione: ogni router é a conoscenza solo dei propri vicini e della loro distanza.

Ovviamente verrá descritto il funzionamento per il nodo `D`, ma contemporaneamente avviene in tutti i nodi.

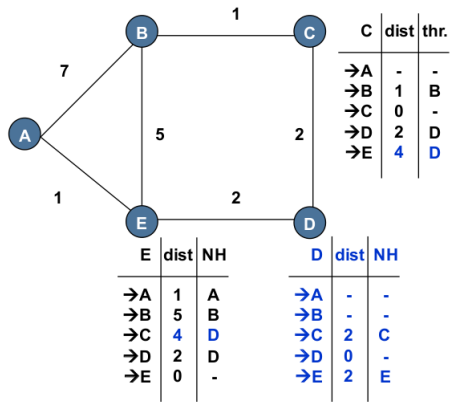


C	dist	NH
→A	-	-
→B	1	B
→C	0	-
→D	2	D
→E	-	-

E	dis	NH
→A	1	A
→B	5	B
→C	-	-
→D	2	D
→E	0	-

D	dis	NH
→A	-	-
→B	-	-
→C	2	C
→D	0	-
→E	2	E

Il router `D` una volta costruito il suo vettore delle distanze lo distribuisce ai suoi vicini `C` e `E`



C	dist	thr.
→A	-	-
→B	1	B
→C	0	-
→D	2	D
→E	4	D

E	dist	NH
→A	1	A
→B	5	B
→C	4	D
→D	2	D
→E	0	-

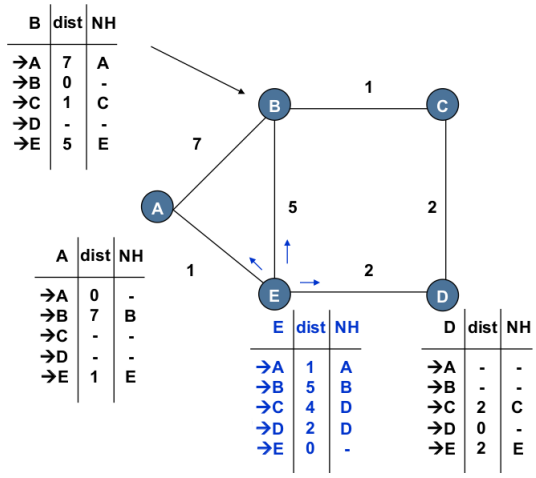
D	dist	NH
→A	-	-
→B	-	-
→C	2	C
→D	0	-
→E	2	E

Il router `C` riceve il vettore e nota che prima non riusciva a raggiungere `E` ma, tramite `D` ora riesce a raggiungerlo a costo

$$2\ (\texttt{C-D}) + 2\ (\texttt{D-E}) = 4$$

Al router `E` accade lo stesso evento del router `C`: riceve il vettore delle distanze da `D` e nota che ora riesce a raggiungere `C` tramite `D` a costo

$$2\ (\texttt{E-D}) + 2\ (\texttt{D-C}) = 4$$



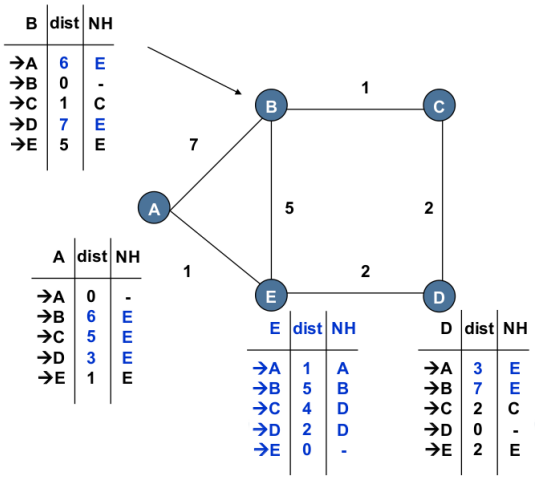
B	dist	NH
→A	7	A
→B	0	-
→C	1	C
→D	-	-
→E	5	E

A	dist	NH
→A	0	-
→B	7	B
→C	-	-
→D	-	-
→E	1	E

E	dist	NH
→A	1	A
→B	5	B
→C	4	D
→D	2	D
→E	0	-

D	dist	NH
→A	-	-
→B	-	-
→C	2	C
→D	0	-
→E	2	E

Ora concentriamo l'attenzione sul nodo `E`. Dopo aver ricevuto i vettore delle distanze ed aggiornato il suo lo invia ai suoi vicini `B`, `A` e `D`.



B	dist	NH
→A	6	E
→B	0	-
→C	1	C
→D	7	E
→E	5	E

A	dist	NH
→A	0	-
→B	6	E
→C	5	E
→D	3	E
→E	1	E

E	dist	NH
→A	1	A
→B	5	B
→C	4	D
→D	2	D
→E	0	-

D	dist	NH
→A	3	E
→B	7	E
→C	2	C
→D	0	-
→E	2	E

I router `B`, `A` e `D` ricevono il nuovo vettore delle distanze di `E` ed aggiornano le voci con solo i valori che portano ad un costo minore tramite il router `E`: il router `B` aggiorna `A` e `D` rispettivamente con

$$5\ (\texttt{B-E}) + 1\ (\texttt{E-A}) = 6$$
$$5\ (\texttt{B-E}) + 2\ (\texttt{E-D}) = 7$$

Il router `A` aggiorna `B`, `C` e `D` rispettivamente con

$$1\ (\texttt{A-E}) + 5\ (\texttt{E-B}) = 6$$
$$1\ (\texttt{A-E}) + 4\ (\texttt{E-C}) = 5$$
$$1\ (\texttt{A-E}) + 2\ (\texttt{E-D}) = 3$$

Ed infine il router `D` aggiorna `A` e `B` rispettivamente con

$$2\ (\texttt{D-E}) + 1\ (\texttt{E-A}) = 3$$
$$2\ (\texttt{D-E}) + 5\ (\texttt{E-B}) = 7$$

Ovviamente il processo continuerebbe fino alla convergenza del sistema ma si nota chiaramente che ci sono molti scambi di vettori che potrebbero essere evitati.

2.3.2 Link State Routing

Il secondo algoritmo usato per la distribuzione e la creazione delle tabelle di routing prende il nome di **Link Stat Routing** *(Instradamento sullo stato dei collegamenti).* Viene comunemente piú utilizzato di Distance Vector perché permette un tempo di convergenza minore e permette al router di avere una visione globale della topologia della rete.

I passi dell'algoritmo quindi sono (per ogni router):

- **scoprire l'indirizzo di rete dei propri vicini**: ogni router invia in broadcast un datagramma `ICMP Hello` e memorizzerá tutti gli indirizzi di rete delle risposte
- **misurazione del costo dei collegamenti**: sfruttando il punto precedente e attraverso una speciale funzione riesce a ricavare un costo per il collegamento verso ogni destinazione
- **costruzione dei pacchetti che contengono lo stato di collegamento**: dopo aver risposto ai datagrammi degli altri router vicini, crea un datagramma contenente il suo stato dei collegamenti
- **distribuzione lo stato dei collegamenti**: utilizzando il flooding distribuisce lo stato dei collegamenti [2]
- **calcolo dei percorsi**: una volta ricevute le informazioni sullo stato del collegamento dagli altri router, é possibile costruire un grafo della rete

Una volta terminato l'algoritmo, il router avrá creato un grafo che gli dará la visione globale della topologia della rete, quindi non piú limitata alla conoscenza del prossimo router per minimizzare il numero di salti per arrivare alla destinazione.

Per la creazione della vera tabella di routing viene quindi eseguito l'algoritmo di Dijkstra per la ricerca dei cammini minimi sul grafo della rete per ogni possibile destinazione.

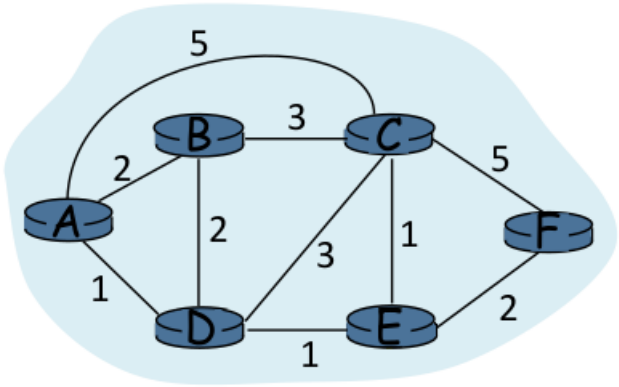
L'implementazione usata in Internet che utilizza il protocollo di tipo link state é **OSPF** *(Open Shortest Path First).*

Riassumendo

- ogni router ha la visione globale della topologia della rete
- necessita di una nozione condivisa di misurazione della distanza
- costruzione del percorso minimo attraverso delle informazioni locali

[2] per evitare la distribuzione infinita si utilizza un campo che viene usato come numero di sequenza; ogni router tiene traccia della coppia `sorgente-sequenza`, cosí se riceve un pacchetto con numero di sequenza inferiore lo scarta, altrimenti lo propaga flooding

Esempio Di seguito viene riporta la topologia di una rete di router con i relativi costi dei collegamenti. Questo grafo si puó supporre che sia memorizzato nel nodo `A` e di eseguire l'algoritmo di Dijkstra per il calcolo dei cammini minimi.



Nella tabella sottostante vengono proposti tutte le iterazioni dell'algoritmo. Sulla colonna `set(N)` vengono inseriti i nodi scelti a costo minimo per ogni iterazione, invece le restanti rappresentano ogni router nel grafo, senza la sorgente `A`, nella forma "`D(X),p(X)`" che significa *distanza da* `X` *attraverso il predecessore* `X`.

Allo Step 0 la tabella risulta popolata solo con i dati dei vicini del router `A`

Step	set(N)	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞

Ad ogni iterazione si considera la colonna con il costo minimo[3], in questo caso il nodo `D` che ha costo `1`. Quindi si crea una nuova riga dove si aggiunge al `set(N)` il nodo `D`.

Step	set(N)	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD	X	Y	Z	W	H

Ora, ogni volta che si aggiunge nuovo router al `set(N)` si devono calcolare le nuove distanze minime che sono raggiungibili dai router attualmente presenti nel `set(N)`: in questo caso al posto di `X, Y, Z, W` e `H` andranno calcolati:

- `X`: per arrivare a `B` posso scegliere:
  - `(A-B)` a costo `2` passando da `A`, quindi `2,A` (costo della riga sopra)
  - `(A-D-B)` a costo `1+2=3` passando da `D`, quindi `3,D`
- `Y`: per arrivare a `C` posso scegliere:
  - `(A-C)` a costo `5` passando da `A`, quindi `5,A` (costo della riga sopra)
  - `(A-D-C)` a costo `1+3=4` passando da `D`, quindi `4,D`

[3] in caso di paritá scelgo quella che mi aggrada di piú

- `Z`: questo elemento rappresenta la colonna del router `D`, che ho giá nel `set(N)`, quindi non faccio niente
- `W`: per arrivare a `E` posso scegliere:
  - attraverso `A` non riesco ad arrivare ad `E`, quindi ∞
  - `(A-D-E)` a costo `1+1=2` passando da `D`, quindi `2,D`
- `H`: per arrivare a `F` non ho nessuna scelta, ne attraverso `A` ne `D`, quindi ∞

Per ogni possibile scelta elencata sopra scelgo quella con costo minimo[3], quindi la riga dello Step 1 risulterá

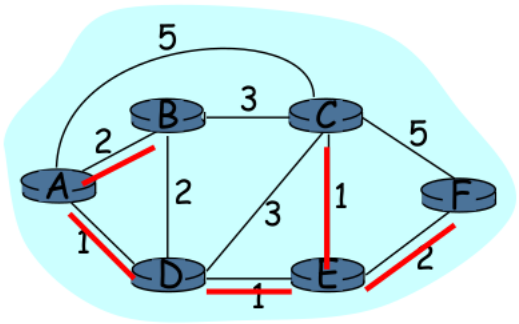
Step	set(N)	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD	2,A	4,D	==	2,D	∞

A questo punto si ricomincia tutto il procedimento appena illustrato con il router con costo minimo preso dalla riga appena creata, ovvero `E`.

Una volta raggiunti tutti i nodi la tabella risulterá

Step	set(N)	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD	2,A	4,D	==	2,D	∞
2	ADE	2,A	3,E	==	==	4,E
3	ADEB	==	3,E	==	==	4,E
4	ADEBC	==	==	==	==	4,E
5	ADEBEF	==	==	==	==	==

A questo punto quello che si é ottenuto é il *minimum spanning tree* del grafo di partenza, ovvero l'albero di copertura minimo, che poi verrá utilizzato per la creazione della tabella di routing del nodo `A`.



2.3.3 Flooding

Il **Flooding** *(Inondazione)* non rientra nelle categorie di algoritmi atti alla creazione o alla distribuzione delle tabelle di routing, ma é semplicemente una tecnica che distribuzione delle informazioni.

Un router che distribuisce delle informazioni in flooding non deve fare altro che riceverle da un'interfaccia di rete e replicarla su tutte le altre interfacce. Il nome deriva proprio da questa caratteristica.

Ovviamente si generano molti informazioni duplicate, che vengono gestiti tramite il campo `TTL` di IP oppure tramite numeri di sequenza opportunamente impostati.

Il vantaggio nell'utilizzo del flooding é la sua intrinseca semplicitá di implementazione e robustezza, sceglie sempre il percorso piú breve perché prova ogni possibile percorso in parallelo.

2.4 Instradamento Globale

Gli algoritmi di instradamento presentati finora avevano la caratteristica di memorizzare uno *"stato"* della rete, ovvero i collegamenti o l'intera topologia. In realtá non é mai possibile che i router odierni memorizzino milioni e milioni di informazioni relative a tutte le possibili sottoreti esistenti, oltretutto il traffico generato dalle informazioni di aggiornamento non lascerebbero capacitá di elaborazione del router per i dati utente.

Per questi motivi viene adottata una sistema di instradamento **gerarchico**, ovvero vengono raggruppati i router, tipicamente presenti su ogni territorio nazionale e gestiti da un ente governativo, per formare un **Autonomous System o AS** *(Sistema Autonomo)*, dove viene applicato OSPF come algoritmo di instradamento.

Questa organizzazione porta il vantaggio immediato che tutti i router all'interno dell'AS avranno delle tabelle di instradamento relativamente piccole, dovute al numero di sottoreti presenti, ma nasce il problema di come collegare vari AS tra di loro. Ogni sistema autonomo mette a disposizione dei router particolari, identificati come **Border Router** *(Commutatori di Confine)*, che hanno il compito specifico di instradare tutti il traffico in entrata e in uscita tra gli AS. Il collegamento tra border router viene chiamato **Backbone Area** *(Area di Dorsale)*.

Ovviamente non ha senso che nelle tabelle di instradamento dei border router compaiano tutte le possibili sottoreti presenti negli altri AS, sarebbe una contraddizione. In questo caso é necessario che i border router effettuino un instradamento di livello *"superiore"*, ovvero che non si basino piú sul numero di sottoreti da attraversare per raggiungere la destinazione, ma sul percorso tra AS.

Purtroppo le tabelle di instradamento che definiscono questo percorso non sono il risultato di un algoritmo per la ricerca del cammino minimo tra quest'ultimi, ma frutto di accordi economico-commerciali tra gli enti dei vari paesi che le gestiscono. In questo mondo puó succedere che datagrammi provenienti da AS *"amici"* prendano un percorso preferenziale rispetto a datagrammi di AS *"non-amici"*.

L'implementazione del protocollo usato dai border router é chiamato **BGP** *(Border Gateway Protocol)*.

3 Spazio di indirizzamento in IP

La parte riguardante lo spazio di indirizzamento, é la parte cruciale di tutto il sistema di Internet. Tutti gli host e i router devono usare uno schema di indirizzamento unico e uniforme.

Attualmente esiste lo standard **IP** con due versioni: **IPv4** e **IPv6**; entrambi completamente supportati in Internet ma il secondo é stato creato per rimpiazzare il primo in quanto fornisce opzioni piú avanzate e uno spazio di indirizzi maggiore.

3.1 Indirizzi IPv4

Gli indirizzi IPv4, o piú semplicemente IP, sono numeri a 32 bit assegnate ad ogni interfaccia di rete. Per una migliorane la comprensione e la gestione viene usata la notazione **Dotted Decimal**, ovvero l'indirizzo viene suddivisa in 4 gruppi da 8 bit ciascuno, ed espressi in forma decimale. Nella tabella seguente sono riportati degli esempi:

Numero binario a 32bit	Notazione Dotted Dicimal
`10000001 00110100 00000110 00000000`	`129.52.6.0`
`11000000 00000101 00110000 00000011`	`192.5.48.3`
`00001010 00000010 00000000 00100101`	`10.2.0.37`

Ma gli indirizzi IP non sono sufficienti per identificare univocamente un'interfaccia di rete. Si prenda in considerazione il seguente esempio: un azienda ha bisogno di connettere alla rete un gruppo di computer, per gli standard di internet é necessario che abbiamo tutti un indirizzo univoco, quindi sará necessario acquistare dall'ICANN un numero congruo di indirizzi. Questo esempio mette in luce che assegnare un indirizzo unico a livello mondiale ad un comune terminale aziendale é un problema sia per chi vuole creare la sua rete privata sia per il numero esiguo di indirizzi pubblici.

Per risolvere questo genere di problematiche si é deciso di utilizzare un nuovo numero a 32 bit chiamato **Sub-Net Mask** *(Maschera di Sottorete)* che, invece di identificare un host, identifica un intera rete, in modo da capire se un dato indirizzo IP appartiene o meno a quella sottorete.

In altre parole agisce come un *"raggruppatore"* di indirizzi, in modo da suddividere un solo IP pubblico in tante sottoreti diverse; questo porta il vantaggio che lo stesso indirizzo puó essere utilizzato in sottoreti differenti.

La suddivisione avviene tramite **AND logico bit-a-bit** tra un indirizzo IP e la sua maschera di sottorete.

Esempio: L'indirizzo `128.208.1.14` con la maschera di sottorete `255.255.255.0` indica:

`128.208.1.14`	indirizzo IP
`255.255.255.0`	maschera di sottorete
`128.208.1.0`	risultato dell'AND bit-a-bit tra indirizzo e maschera
`128.208.1.0 - 128.208.1.255`	possibili indirizzi utilizzabili dagli host

Esempio: L'indirizzo `128.208.1.14` con la maschera di sottorete `255.255.254.0` indica:

`128.208.1.14`	indirizzo IP
`255.255.254.0`	maschera di sottorete
`128.208.0.0`	risultato dell'AND bit-a-bit tra indirizzo e maschera
`128.208.0.0 - 128.208.1.255`	possibili indirizzi utilizzabili dagli host

Come si nota dagli esempi precedenti l'indirizzo `128.208.1.14` identifica due interfacce di rete diverse, il che va contro il concetto stesso di unicitá degli indirizzi. Ma grazie alle sottoreti é possibile riutilizzarlo, risparmiando una quantiatá considerevole di indirizzi pubblici.

La maschera di sottorete, quindi, divide la parte di indirizzo IP che identifica univocamente la rete, chiamato **NetID** *(Prefisso)*, dalla parte riservata agli host, chiamata **HostID** *(Suffisso)*. La NetID rimane uguale per tutti gli host di una data rete, per definizione; a livello globale invece, deve essere assegnata da un'autoritá in modo tale da non creare conflitti in fase di routing. Mentre per l'assegnamento degli HostID é lasciato ai progettisti della rete stessa.

É possibile scrivere la maschera di sottorete in dotted decimal, ma la notazione piú compatta e comoda risulta la **Slash Notation**, ovvero riportare quanti bit sono impostati a `1` dopo il carattere `"/"`. Alcuni esempi:

Notazione Dotted Dicimal	Slash Notation
`255.255.255.0`	`/24`
`255.255.254.0`	`/23`
`255.255.240.0`	`/20`

Esistono tuttavia degli indirizzi *"speciali"* che ogni sottorete possiede e non é possibile assegnarli agli host in quanto identificano:

- **Indirizzo di Rete:** ottenuto con l'AND logico bit-a-bit tra l'indirizzo IP e la maschera di sottorete
- **Indirizzo di Broadcast:** ultimo indirizzo ottenibile nella sottorete, usato per la trasmissione dei datagrammi a tutti gli host
- **Indirizzo di Loop-Back:** generalmente é `127.0.0.1`

Esempio: La scrittura `128.208.0.14/24` indica:

`128.208.0.14`	indirizzo dell'host
`255.255.255.0`	maschera di sottorete
`128.208.0.0`	indirizzo di rete
`128.208.0.255`	indirizzo di broadcast
`128.208.0.1 - 128.208.0.254`	spazio di indirizzamento per gli host

3.2 Indirizzi Pubblici e Privati

L'utilizzo della maschera di sottorete ha portato alla luce la distinzione tra un indirizzo pubblico, assegnato dall'autoritá competente, e indirizzo privato, ottenuto dalla suddivisione della rete in tante sottoreti.

Sono stati definiti anche degli standard sulla forma che dovrebbe assumere un indirizzo privato:

```
10.0.0.0 - 10.255.255.255
172.16.0.0 - 172.16.255.255
192.168.0.0 - 192.168.255.255
```

Quello che accade oggigiorno é che la scarsitá di indirizzi pubblici ha portato a suddividere le reti sempre piú gerarchicamente. In questo modo un ISP puó acquistare un limitato numero di indirizzi e suddividerlo tra i suoi router dei abbonati. Infatti ogni abbonato possiede una piccola rete interna privata con un solo indirizzo pubblico assegnato al suo router.

Con questo tipo di rete si crea il problema di far comunicare due host in sottoreti diverse con potenzialmente lo stesso indirizzo IP. Esistono due soluzioni al problema: il **NAT** *(Network Address Translation)* o il **Proxy**.

3.2.1 NAT

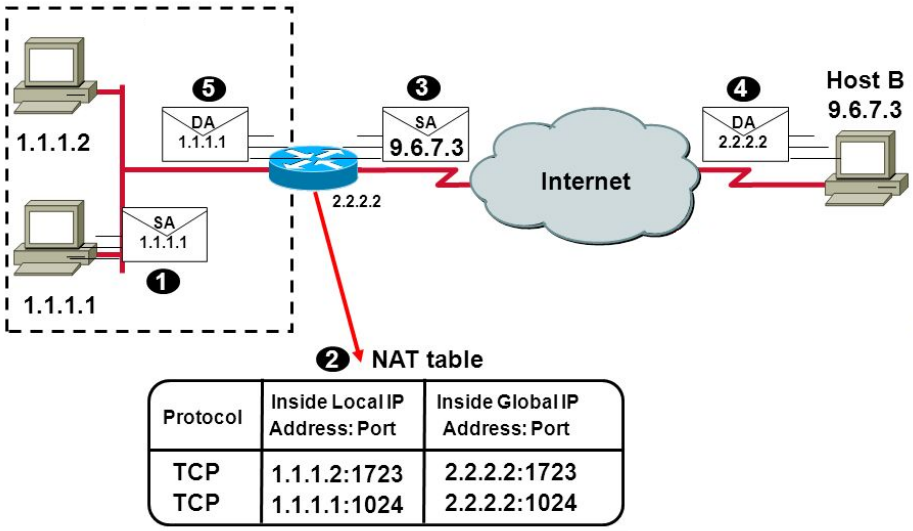
Il sistema NAT permette di sostituire a tutti i datagrammi del livello rete l'indirizzo privato della sottorete interna con l'indirizzo pubblico del router usato per la connessione ad Internet. In altre parole agisce da *"traduttore"* degli indirizzi privati in indirizzi pubblici e viceversa.

In questo modo tutte i datagrammi in uscita verso Internet avranno come sorgente l'indirizzo del router, mascherando l'indirizzamento interno della rete privata.

Per via che questo meccanismo funzioni il router ha bisogno di mantenere una tabella, chiamata **Natting Table** *(Tabella di NAT)*, per salvare tutte le informazioni necessarie alla traduzione degli indirizzi. Solitamente ogni elemento della tabella é cosi composto:

```
SRC IP - DST IP - SRC PORT - DST PORT - PROTOCOL
```

Esempio



Come si vede dall'immagine precedente, l'host nella rete privata con indirizzo `1.1.1.1` cerca di comunicare con l'host con l'indirizzo `9.6.7.3`. Come prima cosa controlla se l'indirizzo di destinazione appartiene alla sua sottorete eseguendo l'AND bit-a-bit con l'indirizzo di destinazione e la sua maschera di sottorete. Verificato che il risultato non é uguale al suo indirizzo di rete, invia i datagrammi al router NAT.

A questo punto viene aggiunto una riga nella tabella di NAT con le informazioni del flussi di datagrammi in uscita, viene costruito un nuovo datagramma cambiando l'indirizzo sorgente in quello del router `2.2.2.2` ed infine viene inviato in Internet.

Come si vede dall'immagine l'Host B non risponde con l'indirizzo di destinazione della rete interna private `1.1.1.1` ma con quello del router `2.2.2.2`. A questo punto eseguirá una traduzione inversa, sempre grazie alla tabella di NAT e inoltrerá il datagramma alla destinazione interna.

3.2.2 Proxy

Un altro modo per permettere ad host con indirizzi privati di comunicare con Internet é quello di utilizzare un Proxy. Un proxy é un software installato su un host server ad indirizzo pubblico che rimane in ascolto delle richieste di livello applicativo da parte di un host interno alla sottorete, quindi esegue una richiesta identica verso Internet con il proprio indirizzo sorgente.

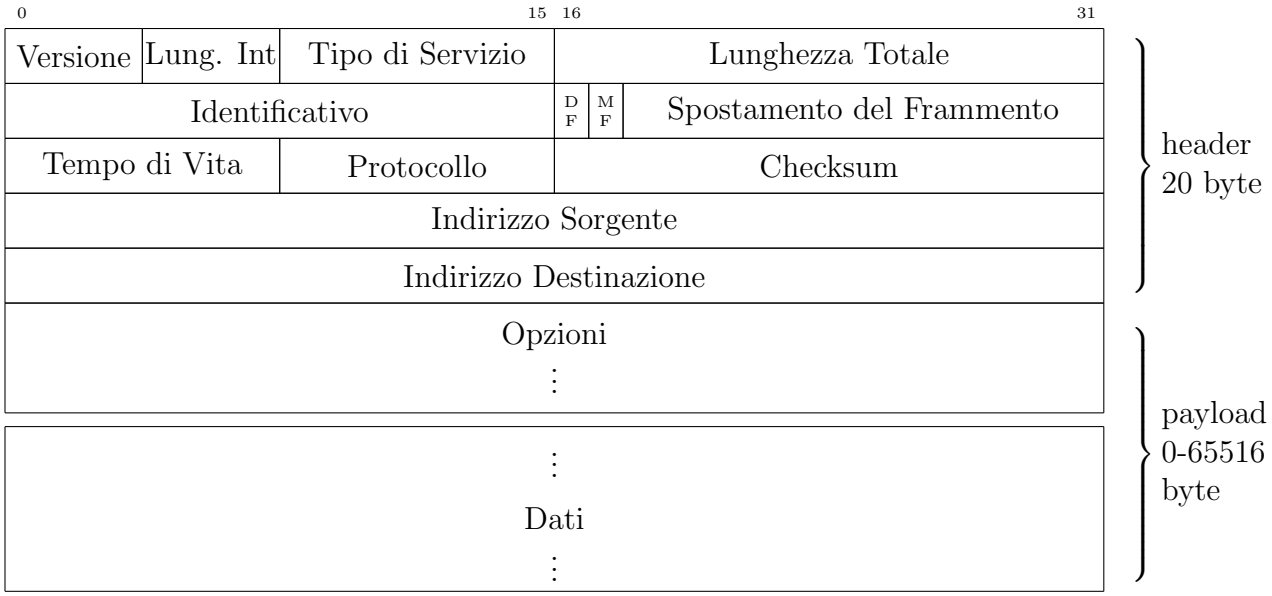
Alla ricezione della risposta, il proxy la invierá all'applicazione originaria sull'host interno della sottorete cambiando l'indirizzo sorgente con il proprio. In questo modo l'host non avrá alcun modo di sapere quale host remoto ha risposto alla sua richiesta, ma saprá soltanto che l'ha ricevuta dal proxy.

Ovviamente, operando a livello Applicazione, devono esistere tanti proxy quante applicazioni.[4]

[4]Al contrario di NAT che, lavorando a livello Rete, é comune a tutte le applicazioni.

3.3 Formato del Datagramma

Il datagramma a livello di rete é costituito da una 20 byte di intestazione e da un 65516 byte di campo dati, per un MSS di 65536 byte (64 kB).



- `Versione`: versione del datagramma: IPv4 o IPv6
- `Lung. Int`: lunghezza dell'header espressa in numero di word a 32 bit
- `Tipo di Servizio`: distingue diverse classi di servizio
- `Lunghezza Totale`: lunghezza totale del datagramma in numero di byte
- `Identificativo`: identifica il datagramma in caso di frammentazione
- `DF`: bit che se impostato indica la non frammentabilitá del datagramma
- `MF`: bit che se impostato indica che questo datagramma é un frammento
- `Spostamento del Frammento`: indica la posizione del frammento nel datagramma corrente
- `Tempo di Vita`[5] tempo di vita del datagramma che viene decrementato dopo ogni inoltro.
- `Protocollo`: indicazione sul protocollo di trasporto usato
- `Checksum`: campo di controllo sull'intestazione del datagramma
- `Indirizzo Sorgente-Destinazione`: indirizzo IP sorgente e destinazione
- `Opzioni`: opzioni aggiuntive di IP, non vengono quasi mai usate

Attualmente i campi che operano con la frammentazione dei datagrammi non vengono piú utilizzati, in quanto si tende a scartare l'intero datagramma per questioni di prestazioni dei router.

[5] Dall'inglese *Time To Live* o *TTL*

3.3.1 ARP

Il protocollo **ARP** *(Address Resolution Protocol)* é una parte cruciale per la comunicazione a livello rete in quanto permette di scoprire la prossima destinazione *"fisica"* a cui inoltrare i datagrammi. Ogni destinazione viene identificata tramite l'indirizzo **MAC** *(Media Access Control)*, numero a 6 byte [6] rappresentato nella seguente forma:

```
00:90:f5:f9:a5:ce
```

Dato un indirizzo destinazione di livello rete, il protocollo ARP si occupa risoluzione di tale indirizzo nel corrispondente MAC.

Il motivo principale per il quale utilizza IP per svolgere le proprie funzioni, é che non sempre un host é interessato alla risoluzione di un indirizzo che risiede sulla sua stessa sottorete, ma molto spesso vuole comunicare con host al di fuori; in questo caso il protocollo IP svolge un ruolo di interfaccia comune tra sottoreti.

La risoluzione di un indirizzo MAC avviene tramite una richiesta a tutta la rete dell'indirizzo MAC corrispondente ad un indirizzo IP, quindi solo l'host con lo stesso indirizzo IP risponderá in unicast al mittente con il proprio MAC.

ARP implementa anche un sistema di *cache*, ovvero un file che risiede nel sistema operativo, nel quale vengono mantenute per circa 30 secondi tutte le associazioni `IP:MAC` che si sono scoperte fino ad un dato momento. Questo meccanismo permette di ridurre il traffico in rete.

3.3.2 ICMP

Il protocollo **ICMP** *(Internet Control Message Protocol)* definisce una serie di messaggi standard per il controllo e la diagnostica a livello di rete.

Purtroppo ICMP e IP sono uno dipendente dall'altro, in quando IP dipende da ICMP per segnalare eventuali errori e ICMP viene incapsulato in un datagramma IP, per passare attraverso reti diverse. Sono stati definiti molti tipi di messaggi ICMP:

- `0 - Echo Reply`: usato dal programma `ping`
- `3 - Destination Unreachable`: quando la destinazione non é raggiungibile
- `5 - Redirect`: l'host, o il router, deve cambiare percorso
- `8 - Echo`: usato dal programma `ping`
- `11 - Time Exceeded`: il campo `TTL` dell'header IP é arrivato a zero
- `12 - Parameter Problem`: l'header IP é stato recapitato incorrettamente
- `30 - Traceroute`: usato dal programma `traceroute`

I datagrammi ICMP sono trattati al pari degli altri circolanti sulla rete, tranne per il fatto che se uno verifica un errore non viene generato un nuovo datagramma ICMP per segnalare l'errore del primo; il motivo é banale: altrimenti la rete sarebbe congestionata solo da pacchetti di errore che generano errori.

[6] nel caso di IPv4

`ping` il programma `ping` invia dei datagrammi ICMP con messaggio `Echo` e si pone in ascolto della risposta con `Echo Replay`. In questo modo, non solo si testa se un host é attivo o meno, ma misurando il tempo trascorso tra l'invio e la ricezione, si misura la latenza tra gli host.

```
$ ping 8.8.8.8
PING 8.8.8.8 8.8.8.8 5684 bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=46 time=56.1 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=46 time=53.8 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=46 time=54.0 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=46 time=54.9 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=46 time=54.7 ms

--- 8.8.8.8 ping statistics ---
6 packets transmitted, 5 received, 16% packet loss, time 5002ms
rtt min/avg/max/mdev = 53.887/54.744/56.102/0.817 ms
```

`traceroute` il programma `traceroute` é utile per tracciare l'ipotetica rotta dei datagrammi. Il funzionamento é molto semplice: si utilizzano i messaggi ICMP `Echo` con `TTL` incrementale partendo da 1 fino a che la destinazione non risponde con un ulteriore datagramma ICMP `Time Exceeded`.

In altre parole: la sorgente, alla prima iterazione invierá il datagramma con `TTL=1`, che eseguirá solo un hop per definizione di `TTL`, misurandone il RTT. Alla seconda iterazione il datagramma avrá `TTL=2`, quindi eseguirá un ulteriore hop dopo quello dell'iterazione precedente e, ancora una volta viene misurato il RTT.

In questo modo si ha una sorta di mappa del percorso che i datagrammi potrebbero prendere nella comunicazione tra gli host.

```
$ tracerouote larepubblica.it
traceroute to larepubblica.it 104.28.4.97, 30 hops max, 60 byte packets
 1  192.168.2.254 192.168.2.254  0.270 ms  0.345 ms  0.407 ms
 2   3  172.17.193.129 172.17.193.129  30.711 ms  34.323 ms  34.599 ms
 4  172.17.192.57 172.17.192.57  36.321 ms 172.17.192.53 172.17.192.53  38.547 ms
    -> 172.17.192.49 172.17.192.49  38.746 ms
 5  172.19.244.94 172.19.244.94  45.357 ms  46.577 ms
    -> 172.19.242.145 172.19.242.145  63.910 ms
 6  etrunk38.milano1.mil.seabone.net 195.22.192.108  50.315 ms  30.919 ms
    -> etrunk38.milano50.mil.seabone.net 195.22.196.92  39.778 ms
 7  ae10.milano58.mil.seabone.net 195.22.208.117  51.659 ms
    -> ae11.milano58.mil.seabone.net 195.22.208.79  34.529 ms
    -> ae10.milano58.mil.seabone.net 195.22.208.117  51.789 ms
 8  cloudflare.milano58.mil.seabone.net 195.22.196.97  36.523 ms  39.928 ms  42.397 ms
 9  104.28.4.97 104.28.4.97  48.575 ms  47.245 ms  44.470 ms
```

3.4 DHCP

Il protocollo **DHCP** *(Dynamic Host Configuration Protocol)* é stato progettato con lo scopo di non configurare manualmente ogni singola interfaccia di rete, ma che ci sia un'autoritá a cui tutti i nuovi host chiedano la configurazione nel momento in cui si vogliono collegare alla sottorete.

Questo processo avviene tramite una richiesta DHCP del nuovo client, mandata in broadcast sulla rete, alla quale solo l'autoritá responsabile, il server DHCP, risponderá tramite un datagramma in broadcast con l'indirizzo IP che intende assegnare al nuovo client. Questo meccanismo é chiamata *offerta* dell'indirizzo; il client puó rifiutare, ma generalmente accetta sempre.

Ovviamente l'indirizzo assegnato in questo modo ha una validitá limitata: una volta scaduta il processo ricomincia, lasciando libera scelta al server DHCP di rinnovare lo stesso indirizzo o di assegnarne uno nuovo.

É possibile, anche, configurare il server in modo tale da rispondere sempre con lo stesso indirizzo IP alla richiesta proveniente da un determinato indirizzo MAC; questo é utilie se si vuole avere una macchina sempre con lo stesso indirizzo IP, come un server.