IP for Fun

Carlo Contavalli – Copyright © 2000-2002 $Revision: 1.7 $ – $Date: 2002/11/16 5:23:56 $

Nel kernel 2.2 sono state aggiunte diverse funzionalità che non erano presenti nei kernel precedenti. Queste nuove funzionalità hanno reso "incompleti" comandi classici come ifconfig e route. Sono stati quindi introdotti dei nuovi comandi che a lungo andare rimpiazzeranno quelli standard. Questo breve documento si propone come scopo quello di introdurre l'utente all'utilizzo basilare del comando ip, uno di questi nuovi comandi. Nelle pagine a seguire, utilizzerò spesso i termini "interfaccia" e "scheda di rete" come sinonimi, anche se in realtà i due termini spesso non sono interscambiabili. Utilizzerò frequentemente anche la parola "pacchettino", riferendomi in maniera un pochino più amichevole ai "pacchetti" ip o di qualsiasi altro tipo che scorrazzano per le nostre reti. Questo articolo ripercorrerà le operazioni che debbono essere svolte sotto un qualsiasi sistema linux per configurare una scheda di rete utilizzando solamente il comando ip.

Contents

1 Copyright e note legali...

2 Prima di cominciare

Questo documento è la seconda versione di un documento che scrissi parecchio tempo fa agli albori del kernel 2.4.

Le prime sezioni sono rimaste praticamente invariate, salvo qualche errore che è stato corretto. Sono state però aggiunte alcune sezioni su funzioni più avanzate del comando ip di cui non avevo parlato in precedenza.

Spero che questo documento possa esservi utile e sarei felice di ricevere commenti, correzioni o qualsivoglia feedback vogliate mandare al mio indirizzo di posta elettronica <ccontavalli at commedia.it>...

Chi di voi volesse scaricare una versione completa ed aggiornata di questo documento, potrebbe trovarla su http://www.commedia.it/ccontavalli/#docs-it .

In ogni caso, quando si parlerà di IP e di protocolli di rete, se non esplicitamente indicato, mi riferirò alla versione 4 del protocollo IP (aka, IPv4).

3 Preparare il proprio sistema

Ok, prima di cominciare, vi conviene controllare la presenza di tutto ciò di cui avrete bisogno: dovete avere una scheda di rete riconosciuta dal kernel con i relativi moduli caricati, una versione del kernel successiva alla 2.2.x (una 2.4.x è perfetta :-) compilata ed installata con il supporto per le netlink socket nonché il programma "ip" disponibile sicuramente presso il sito della vostra distribuzione preferita (il pacchetto si chiama iproute in Debian) oppure scaricabile dal sito *ftp://ftp.inr.ac.ru/ip-routing/* ftp://ftp.inr.ac.ru/ip-routing/ in formato sorgente.

4 Da ifconfig a ip

Queste prime sezioni tenteranno di introdurvi all'utilizzo basilare del comando ip partendo da esempi che utilizzano comandi unix stnadard come ifconfig o route.

4.1 Installare una scheda di rete

Uno dei comandi più conosciuti per la gestione delle interfacce di rete è sicuramente "ifconfig". Per verificare la corretta installazione di una interfaccia, è uso comune dare il comando

```
# ifconfig -a
```

Questo, seppure standard ed ormai parte dei comandi storici di tutti i sistemi unix, può essere sostituito da un più complesso ma altrettanto efficace:

```
# ip link show
```

oppure, più brevemente dal comando

```
# ip link
```

che vi dovrebbe dare un output molto simile a questo:

```
1: lo: <LOOPBACK,PROMISC,UP> mtu 3924 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop qlen 100
    link/ether 00:c0:df:ae:33:48 brd ff:ff:ff:ff:ff:ff
3: sl0: <POINTOPOINT,MULTICAST,NOARP,UP> mtu 1500 qdisc pfifo_fast qlen 10
    link/slip
```

Vediamo quindi di analizzare questa schermata prendendo ad esempio l'interfaccia eth0. Prima di tutto, subito dopo il nome dell'interfaccia è indicato lo stato della stessa racchiuso tra un < ed un >, in questo caso BROADCAST e MULTICAST. Di seguito è indicato l'mtu della scheda (1500), ovvero la dimensione massima in bytes di un pacchetto che può essere spedito utilizzando questa interfaccia (dipende dall'hardware utilizzato dalla scheda di rete – 1500 è il valore utilizzato dalle schede ethernet).

La prima novità appare invece con il parametro qdisc. qdisc significa "Queuing Discipline", ed indica il sistema usato dal kernel per gestire il queue di questa interfaccia (noop).

Ma che cos'è un queue? Un buon esempio di queue è quella che normalmente viene chiamata "coda di stampa". Spesso delle risorse non possono essere utilizzate contemporaneamente da più utenti (come le stampanti), e

devono quindi essere costruite delle "code" (queue in inglese): quando il dispositivo è occupato, tutto ciò che deve essere stampato viene accumulato da qualche parte in attesa che la risorsa si liberi, portandosi poi alla pari con il lavoro. Capita però (con le stampanti in particolare) che qualcuno monopolizzi la risorsa facendo attendere parecchio tempo a tutti gli altri utenti.

Bene, la "queuing discipline" decide come deve essere gestito l'uso della risorsa che non può essere condivisa. In questo caso si tratta di una scheda di rete e più in generale della banda. La disciplina di default è la disciplina pfifo_fast che corrisponde al classico "chi prima arriva meglio si accomoda", fifo infatti sta per "First In First Out", il primo che entra è il primo che esce. Naturalmente è una disciplina molto poco democratica, e sia il kernel 2.2.x che il kernel 2.4.x provvedono a fornire diverse discipline a livello sperimentale (e non) che consentono di cambiare questo comportamento (come l'sfq, il tbf, il red ...). Per esempio, l'sfq o "Stochastic Fairness Queue", fa in modo da dividere l'utilizzo di un'interfaccia in maniera che un po' tutti siano in grado di usufruirne. Se la risorsa non si libera poi, vengono accumulati al massimo qlen (100) arretrati. Nel nostro caso, eth0 usa la disciplina "noop", che corrisponde a "non fare un tubo", o "no operation". Questo perché se notate i flag dell'interfaccia essa non risulta UP (a differenza di sl0), il che significa che non è stata configurata o che è stata disattivata (vedi più avanti). Se la vostra interfaccia fosse invece già stata configurata a boot time, mostrerebbe probabilmente un bell'UP ed un qdisc uguale a pfifo_fast, e nell'esempio seguente mostrerebbe un indirizzo ip già assegnato.

In questa schermata possiamo anche vedere l'indirizzo hardware della scheda di rete ed il suo indirizzo di broadcast (00:c0:df:ae:33:48 ed ff:ff:ff:ff:ff:ff).

Possiamo però notare che, a differenza del comando "ifconfig -a", non vengono date informazioni circa l'indirizzo ip di queste interfacce. Con il nuovo comando ip, la gestione "hardware" e quella logica di ogni dispositivo di rete sono quasi completamente indipendenti. Possiamo quindi visualizzare le informazioni relative alla gestione logica delle interfacce utilizzando il comando:

```
# ip address show
```

o più brevemente

```
# ip addr
```

Notate anche che perché ip riconosca un comando è necessario scrivere soltanto quanto basta ad ip perché il comando non possa essere confuso con un altro, cosicché sono comandi validi "ip add", "ip a" o riferendoci all'esempio di prima "ip l".

L'output del comando "ip addr" nel nostro esempio sarebbe:

```
1: lo: <LOOPBACK,PROMISC,UP> mtu 3924 qdisc noqueue
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
2: eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop qlen 100
   link/ether 00:c0:df:ae:33:48 brd ff:ff:ff:ff:ff:ff
3: sl0: <POINTOPOINT,MULTICAST,NOARP,UP> mtu 1500 qdisc pfifo_fast qlen 10
   link/slip
   inet 192.168.0.1 peer 194.109.2.10/24 scope global sl0
```

Vediamo qui le stesse informazioni di prima più l'indirizzo "inet" (internet ipv4) dell'interfaccia, seguito da un /8 per "lo" o /24 per "sl0". Il numero di seguito alla / indica quanti bit devono essere guardati per decidere se un altro computer si trova sulla stessa rete, ed è un modo più breve per indicare una "network mask". Ad esempio, l'indirizzo dell'host 127.0.0.2, che può anche essere scritto in binario come 01111111.00000000.00000000.00000010, si trova sulla stessa rete di 127.0.0.1 in quanto i primi 8 bit dell'host

127.0.0.2 sono uguali ai primi 8 bit dell'host 127.0.0.1. Nel caso di sl0, il /24 indica che le prime 3 cifre di un altro host devono essere uguali a 194.109.2 perché questo si trovi sulla stessa rete.

Possiamo però vedere che eth0 non ha alcun indirizzo ip assegnato. Questo principalmente per un motivo: nessuno si è ancora preso la briga di configurarlo. Siamo quindi pronti per affrontare il prossimo passo.

4.2 Gestione degli indirizzi ip

A questo punto siamo pronti per assegnare un indirizzo ip alla nostra scheda di rete. Per fare questo, potremmo utilizzare il comando "ifconfig eth0 192.168.200.1", oppure il più recente:

```
# ip addr add 192.168.200.1 dev eth0
```

Eseguiamo ancora una volta il comando "ip addr" per vedere il risultato della precedente istruzione:

```
( . . . )
2: eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast
   qlen 100 link/ether 00:c0:df:ae:33:48 brd ff:ff:ff:ff:ff:ff
   inet 192.168.200.1/32 scope global eth0
( . . . )
```

Ecco che ad eth0 è stato assegnato l'indirizzo 192.168.200.1/32. Abbiamo però commesso un errore: quel /32 indica che gli indirizzi ip dei destinatari dei pacchettini uscenti dalla scheda di rete devono avere tutti e 32 i bit uguali a quelli dell'indirizzo della nostra scheda di rete (in pratica, da questa scheda di rete devono uscire solo pacchettini con indirizzo di destinazione 192.168.200.1 (!!)). E questo probabilmente non è quello che volevamo. Va quindi indicata esplicitamente la network mask del dispositivo, che in questo caso sarebbe 255.255.255.0, corrispondente a 11111111.11111111.11111111.00000000 quindi ad un /24. Siamo quindi costretti a correggere il nostro errore dando un comando del tipo:

```
# ip addr add 192.168.200.1/24 dev eth0
```

Ancora, questo sarebbe l'output di "ip addr":

```
( . . . )
2: eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast
   qlen 100 link/ether 00:c0:df:ae:33:48 brd ff:ff:ff:ff:ff:ff
   inet 192.168.200.1/32 scope global eth0
   inet 192.168.200.1/24 scope global eth0
( . . . )
```

Anche qua compare però un errore. Utilizzando "ifconfig" era possibile assegnare uno ed un solo indirizzo ip ad una scheda di rete. Con ip questo non è più vero: ogni interfaccia può avere un numero indeterminato di indirizzi ip. Basta semplicemente aggiungere o togliere indirizzi ad una scheda di rete. ifconfig era in grado di fare qualcosa di simile (i famosi alias), ma per lui era necessario creare dei dispositivi virtuali come "eth0:0" o "eth0:1".

Per correggere l'errore dobbiamo quindi eliminare l'indirizzo 192.168.200.1/32, e per fare questo possiamo utilizzare il comando:

```
# ip addr del 192.168.200.1/32 dev eth0
```

Dopo questo comando, "ip addr" dovrebbe mostrare la configurazione corretta. Qualcuno di voi avrà poi già provato a "pingare" qualche altro computer della rete, con grande delusione, probabilmente. Questo per due motivi:

- non abbiamo ancora configurato le tabelle di routing
- non abbiamo detto al kernel di attivare l'interfaccia, cioè di metterla "UP"

Per poter configurare le tabelle di routing relative ad un'interfaccia, dobbiamo prima attivarla, dando il comando:

```
# ip link set eth0 up
```

Come già indicato prima, "ip link" si occupa di gestire le interfacce di rete dal punto di vista fisico. Altre cose che può fare sono: disattivare un dispositivo (ip link set eth0 down), disattivare o attivare l'uso dell'address resolution protocol (ip link set eth0 arp off oppure on), cambiare il nome dell'interfaccia (... eth0 name nuovonome), cambiare l'mtu dell'interfaccia (... eth0 mtu 1300 per esempio), cambiare l'indirizzo hardware dell'interfaccia o l'indirizzo hardware di broadcast (... eth0 address xx:xx:xx:xx:xx:xx oppure ... eth0 broadcast xx:xx:xx:xx:xx:xx) o ancora disattivare o attivare il multicast, l'uso di indirizzi dinamici o cambiare la lunghezza del queue. La maggior parte di questi parametri vi conviene però modificarli soltanto se siete veramente consapevoli dei risultati che volete ottenere e quando comunque l'interfaccia è spenta, cioè impostata "down".

4.3 Configurare le tabelle di routing

Una volta configurata un'interfaccia, è necessario configurare correttamente quelle che vengono definite "tabelle di routing". Una tabella di routing contiene l'elenco di tutte le reti raggiungibili tramite il nostro computer e delle informazioni che consentano al kernel del nostro sistema operativo di decidere quale interfaccia utilizzare a secondo delle necessità dell'utente.

Una tabella di routing è quindi costituita da una riga per ogni rete che è possibile raggiungere con le indicazioni sul come raggiungerla. Per visualizzare la tabella di routing correntemente utilizzata è possibile utilizzare il comando vecchio stile "route" oppure il comando:

```
# ip route
```

Che darebbe un output molto simile al seguente:

```
194.109.2.0/24 dev sl0 proto kernel scope link src 192.168.0.1
192.168.200.0/24 dev eth0 proto kernel scope link src 192.168.200.1
```

Dalla prima riga vediamo che possiamo raggiungere tutti i computer i cui primi 24 bit di indirizzo sono uguali a 194.109.2.0 passando attraverso sl0, interfaccia che corrisponde alla prima porta seriale. Vediamo anche che l'indirizzo sorgente di questi pacchettini sarebbe 192.168.0.1. Questo, ad una prima analisi, potrebbe sembrare strano. Com'è possibile che pacchetti destinati alla rete 194.109.2.0/24 debbano uscire da sl0 con un indirizzo di provenienza uguale a 192.168.0.1 che fa parte di tutt'altra rete? Questo è dovuto al fatto che sl0 utilizza una connessione "punto a punto" (point to point – come mostrato dal comando ip addr), ed è quindi un tipo un po' particolare di collegamento che mette l'interfaccia di rete in contatto con un "peer" (compagno) che è poi effettivamente collegato alla rete 194.109.2.0. Per quanto riguarda eth0, invece, vediamo che esiste già una riga nella tabella di routing: questo perché tutte le volte che attiviamo un'interfaccia (la settiamo come UP), automaticamente le reti raggiungibili tramite la stessa vengono aggiunte alla tabella di routing.

Rimane però un problema. Con questa tabella di routing possiamo raggiungere solo la rete 194.109.2 o la rete 192.168.200, ma non tutte le altre reti che fanno parte di internet. Sarebbe però impensabile aggiungere una riga per ogni rete collegata ad internet, così è possibile utilizzare una sorta di jolly: la rete di "default", con indirizzo 0.0.0.0/0, ovvero, tutti i computer (nessun bit deve essere uguale a quello dell'indirizzo).

Per raggiungere computer facenti parte di un'altra rete però, si deve far uso di quello che viene definito gateway o router, ovvero di un computer collegato ad entrambe le reti in grado di trasmettere i dati da una rete all'altra.

Ipotizzando quindi di avere sulla nostra rete un gateway con indirizzo 192.168.200.3 che ci colleghi al resto del mondo, dovremmo configurare il kernel dicendogli di mandare tutti i pacchettini destinati ad ogni rete sconosciuta al computer con questo indirizzo ip. Per fare questo potremmo usare il comando "route add -net default gw 192.168.200.3" oppure il comando:

```
# ip route add default via 192.168.200.3
```

Che dice al kernel qualcosa del tipo "se non sai dove mandare un pacchettino, mandalo a 192.168.200.3". Se a questo punto volessimo eliminare il route appena aggiunto, potremmo dare il comando:

```
# ip route del default
```

Oppure potremmo utilizzare il comando "ip route replace" per rimpiazzare il route con un altro, come mostrato qua sotto:

```
# ip route replace default dev eth1 via 192.168.300.3
```

Il comando "ip route" consente di fare molte altre cose (ip route help né può essere testimone): può, per esempio, gestire delle funzioni di NAT abbastanza basilari, oppure bilanciare il traffico su due differenti gateway o ancora indicare su che interfaccia il kernel manderebbe un pacchetto.

5 Funzionalità avanzate di ip

Nei paragrafi precedenti ho spesso parlato di tabelle di routing al plurale, e non si tratta di un errore: nei kernel 2.2.x e 2.4.x si hanno infatti a disposizione più tabelle di routing.

Di default, infatti, sono presenti 3 tabelle di routing: la tabella main, la tabella local e la tabella cache.

La tabella main contiene la tabella principale, quella che, per intenderci, viene modificata inconsciamente dal comando "route" e che viene modificata in caso non venga indicata esplicitamente un'altra tabella. La tabella cache tiene una cache (come il nome lascia intuire :) dei route utilizzati di recente mentre la tabella local contiene tutti gli indirizzi locali (normalmente gli indirizzi di tutte le interfacce, quelli di broadcast e quelli di multicast). Per vedere il contenuto di una tabella specifica è necessario dare il comando "ip route show table nometabella".

Per vedere il contenuto di tutte le tabelle, potete utilizzare come "nometabella" la parola "all", che vi dovrebbe dare un output simile a:

```
192.168.200.0/24 dev eth0  proto kernel  scope link  src 192.168.200.1
127.0.0.0/8 dev lo  scope link
default via 192.168.200.3 dev eth0
broadcast 127.255.255.255 dev lo  table local  proto kernel  scope link  src 127.0.0.1
broadcast 192.168.200.255 dev eth0  table local  proto kernel  scope link  src 192.168.200.1
local 192.168.200.1 dev eth0  table local  proto kernel  scope host  src 192.168.200.1
```

```
broadcast 192.168.200.0 dev eth0  table local  proto kernel  scope link  src 192.168.200.1
broadcast 127.0.0.0 dev lo  table local  proto kernel  scope link  src 127.0.0.1
local 127.0.0.1 dev lo  table local  proto kernel  scope host  src 127.0.0.1
local 127.0.0.0/8 dev lo  table local  proto kernel  scope host  src 127.0.0.1
```

In questa schermata manca il collegamento seriale (nel frattempo è stato disattivato) e mancano le righe (entry) sugli indirizzi di multicast.

Ancora una volta però, iniziamo ad analizzare la prima riga. In questa riga vediamo che possiamo raggiungere la rete 192.168.200.0/24 attraverso l'interfaccia eth0, vediamo che il protocollo utilizzato per gestire questo indirizzo è quello del kernel, vediamo che lo scope è impostato a link e infine vediamo che l'indirizzo ip sorgente da usare è 192.168.0.1.

Per quanto riguarda il protocollo, questo indica come è stato inserito il route. Nel nostro caso, "kernel" indica che il route è stato automaticamente inserito quando abbiamo attivato l'interfaccia (con ip link set eth0 up). Altri possibili valori sarebbero potuti essere:

- "static", ad indicare che un amministratore ha voluto inserire manualmente il valore e che non deve essere automaticamente modificato
- "boot" che è stato inserito sempre dal *kernel* utilizzando protocolli come il dhcp od il bootp
- "redirect" che è stato inserito per via di un messaggio ICMP che ci indicava un percorso migliore per raggiungere una determinata destinazione

Il parametro "scope" è invece normalmente impostato a "link" per tutti gli host direttamente raggiungibili (per cui non c'è bisogno di passare per un gateway) e quindi per gli indirizzi di broadcast, mentre "host" è usato per tutti gli altri indirizzi nella tabella di local (notate che 127.0.0.0/8 in questo caso non è considerato un indirizzo di broadcast).

Infine, il parametro src indica quale indirizzo ip deve essere utilizzato come sorgente per mandare pacchettini agli altri computer connessi alla rete indicata. Tutti questi parametri possono essere modificati manualmente sempre utilizzando il comando ip route.

Per esempio, è possibile aggiungere un route statico dando un comando simile a:

```
# ip route add 172.163.54.12 via 192.168.200.3 proto static
```

Notate inoltre che in alcune righe viene mostrato il parametro "table". Questo è seguito dal nome della tabella in cui si trova il route. Quando però il nome della tabella viene omesso, come nelle prime tre righe, il route deve essere considerato parte della tabella "main", quella cioè di default.

Prima di andare oltre, è necessario aggiungere qualche piccola informazione su come il kernel faccia ad individuare il route "corretto" da utilizzare (perchè, per esempio, non manda sempre tutti i pacchetti al route di default, ovvero 0.0.0.0/0?). Quando un pacchetto deve essere inviato, il kernel cerca tra tutte le righe nella tabella di routing quelle che indicano un destinatario che potrebbe andare bene. Volendo raggiungere 192.168.0.1, potrebbe quindi scegliere per esempio tre righe come 192.168.0.0/24, 192.168.0.0/16 o 0.0.0.0/0 (default). In caso di ambiguità come queste, sceglierebbe quindi la riga che verifica un numero di bit maggiori (in questo caso, 192.168.0.0/24). Infine, se dovessero rimanere ancora ambiguità, sceglierebbe il route inserito per primo o, se sono stati inseriti dei costi, quello con il costo minore.

Vediamo quindi di fare un esempio. Immaginiamo di avere due interfacce collegate ad una stessa rete. Beh, essendo collegate ad una stessa rete, avranno due indirizzi molto simili, ad esempio 192.168.200.1 ed 192.168.200.2, ed avranno due route molto simili indicati nella tabella di routing, per esempio:

```
192.168.200.0/24 dev inf0  proto kernel  scope link  src 192.168.200.1
192.168.200.0/24 dev inf1  proto kernel  scope link  src 192.168.200.2
```

dove per "inf" è intesa una interfaccia di rete qualsiasi. In questo caso il kernel sceglierebbe probabilmente il primo route, pur non avendo alcun valido motivo per non utilizzare il secondo. Se invece inf0 fosse molto più lenta della seconda interfaccia, si potrebbero associare dei costi a queste interfacce in modo da far utilizzare al kernel sempre inf1 (potrebbe essere il caso, per esempio, di un collegamento di backup via modem isdn o di qualcosa di simile), con qualcosa del tipo:

```
ip route flush dev inf0
ip route flush dev inf1

ip route add 192.168.200.0/24 dev inf0 preference 10
ip route add 192.168.200.0/24 dev inf1 preference 1
```

Dove i primi due comandi eliminerebbero tutti i route relativi ad inf0 ed inf1, mentre gli altri comandi creerebbero rispettivamente un route via inf0 con preferenza a 10 (metric) ed un route tramite inf1 con preferenza pari ad 1.

Attenzione però che la preferenza deve essere intesa come un "costo": il kernel sceglierà infatti il route con preferenza minore.

E' possibile fare anche bilanciamento del carico su più connessioni, ma fino a qualche versione del kernel fa, utilizzando il sistema di bilanciamento standard, non era possibile utilizzare molte funzioni di NAT (Network Address Translation) e vi conviene quindi cercare documentazione specifica in proposito (magari guardando anche ciò che riguarda il "bonding").

5.1 Tabelle di routing multiple

Si presenta però spesso la necessità di inviare traffico differente attraverso interfacce differenti, oppure di avere degli stessi indirizzi ip su diverse interfacce di rete.

Nei kernel precedenti al 2.2 e utilizzando comandi diversi da ip era quasi impossibile utilizzare un firewall linux in queste situazioni. Attualmente invece, è possibile utilizzare più tabelle di routing.

Immaginiamo quindi di avere, per esempio, due connessioni ad internet, una attraverso un provider "serio", per la nostra sala macchine, dove la banda è garantita e deve essere riservata per il traffico dei clienti, ed una connessione ADSL per il nostro ufficio.

In questo caso, una soluzione (sebbene non la migliore ma abbastanza didattica) potrebbe essere quella di mantenere le due reti il più possibile separate, introducendo un firewall con 4 interfacce:

1. la prima collegata alla rete degli uffici
2. la seconda collegata alla rete del provider ADSL
3. la terza collegata alla rete della sala macchine
4. la quarta collegata al provider serio

In questo caso il vostro router linux dovrebbe instradare i pacchetti differentemente, a seconda che vengano o siano destinati all'ufficcio o alla sala macchine.

Per fare questo utilizzando il nuovo comando ip è possibile creare due tabelle di routing distinte: una per la sala macchine e il provider serio, l'altra per l'ufficio e per la linea adsl.

Iniziamo quindi dando un'occhiata nella directory `/etc/iproute2` (o `/etc/iproute` a seconda della distribuzione che state usando). In questa directory vengono mantenuti tutta una serie di file utilizzati dal comando ip per convertire numeri in nomi e viceversa.

Tutte le informazioni vengono infatti memorizzate dal kernel in formato numerico, mentre per noi poveri esseri umani è spesso più facile utilizzare dei nomi.

Se deste per esempio un'occhiata al file rt_protos, sempre nella directory /etc/iproute, vedreste le parole "static", "boot" o "redirect" di cui vi parlavo prima (quando vi descrivevo il significato del campo proto) precedute dal numero utilizzato internamente dal kernel. Allo stesso modo, nel file rt_scopes potreste vedere le associazioni tra scope indicato in formato numerico e lo scope mostrato da ip; nel file rt_dsfield le associazioni utilizzate per la gestione dei servizi differenziati e nel file rt_realms i realm utilizzati (purtroppo non se ne parelrà in questo documento). Infine, nel file rt_tables, troverete le associazioni tra l'identificativo delle varie tabelle di routing ed il loro nome.

Aprite dunque questo file con il vostro editor preferito ed aggiungete una riga del tipo

```
252     smacchine
```

Dove 252 è semplicemente un numero non utilizzato e smacchine un nome arbitrario per la nostra nuova tabella di routing (abbiamo appena battezzato la tabella 252 come "smacchine").

Notate però che le righe precedeute da un "#" sono commenti, e che probabilmente nel vostro file tutte le righe sono commentate. Questo principalmente per un motivo: il comando ip mantiene direttamente tutte le associazioni tra quelle che sono le tabelle standard ed i numeri corrispondenti, e, per evitare confusione, farà in modo da elegantemente ignorare ogni riga nel file di configurazione che vada a modificarle.

Per aggiungere dei route a questa nuova tabella è sufficiente utilizzare i soliti comandi indicando però esplicitamente il nome della tabella.

Immaginiamo quindi che

1. l'interfaccia di rete eth-off collegata agli uffici abbia come indirizzo 10.0.0.1 e che gli uffici utilizzino come netmask 255.255.255.0.
2. l'interfaccia di rete eth-adsl collegata al provider adsl abbia un indirizzo dinamico assegnato da un server dhcp.
3. l'interfaccia di rete eth-smac collegata alla sala macchine abbia come indirizzo 1.2.3.1 e che questa rete utilizzi come netmask 255.255.255.240.
4. l'interfaccia di rete eth-serio collegata al provider serio abbia come indirizzo 1.2.3.2 e che sia collegata direttamente al router (tramite crossover) del provider con indirizzo 1.2.3.14.

Ipotizzando di aver già impostato correttamente gli indirizzi ip sulle varie schede di rete, ci troveremmo quindi ad una situazione simile (con ip addr show) alla seguente:

```
1: lo: <LOOPBACK,UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope host lo
2: eth-off: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:50:04:8e:a2:da brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.1/24 brd 10.0.0.255 scope global eth0
3: eth-adsl: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:50:04:8e:a2:db brd ff:ff:ff:ff:ff:ff
4: eth-smac: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:50:04:8e:a2:dc brd ff:ff:ff:ff:ff:ff
    inet 1.2.3.1/28 brd 1.2.3.15 scope global eth0
5: eth-serio: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:50:04:8e:a2:dd brd ff:ff:ff:ff:ff:ff
    inet 1.2.3.14/32 brd 192.168.200.255 scope global eth0
```

Ma vediamo di analizzare la situazione prima di proseguire. Per quanto riguarda eth-off, non credo che vi sia alcun problema: è configurata in una delle maniere più standard possibili. eth-adsl, invece, non è stata attivata: questo principalmente perchè se ne occuperà pump (o dhcpcd – sono entrambi dei client dhcp) entro poco. eth-smac ha come indirizzo quello sopra indicato (1.2.3.1) e come netmask un /28. Questo perchè 255.255.255.240 in formato binario si può scrivere come 11111111.11111111.11111111.11110000, cioè con 28 "uno" (si può anche calcolare in questo modo: 255 (il massimo) - 240 = 16 (ok, 15, ma arrotondiamolo alla potenza di 2 più vicina), 16 si può esprimere utilizzando al più 4 bit (4 = logaritmo base 2 di 16, 2**4 è proprio uguale a 16) – 32 - 4 = 28). Ok, andando avanti, eth-serio ha come indirizzo ip 1.2.3.14/32. Teoricamente però, 1.2.3.14 si dovrebbe trovare sulla rete collegata ad eth-smac. E' per questo che la netmask è stata specificata con un /32: ad indicare che nessun altro computer di tale rete è raggiungibile tramite questa interfaccia. Vediamo però cosa ha fatto il kernel in automatico sulle nostre tabelle di routing:

```
# ip route show table main
  127.0.0.0/8 dev lo  scope link
  10.0.0.0/24 dev eth-off proto kernel scope link src 10.0.0.1
  1.2.3.0/28 dev eth-smac proto kernel scope link src 1.2.3.1
# ip route show table smacchine
#
```

Ok, prima di tutto non è stata aggiunta alcuna entry per l'interfaccia eth-serio nella tabella main (quella di default): questo soprattutto perchè dal punto di vista del kernel eth-serio non è collegata ad alcuna rete (ricordate il /32?).

In secondo luogo, la tabella smacchine è completamente vuota. L'unico modo per aggiungere delle entry in tabelle diverse da quella di default è quello di richiederlo esplicitamente.

Ipotizzando quindi di riuscire a fare in modo che i nostri server utilizzino la tabella "smacchine" anziché la tabella main, in questa tabella dovranno essere presenti sicuramente le seguenti entry:

- 1.2.3.0/28, per permettere al nostro router di comunicare con i nostri server
- un entry per permettere al nostro router di comunicare col gateway del provider serio, che, come ricorderete, fa parte di un'altra rete.
- un entry per permettere alla rete dei server di contattare direttamente l'ufficio (senza passare da internet)
- un entry di default per permettere al router di raggiungere il resto del mondo.

Vediamo quindi di aggiungere queste entry:

```
# ip route add 1.2.3.0/28 dev eth-smac table smacchine
# ip route add 1.2.3.2 dev eth-serio table smacchine
# ip route add 10.0.0.0/24 dev eth-off table smacchine
# ip route add default via 1.2.3.2 dev eth-serio table smacchine
```

Le particolarità iniziano qui dalla seconda riga: in caso il router dovesse contattare il computer con indirizzo 1.2.3.2 utilizzerebbe questa riga piuttosto che la prima in quanto è quella che corrisponde per un numero maggiore di bit. In questo caso, è indispensabile perchè tutto funzioni correttamente specificare come scheda di rete eth-serio. In pratica, è come se con la prima riga avessimo detto al kernel che "tutti i computer con indirizzo 1.2.3.0/28 sono raggiungibili tramite eth-smac", e con la seconda "tranne 1.2.3.2, che è raggiungibile tramite eth-serio".

Infine, la quarta riga indica che per raggiungere internet è possibile utilizzare il gateway con indirizzo 1.2.3.2, che, dalla seconda riga, si trova su eth-serio.

Ci manca ancora un passo però per arrivare ad una configurazione "funzionante": dire al kernel in quali situazioni utilizzare la tabella "smacchine" invece della tabella "main".

Per fare questo, è necessario utilizzare il comando "ip rule" (regole ip). Nel nostro caso basterebbe qualcosa del tipo

```
# ip rule add from 1.2.3.0/28 lookup smacchine
# ip rule add to 1.2.3.0/28 lookup smacchine
```

per dire al kernel di utilizzare la tabella smacchine per tutto ciò:

1. che proviene da 1.2.3.0/28
2. che è destinato a 1.2.3.0/28

Infine, dobbiamo assegnare un indirizzo ip all'interfaccia eth-adsl ed un route di default per la tabella main. Per fare questo, possiamo semplicemente utilizzare "dhcpcd -i eth-adsl" per richiedere tramite dhcp le informazioni mancanti.

E' importante notare che è fondamentale lasciare la tabella di default per gli uffici e la linea ADSL ed utilizzare un'altra tabella per i server. Questo principalmente perchè comandi come pump o dhcpcd non consentono ancora di specificare quale tabella dovrà essere modificata con le informazioni reperite tramite dhcp o bootp, ed andranno quindi a modificare la tabella di main.

5.2 Regole di ip

Ok, abbiamo già visto il comando "ip rule add" per aggiungere delle regole. Nel nostro caso, abbiamo aggiunto delle regole basate sull'indirizzo ip sorgente e destinazione dei pacchetti ip. `ip rule` consente di aggiungere delle regole basate anche:

- sul `tos`
- sul dispositivo di input (con `dev` o `iif`)
- sul `fwmark`

Come potete vedere, non è possibile fare dei match in base alla porta tcp o udp. Il problema è risolto tramite l'utilizzo di fwmark, che vi consente di riconoscere pacchetti marcati usando comandi come iptables o ipchains, strumenti che consentono di scrivere regole molto più selettive su tutti i campi dei vari protocolli (vi conviene guardare la documentazione di iptables o ipchains per maggiori informazioni).

Per vedere invece le regole correntemente inserite, è sufficiente utilizzare un comando simile a "ip rule show", che ci darebbe un output simile a:

```
0:      from all lookup local
32764:  from all to 1.2.3.0/28 lookup smacchine
32765:  from 1.2.3.0/28 lookup smacchine
32766:  from all lookup main
32767:  from all lookup 253
```

Ok, il primo numero a sinistra indica l'ordine in cui vengono controllate le regole (una regola con un numero minore viene controllata per prima – indica cioè la posizione della regola), le prime parole seguenti indicano cosa deve essere verificato nel pacchetto (all corrisponde a 0.0.0.0/0) mentre normalmente le ultime due parole indicano l'azione da compiere. In tutte le regole sopra esposte, viene indicato di effettuare il lookup in particolari tabelle.

In generale, è possibile specificare come azione quella di:

- effettuare il lookup in particolari tabelle con `lookup`
- proibire il transito dei pacchetti (generando l'errore ICMP communication prohibited) con `prohibit`
- rendere il percorso irraggiungibile (generando l'errore ICMP network unreachable) con `unreachable`
- buttare via completamente il pacchetto con `blackhole`
- effettuare una forma primitiva (utile per altri scopi rispetto quelli per cui è normalmente utilizzato) di nat con `nat`

Vediamo quindi un esempio utilizzando qualche regola un pochino più avanzata

```
# ip rule add from 10.0.0.11 unreachable pref 11
# ip rule add tos 0x12 from 10.0.0.12 lookup t3
# ip rule add from 10.0.0.0/24 fwmark 12 prohibit pref 2
```

Con il primo comando abbiamo inserito una regola in posizione 11 che fa in modo che tutti i pacchetti ip provenienti dal computer 10.0.0.11 vengano respinti con un un bel "network unreachable".

La seconda regola prende invece tutti i pacchetti provenienti da 10.0.0.12 con un tos di valore 0x12 (maximize throughput e maximize reliability) e fa effettuare il lookup della tabella chiamata t3. In questo caso, non è stata data una posizione specifica con l'istruzione `pref`. Attenzione che comandi successivi, a meno che non venga esplicitamente indicata una posizione, aggiungono sempre regole con precedenza maggiore (quindi posizione minore) rispetto tutte le altre regole. Questa seconda regola verrebbe quindi inserita in posizione 10. Infine, l'ultima regola viene inserita in posizione 2 e fa in modo che vengano proibiti tutti i pacchetti provenienti dalla rete 10.0.0.0/24 e marcati dal firewall con un valore di 12.

Attenzione però che la prima regola, con priorità 0 che effettua il lookup nella tabella local non può essere rimossa.

Per maggiori informazioni su come marcare i pacchetti con ipchains o iptables, vi conviene riferirvi alle relative pagine di manuale.

6 Troubleshooting – individuare gli errori

Finchè le regole sono poche o comunque le tabelle di routing sono abbastanza semplici, capita di rado di fare errori. Quando invece le regole iniziano a moltiplicarsi ed esistono delle entry ambigue nelle varie tabelle di routing è indispensabile essere in grado di effettuare dei controlli e delle verifiche.

Per fare questo, è possibile utilizzare il comando "ip route get", che ci permette di sapere da che parte verrebbe instradato un pacchetto con determinate caratteristiche. Per esempio

```
# ip route get 192.168.12.14 from 1.2.3.4 iif eth-smac
```

ci direbbe da che parte verrebbe mandato un pacchettino destinato a 192.168.12.14 proveniente da 1.2.3.4 ed entrato nel nostro router tramite interfaccia eth-smac.

Attenzione che se le informazioni fornite sono incoerenti, ip vi butterà fuori con un errore del tipo "invalid argument".

Vi posso garantire che "ip route get" può farvi risparmiare parecchio tempo...

7 Da arp a ip

Oltre ai comandi ifconfig e route, il comando `ip` può rimpiazzare alcuni altri comandi. In particolare, al posto di "arp" può essere utilizzata la famiglia di comandi "ip neigh" per la gestione dell'Address Resolution protocol.

Per esempio, il comando "ip neigh show" mostra gli host dei quali è conosciuto l'indirizzo hardware, e corrisponde quindi al comando "arp -a". E' possibile invece aggiungere host nella cache usando il comando "ip neigh add" come mostrato qua sotto:

```
# ip neigh add 192.168.200.5 lladdr 00:5c:4a:3d:2e:5f nud permanent dev eth0
```

In questo caso abbiamo aggiunto un host con indirizzo ip 192.168.200.5 e indirizzo hardware 00:5c:4a:3d:2e:5f. "nud" specifica invece in che modo il record deve essere creato nella cache, e "permanent" specifica che non deve mai essere rimosso. Altre possibilità sono "noarp", "stale" oppure "reachable", che comunque vengono utilizzati soltanto in casi particolari. Il comando equivalente sarebbe stato in questo caso "arp -s 192.168.200.5 00:5c:4a:3d:2e:5f". E' anche possibile rimuovere o cambiare dei record nella cache usando il comando "ip neigh del" ed "ip neigh change" come mostrato qua sotto:

```
# ip neigh change 192.168.200.5 nud noarp dev eth0
```

Con questo primo comando abbiamo cambiato lo stato del record relativo all'host 192.168.200.5 da permanente a "noarp", che indica al kernel la validità del record fino alla sua scadenza senza bisogno di riconvalida. Notate che a questo punto questo record non verrebbe mostrato dando un altro "ip neigh show". Per vederlo, bisognerebbe specificare esplicitamente che si desidera vedere tutti i record settati a noarp, con il comando "ip neigh show nud noarp".

```
# ip neigh del 192.168.200.5 dev eth0
```

Con questa riga abbiamo invece rimosso il record relativo allo stesso host. Notate anche che il più delle volte è necessario specificare esplicitamente su che interfaccia si ha intenzione di lavorare.

8 Altre funzioni interessanti

Se vi vorrete scrivere degli script da voi o per modificare una configurazione a runtime senza troppa fatica e senza bisogno di riavviare, potrebbe essere comodo usare il comando "flush", disponibile nella maggior parte dei casi.

Per esempio, per eliminare tutti gli indirizzi ip assegnati ad una scheda di rete, si potrebbe usare senza troppo sforzo un comando del tipo

```
# ip addr flush dev eth0
```

oppure, per ripulire una tabella di routing

```
# ip route flush table main
# ip route flush table smacchina
```

Naturalmente, è anche possibile eliminare singolarmente sia gli indirizzi ip assegnati alle schede di rete, sia le entry nelle tabelle di routing, ripetendo esattamente quello che si è scritto col comando add, utilizzando invece l'opzione del, per esempio con

```
# ip addr del 10.30.0.0/24 dev eth0
```

Come con ifconfig, è anche possibile visualizzare delle statistiche utilizzando l'opzione "-s", ad esempio

```
# ip -s link show
```

vi mostrerebbe le statistiche sui dispositivi hardware installati.

Altre funzioni interessanti non discusse su questo documento sono:

- la gestione dei tunnel
- il proxy arp (non proprio relativo a ip, ma comunque interessante)
- la gestione degli indirizzi di multicast

9 Conclusione

Abbiamo visto come si può configurare e gestire una scheda di rete in linux utilizzando solamente il comando ip. ip però non è ancora il sistema utilizzato dalla maggior parte delle distribuzioni, e spesso è necessario scriversi da sé gli script di configurazione necessari da inserire nel processo di init, anche se in genere ip convive abbastanza bene con tutti i suoi fratelli più standard.

ip è in grado di fare molte più cose rispetto quelle di cui vi ho parlato e potete avere maggiori informazioni consultando l'"ip command reference" (ip-cref.ps) fornita in formato postscript con tutte le versioni di ip oppure chiedere aiuto utilizzando "ip help" o "ip famiglia help".

C'è anche disponibile una piccola howto che va completandosi scaricabile da http://www.ds9a.nl/2.4Routing , che tratta specificatamente le funzionalità di routing avanzate. Buon Divertimento!