Configurare i tunnel con iproute2

Simone Piunno
Deep Space 6

simone@deepspace6.net

1. iproute2

iproute2 è un pacchetto per la gestione avanzata della configurazione di rete in ambente Linux. Si tratta in pratica di una serie di utility che fanno uso delle rtnetlink socket una moderna e potente interfaccia di configurazione dinamica dello stack di rete implementata da Alexey Kuznetsov a partire dai kernel della versione 2.2.

La caratteristica più interessante di **iproute2** è il fatto che fornisce un unico comando per fare in maniera organica e integrata tutte le cose che eravamo abitutati a fare con **ifconfig**, **arp**, **route** e **iptunnel**, oltre a molte altre.

iproute2 viene ormai installato di default da tutte le maggiori distribuzioni, anche se i loro scripts di inizializzazione tendono ad usare ancora i comandi del pacchetto *net-tools* (es. **ifconfig** o **iptunnel** - quest'ultimo è attualmente deprecato). Se la vostra distribuzione non contiene questo importante pacchetto potete sempre scaricarlo da ftpsite e compilarvelo voi stessi.

Il difetto principale di **iproute2** è in questo momento la scarsità di documentazione, poichè però l'interfaccia del comando **ip** è molto semplice, chi conosce già **ifconfig** e **route** non dovrebbe trovare molte difficoltà nell'utilizzo di questo nuovo strumento.

2. Il tunnel, questo sconosciuto

Spesso capita che due nodi della rete vogliano scambiare traffico che è stato incapsulato con un protocollo differente da IPv4 o che è indirizzato verso una LAN privata i cui indirizzi IPv4 non sono validi su internet. In queste situazioni il problema si risolve generalmente utilizzando una connessione virtuale punto-punto tra i due nodi, chiamata *tunnel*.

Ogni pacchetto in transito sulla rete può essere pensato come una busta contente dei bit e con gli indirizzi di mittente e destinatario stampigliati all'esterno. I tunnel sostanzialmente prendono questa busta e la infilano in una busta ulteriore, di fatto dirottando il viaggio del pacchetto. Al raggiungimento del destinatario fittizio, la busta esterna viene rimossa e il pacchetto prosegue il suo viaggio verso il destinatario vero.

I due nodi che si occupano di mettere e togliere la busta aggiuntiva vengono chiamati endpoint e devono avere un indirizzo IPv4 reciprocamente noto. Per questo motivo i tunnel in genere non funzionano quando attraversano un procedimento di network address translation (NAT). Inoltre, se il tunnel attraversa un firewall, quest'ultimo dovrà essere opportunamente configurato per lasciar passare questo tipo di traffico.

Un tipico uso dei tunnel è nel collegamento di due nodi IPv6 separati da un rete IPv4. I due nodi possono costruire un tunnel che incapsuli ogni pacchetto IPv6 in un pacchetto IPv4, e in questo modo possono simulare un connessione IPv6 reale e interconnettere tra loro due isole IPv6 (6bone è tenuta insieme proprio in questo modo, con una ragnatela di tunnel). I tunnel per trasportare IPv6 sopra IPv4 sono di due tipi: automatico RFC2373 e configurato manualmente. In questo documento ci occuperemo soltanto del secondo tipo.

3. Creare un tunnel

La creazione di un tunnel con **iproute2** è molto semplice. Per prima cosa bisogna dare un nome al nostro tunnel. Ad esempio scegliamo di chiamarlo *pippo*:

```
ip tunnel add pippo mode sit remote 192.168.1.42
```

In questo modo abbiamo creato un tunnel sit (IPv6-in-IPv4) il cui altro estremo si trova all'ip 192.168.1.42. Da notare che non abbiamo detto nulla sull'ip che vogliamo usare al nostro capo del tunnel, sull'interfaccia sulla quale va attestato, ecc. Possiamo vedere il risultato con il comando **ip tunnel show**:

```
# ip tunnel show
sit0: ipv6/ip  remote any  local any  ttl 64  nopmtudisc
pippo: ipv6/ip  remote 192.168.1.42  local any  ttl inherit
```

Il nostro tunnel compare in seconda riga. Ora possiamo anche vedere un elenco di tutte le interfacce di rete disponibili, siano essere reali o fittizie:

```
# ip link show
1: lo: <loopback,up> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <broadcast,multicast,up> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:48:54:1b:25:30 brd ff:ff:ff:ff:ff:ff
4: sit0@none: <noarp> mtu 1480 qdisc noop
    link/sit 0.0.0.0 brd 0.0.0.0
6: pippo@none: <pointopoint,noarp> mtu 1480 qdisc noop
    link/sit 0.0.0.0 peer 192.168.1.42
```

La cosa da notare è che mentre lo e eth0 sono marcati *up* (ovverosia sono interfacce attivate), i due tunnel invece non lo sono (quindi sono interfacce configurate ma non ancora attivate). Infatti, utilizzando il vecchio **ifconfig** (che mostra solo interfacce attivate) vedremo solo:

```
# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:48:54:1b:25:30
          inet addr:192.168.0.1  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::248:54ff:fe1b:2530/10 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 b)  TX bytes:528 (528.0 b)
          Interrupt:9 Base address:0x5000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 scope:host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:35402 errors:0 dropped:0 overruns:0 frame:0
          TX packets:35402 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3433996 (3.2 mb)  TX bytes:3433996 (3.2 mb)
```

Bisogna quindi ricordarsi che **ip link** vede tutte le interfacce presenti, siano esse attivate o meno. Per attivare l'interfaccia pippo si usa il comando:

```
ip link set pippo up
```

e per disattivarla:

```
ip link set pippo down
```

per eliminare il tunnel del tutto si usa:

```
ip tunnel del pippo
```

4. Argomenti avanzati

4.1. Tunnel GRE

Se non ci interessa incanalare traffico IPv6 all'interno di un tunnel IPv4, ma ad esempio vogliamo incanalare traffico IPv4, allora invece di utilizzare l'opzione `mode sit` per la creazione di un tunnel di

tipo `sit` (IPV6-in-IPv4) vorremo probabilmente usare l'opzione *`mode gre`*, che ci consente di creare un tunnel di tipo RFC2784. Ecco un esempio:

```
# ip tunnel add pippo4 mode gre remote 192.168.1.42
# ip tunnel show
gre0: gre/ip  remote any  local any  ttl inherit  nopmtudisc
pippo4: gre/ip  remote 192.168.1.42  local any  ttl inherit
# ip link show
1: lo: <loopback,up> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <broadcast,multicast,up> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:48:54:1b:25:30 brd ff:ff:ff:ff:ff:ff
7: gre0@none: <noarp> mtu 1476 qdisc noop
    link/gre 0.0.0.0 brd 0.0.0.0
9: pippo4@none: <pointopoint,noarp> mtu 1476 qdisc noop
    link/gre 0.0.0.0 peer 192.168.1.42
```

`GRE` è un particolare tipo di tunnel supportato dai router Cisco e capace di trasportare diverse tipologie di traffico sopra IPv4. Esiste anche un altro tipo di tunnel implementato da linux: `ipip` che è un tipo di tunnel IPv4-in-IPv4 implementato solo su Linux e non permette di trasportare alcuni tipi di traffico (es. broadcast, ipx), quindi in generale il suo utilizzo è sconsigliabile.

4.2. Esplicitare l'estremo locale del tunnel

Anche se il kernel è abbastanza intelligente per capire da solo a quali indirizzo ip ed interfaccia l'estremo locale del tunnel deve essere connesso, può essere una buona idea indicare esplicitamente questi dati. Per fare questo possiamo usare i parametri *`local`* e *`dev`*, ad esempio:

```
# ip tunnel add pippo mode sit local 192.168.0.1 remote 192.168.1.42 dev eth0
# ip tunnel show
sit0: ipv6/ip  remote any  local any  ttl 64  nopmtudisc
pippo: ipv6/ip  remote 192.168.1.42  local 192.168.0.1  dev eth0  ttl inherit
# ip link show
1: lo: <loopback,up> mtu 16436 qdisc noqueue
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <broadcast,multicast,up> mtu 1500 qdisc pfifo_fast qlen 100
   link/ether 00:48:54:1b:25:30 brd ff:ff:ff:ff:ff:ff
4: sit0@none: <noarp> mtu 1480 qdisc noop
   link/sit 0.0.0.0 brd 0.0.0.0
11: pippo@eth0: <pointopoint,noarp> mtu 1480 qdisc noop
   link/sit 192.168.0.1 peer 192.168.1.42
```

Da notare che ora l'interfaccia è identificata dal nome pippo@eth0.

4.3. Time-to-live

Quando si usano dei tunnel è estremamente semplice creare inavvertitamente dei loop nella rete. Per questo motivo è di fondamentale importanza tenere basso il valore del time-to-live (TTL) dei pacchetti. Questo valore può essere specificato tramite il parametro `ttl` del comando **ip tunnel add** e per default viene ereditato il valore associato alla scheda di rete cui il tunnel viene connesso. IANA consiglia di usare un TTL pari a 64.

5. Assegnare indirizzi alle interfacce

Come ogni altra interfaccia di rete, i tunnel possono avere uno o più indirizzi ip assegnati.

5.1. Indirizzo principale

Assegnare l'indirizzo principale à semplicissimo:

```
ip addr add 3ffe:9001:210:3::42/64 dev pippo
ip addr add 192.168.0.2/24 dev pippo4
ip addr add 10.20.30.40/8 dev eth0
```

Il numero indicato a destra della barra serve per suggerire al kernel quale dovrebbe essere l'estensione del prefisso di rete, utile soprattutto per calcolare automaticamente l'indirizzo broadcast e la netmask sulle LAN IPv4. Poichè però il tunnel è una interfaccia punto-punto, tale valore è per esso ininfluente.

Nota bene: per poter assegnare un IP ad una interfaccia bisogna prima averla attivata con **ip link set nome up**.

Per rimuovere indirizzi IP da una interfaccia basta naturalmente usare **del** al posto di **add**:

```
ip addr del 3ffe:9001:210:3::42/64 dev pippo
ip addr del 192.168.0.2/24 dev pippo4
```

Possiamo anche chiedere un elenco di tutti gli indirizzi IP presenti sul nostro server:

```
# ip addr show
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
    inet6 ::1/128 scope host
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:48:54:1b:25:30 brd ff:ff:ff:ff:ff:ff
```

```
    inet 192.168.0.1/24 brd 192.168.0.255 scope global eth0
    inet6 fe80::248:54ff:fe1b:2530/10 scope link
4: sit0@NONE: <NOARP> mtu 1480 qdisc noop
    link/sit 0.0.0.0 brd 0.0.0.0
5: pippo@NONE: <POINTOPOINT,NOARP> mtu 1480 qdisc noop
    link/sit 0.0.0.0 peer 192.168.1.42
    inet6 3ffe:9001:210:3::42/64 scope global
    inet6 fe80::c0a8:1/10 scope link
```

5.2. Aliasing

Per quanto riguarda l'uso di indirizzi multipli su una singola interfaccia, chi è abituato ad usare **ifconfig** sarà sorpreso di vedere che multipli comandi **ip addr add** non generano delle interfacce fittizie tipo eth0:1, eth0:2 eccetera (uno schema di denominazione retaggio dei kernel 2.0). Ad esempio:

```
# ip addr add 192.168.0.11/24 dev eth0
# ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:48:54:1b:25:30 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.1/24 brd 192.168.0.255 scope global eth0
    inet 192.168.0.11/24 scope global secondary eth0
    inet6 fe80::248:54ff:fe1b:2530/10 scope link
# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:48:54:1B:25:30
          inet addr:192.168.0.1  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::248:54ff:fe1b:2530/10 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 b)  TX bytes:528 (528.0 b)
          Interrupt:9 Base address:0x5000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:34732 errors:0 dropped:0 overruns:0 frame:0
          TX packets:34732 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3386912 (3.2 Mb)  TX bytes:3386912 (3.2 Mb)

pippo     Link encap:IPv6-in-IPv4
          inet6 addr: 3ffe:9001:210:3::42/64 Scope:Global
          inet6 addr: fe80::c0a8:1/10 Scope:Link
          UP POINTOPOINT RUNNING NOARP  MTU:1480  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
```

```
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

Il nostro IP aggiuntivo viene riportato da **ip addr show** (e funziona se proviamo ad usarlo) ma **ifconfig** non si accorge della sua esistenza! Per risolvere il problema possiamo usare il parametro label:

```
# ip addr add 192.168.0.11/24 label eth0:1 dev eth0
# ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:48:54:1b:25:30 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.1/24 brd 192.168.0.255 scope global eth0
    inet 192.168.0.11/24 scope global secondary eth0:1
    inet6 fe80::248:54ff:fe1b:2530/10 scope link
# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:48:54:1B:25:30
          inet addr:192.168.0.1  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::248:54ff:fe1b:2530/10 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 b)  TX bytes:528 (528.0 b)
          Interrupt:9 Base address:0x5000

eth0:1    Link encap:Ethernet  HWaddr 00:48:54:1B:25:30
          inet addr:192.168.0.11  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Interrupt:9 Base address:0x5000
```

Notare che nella label, a destra dei due punti possiamo mettere una stringa arbitraria, non importa che si tratti di un numero.

5.3. Quale IP dare al tunnel

Associare un indirizzo IP globale/pubblico (rispettivamente un indirizzo IPv6 per tunnel SIT/IPv6-in-IPv4 e un indirizzo IPv4 per tunnel GRE/IPv4-in-IPv4) ad un tunnel è probabilmente la scelta migliore quando il nostro computer è un singolo host e non un router di confine che vuole offrire connettività IPv6 per una intera LAN.

Se invece stiamo configurando un router, conviene lasciare al tunnel il solo indirizzo link-local nel caso di tunnel IPv6-in-IPv4 (indirizzo che normalmente viene attribuito automaticamente tramite stateless address configuration oppure configurato manualmente), oppure assegnare un indirizzo di rete privata nel caso di tunnel IPv4-in-IPv4 (in IPv4 non esistono gli indirizzi di tipo link-local). L'indirizzo valido sarà quindi attribuito solo a eth0 (o all'interfaccia rivolta alla LAN). Notare che in questa situazione occorre attivare il forwarding dei pacchetti tra le interfacce, tramite questi comandi:

```
sysctl -w net.ipv4.conf.all.forwarding=1  # per tunnel GRE (IPv4-in-IPv4)
sysctl -w net.ipv6.conf.all.forwarding=1  # per tunnel SIT (IPv6-in-IPv4)
```

Con IPv4 si può anche decidere di abilitare il forwarding solo tra alcune interfacce invece che fra tutte, in questo caso per esempio potremmo voler usare i comandi:

```
sysctl -w net.ipv4.conf.eth0.forwarding=1
sysctl -w net.ipv4.conf.pippo.forwarding=1
```

Avvertimento

il significato di questo parametro per IPv6 è diverso e funziona in modo differente da come ci si aspetterebbe, vedi la documentazione del kernel per maggiori informazioni.

6. Routing

Ora che abbiamo configurato un tunnel dobbiamo decidere quale tipo di traffico dirigere attraverso di esso. Per IPv6 la scelta più comune è la seguente:

```
ip route add 2000::/3 dev pippo
```

In questo modo tutto il traffico IPv6 verso indirizzi che hanno i primi tre bit pari a 001 (ovvero tutto lo spazio di indirizzamento globale unicast IPv6) viene indirizzato verso l'interfaccia pippo. Nonostante sia stato selezionato soltanto un ottavo dello spazio d'indirizzamento IPv6, ogni nostro possibile interlocutore si troverà in questo arco di indirizzi.

Possiamo vedere la tabella di instradamento IPv4 con il comando:

```
# ip route
192.168.0.0/24 dev eth0  scope link
127.0.0.0/8 dev lo  scope link
```

e la tabella di instradamento IPv6 con il comando:

```
# ip -6 route
3ffe:9001:210:3::/64 via :: dev pippo  proto kernel  metric 256  mtu 1480 advmss 1420
fe80::/10 dev eth0  proto kernel  metric 256  mtu 1500 advmss 1440
fe80::/10 via :: dev pippo  proto kernel  metric 256  mtu 1480 advmss 1420
ff00::/8 dev eth0  proto kernel  metric 256  mtu 1500 advmss 1440
```

```
ff00::/8 dev pippo  proto kernel  metric 256  mtu 1480 advmss 1420
default dev eth0  proto kernel  metric 256  mtu 1500 advmss 1440
unreachable default dev lo  metric -1  error -101
```

In caso sia necessario specificare un gateway (non succede con i tunnel) allora possiamo usare il parametro *via*, per esempio:

```
ip route add 192.168.1.0/24 via 192.168.0.254 dev eth0
```

Per eliminare una rotta si usa naturalmente **ip route del** ma attenzione a quello che fate... se scrivete **ip route del default** state eliminando la rotta di default per IPv4, non quella per IPv6! Per eliminare il default di IPv6 ci vuole **ip -6 route del default**.

7. Applicazione pratica

7.1. Un esempio completo

Ecco in breve come si prepara un tipico tunnel per connettere un host a 6bone:

```
ip tunnel add $TUNNEL mode sit local any remote $V4_REMOTEADDR ttl 64
ip link   set $TUNNEL up
ip addr   add $V6_LOCALADDR dev $TUNNEL
ip route  add 2000::/3      dev $TUNNEL
```

dove *$TUNNEL* è un nome arbitrario con cui vogliamo identificare il tunnel, *$V4_REMOTEADDR* è l'estremo remoto IPv4 del nostro tunnel e *$V6_LOCALADDR* è l'indirizzo locale IPv6 che ci è stato assegnato. Abbiamo usato il valore `any` per il parametro *local* prevedendo di avere un indirizzo IP dinamico assegnato durante il dialup e diverso ogni volta. Naturalmente dovremo trovare il modo di informare il nostro corrispondente man mano che il nostro indirizzo cambia ma questo esula dallo scopo di questo howto, anche perché questo tipo di segnalazione non è standardizzata.

Per eliminare il tunnel:

```
ip tunnel del $TUNNEL
```

rimuove automaticamente anche la regola di routing e l'indirizzo IPv6.

7.2. Comfort

A questo punto, una volta che abbiamo verificato che tutto funzioni, possiamo inserire i comandi descritti precedentemente in uno script, chiamarlo `ip-up.local` e salvarlo nella directory `/etc/ppp/`. In questo modo quelle istruzioni verranno eseguite automaticamente *ad ogni connessione*. Se vogliamo possiamo anche mettere il comando di cancellazione in un altro script, nominarlo `ip-down.local` e metterlo nella stessa directory, cosicchè *ad ogni disconnessione* il tunnel verrà cancellato.

Nel caso in cui ad esempio il tunnel venga realizzato con il tunnel broker di NGNET potremo automatizzare anche la procedura di registrazione del nostro IPv4. Ecco come potrebbe essere fatto `ip-up.local`:

```
#!/usr/bin/perl
########################################################################
# Auto-setup script for NGNET's Tunnel Broker.
########################################################################

# Configuration, fill with your values
# -------------------------------------
my $username  = '';
my $password  = '';
my $interface = '';
my $v6hname   = '';

# Don't touch anything below this line
# -------------------------------------

my $ngnet_tb  = '163.162.170.173';
my $ipv6_pref = '2001:06b8:0000:0400::/64';
my $url       = 'https://tb.ngnet.it/cgi-bin/tb.pl';

use strict;
use IO::File;
use LWP::UserAgent;

# Get our IPv4 address
my $lines;
my $f = IO::File->new();
$f->open("/sbin/ip addr show dev $interface|") or die("$!\n");
$f->read($lines, 4096);
$f->close();
$lines =~ /(\d+\.\d+\.\d+\.\d+)/ or die('Impossible condition');
my $v4addr = $1;

# Logging in
my $ua = LWP::UserAgent->new(keep_alive => 5);
my $resp = $ua->post($url, {
 oper     => 'reg_accesso',
 username => $username,
 password => $password,
 submit   => 'Submit'
```

```
});
$resp->is_success() or die('Failed reg_accesso: '.$resp->message);
$resp->as_string =~ /name=sid.*value=\"([^\"]+)\"/i or die('Missing sid');
my $sid = $1;

# Retrieve IPv6 addresses
my $myipv6;
my $ipv6end;
$resp = $ua->post($url, {
 oper        => 'tunnel_info',
 sid         => $sid,
 username    => $username,
 submit      => 'Submit'
});
$resp->is_success() or die('Failed tunnel_info: '.$resp->message);
$resp->as_string =~ /name=ipv6client.*value=\"([^\"]+)\"/i and $myipv6 = $1;
$resp->as_string =~ /name=ipv6server.*value=\"([^\"]+)\"/i and $ipv6end = $1;
die("missing IPv6 endpoints") unless ($myipv6 and $ipv6end);

# Extend tunnel lifetime
$resp = $ua->post($url, {
 oper        => 'tunnel_extend',
 sid         => $sid,
 username    => $username,
 submit      => 'Submit'
});
$resp->is_success() or die('Failed tunnel_extend: '.$resp->message);

# Update parameters on the remote side
$resp = $ua->post($url, {
 oper        => 'update_parameter',
 sid         => $sid,
 os_type     => 'Linux',
 ipv4client  => $v4addr,
 fl_entry    => $v6hname,
 username    => $username,
 ipv6_pref   => $ipv6_pref,
 submit      => 'Submit'
});
$resp->is_success() or die('Failed update_parameter: '.$resp->message);

# Set up tunnel on our side
system("/sbin/modprobe ipv6");
system("/sbin/ip tunnel add ngnet mode sit local any remote $ngnet_tb ttl 64");
system("/sbin/ip link set ngnet up");
system("/sbin/ip addr add $myipv6 dev ngnet");
system("/sbin/ip route add 2000::/3 dev ngnet");
```

`ip-down.local` si può limitare a distruggere il tunnel:

```
#!/bin/bash
/sbin/ip tunnel del ngnet
```

8. Ringraziamenti

Grazie a Giacomo Piva per la parte di integrazione con pppd e NGNET

Riferimenti

Ecco alcuni utili link per approfondire:

[IANA] Internet assigned numbers authority (http://www.iana.org/).

[ftpsite] *iproute2 ftp site (ftp://ftp.inr.ac.ru/ip-routing/)*.

[RFC2784] *Generic Routing Encapsulation (GRE) (http://www.ietf.org/rfc/rfc2784.txt)*, IETF, March 2000, Farinacci, Li, Hanks, Meyer, Traina.

[RFC2373] *IP Version 6 Addressing Architecture (http://www.ietf.org/rfc/rfc2373.txt)*, IETF, July 1998, Hinden, Deering.

[RFC2893] *Transition Mechanisms for IPv6 Hosts and Routers (http://www.ietf.org/rfc/rfc2893.txt)*, IETF, August 2000, Gilligan, Nordmark.

[NGNET] Telecom Italia Lab NGNET (http://www.ngnet.it/).