



The IPv6 Internet: Connect Today with Linux

Ibrahim Haddad



The Internet protocol version 6 (IPv6) is the next generation Internet protocol designed by the Internet Engineering Task Force (IETF) as a replacement for the Internet protocol version 4 (IPv4). Most of today's Internet uses IPv4, which has been remarkably resilient despite its age; however, it is beginning to have problems in various areas. Its most visible shortcoming is the growing shortage of IPv4 addresses needed by all the new devices connecting to the Internet. Other limitations exist in Quality-of-Service (QoS), security, auto-configuration, and mobility aspects. As a result, the IETF defined IPv6 to fix the problems in IPv4 and added many enhancements to cater to the future Internet.

Migration from IPv4 to IPv6 has been underway for a few years now, encouraged by the availability of IPv6 implementations on most operating systems and router platforms, and the availability of an IPv6 backbone that is being used for testing and deployment. In this article, I will provide a tutorial that will allow you to enable IPv6 support on your Linux machine and connect it to the IPv6 backbone (also called IPv6 Internet or the 6bone).

IPv6 Support in the Linux Kernel

You can enable support for IPv6 in the Linux kernel both as a built-in feature and as a loadable module. I will demonstrate both methods using a Linux machine installed with Fedora Core and running kernel version 2.6.0-test11 (latest version available at the time of writing).

You can either use your currently installed kernel source, or download the latest kernel from the Web. In my examples, I use the latest experimental version 2.6.0-test11, but you can use any other 2.4 or 2.6 kernel and follow the same steps.

Save the downloaded kernel package under `/usr/src` and uncompress it using the following command:

```
% cd /usr/src
```

```
% tar jxvf linux-2.6.0-test11.tar.bz2
```

This will create a directory called linux-2.6.0-test11. Next, you'll need to clean up some references to your old kernel version. Delete the symbolic link to the older kernel source tree:

```
% cd /usr/src
% rm linux
```

Then create a soft link to the new 2.6.0-test11 kernel tree:

```
% ln -s linux-2.6.0-test11 linux
```

You can also clean up any existing .o files and some older dependencies. This is true if you are recompiling an old kernel source trees that you previously compiled:

```
% cd linux
% make mrproper
```

At this point, you are ready to configure the kernel using **make config**, **make menuconfig**, or **make xconfig**:

```
% make xconfig
```

Under the "Code maturity level options", because IPv6 is still an experimental feature, you should enable the "Prompt for development and/or incomplete code/driver" option. This will allow you to activate the IPv6 support option under the "Networking support options" section.

Under the "Loadable module support options" section, the option "Enable loadable module support" is usually enabled by default. You can also enable the following options, although they are not necessary for the regular user. You can judge whether you really want them by the explanation provided:

"Module Unloading" and "Forced Module Unloading" -- Enabling this option will allow you to unload a module or force a module to unload, even if the kernel believes it is unsafe.

"Module versioning support (EXPERIMENTAL)" -- Enabling this option will make it possible to load modules compiled with a different kernel version. This is not required in this example because you are compiling the IPv6 module with the same kernel you are running. However, you can enable it.

"Automatic Kernel module loading" -- Enabling this option will allow the kernel to load modules for itself.

The last required configuration option is under the "Networking support options" section. You should enable "IPv6 protocol (EXPERIMENTAL)" either as a built-in kernel feature or as a module. Next, save the configuration and exit the kernel configuration tool to compile the kernel using:

```
% make bzImage
```

The result of the compilation is a compressed kernel image in `/usr/src/linux/arch/i386/boot/`. If you enabled support for IPv6 or other features as modules, you need to compile and install the modules using the following command:

```
% make modules && make modules_install
```

To complete your setup, copy the new kernel image `bzImage` and `System.map` to your boot directory:

```
% cp arch/i386/boot/bzImage /boot/vmlinuz-2.6.0-test11-ipv6
```

```
% cp /usr/src/linux/System.map /boot/System.map-2.6.0-test11-ipv6
```

```
% rm /boot/System.map
```

```
% ln -fs /boot/System.map-2.6.0-test11-ipv6 /boot/System.map
```

If IPv6 support was enabled as a module, the module (`ipv6.o`) will be created in `/lib/modules/linux-2.6.0-test11/kernel/net/ipv6`.

The safest method to test the new kernel without affecting your working setup is to update your boot loader with a new entry. This way if the new kernel doesn't boot correctly, you can always reboot with your working kernel and fix the problem. If you use LILO, you must add a new entry in the LILO configuration file (`/etc/lilo.conf`) for the new IPv6-enabled kernel as follows:

```
Image=/boot/vmlinuz-2.6.0-test11-ipv6
label=2.6.0-test11-ipv6
root=/dev/hda1
read-only
```

Thus, you need to make sure that the root directive references the right partition on your system. Next, run `/sbin/lilo` to install the boot loader with the new configured options in `/etc/lilo.conf`. Following these steps, you will have an entry presented to the users in LILO at boot time called `2.6.0-test11-ipv6`. On the other hand, if you use GRUB as your boot loader, you need to update `/etc/grub.conf` as follows:

```

title 2.6.0-test11-ipv6
root (hd0,0)
kernel /vmlinuz-2.6.0-test11-ipv6 ro root=/dev/hda1

```

You are now ready to reboot your Linux machine with the new kernel:

```
% shutdown -r now
```

When the boot loader prompt comes up, choose to boot with 2.6.0-test11-ipv6. After rebooting, if you compiled IPv6 as a module, you must load the module to enable IPv6 support:

```

[root@fedora-core bin]# insmod ipv6
Using /lib/modules/linux-2.6.0-test11/kernel/net/ipv6/ipv6.o
[root@fedora-core bin]

```

Testing the Setup

Now you can verify the network interfaces on your Linux machine by typing **ifconfig** at the command prompt:

```

[root@fedora-core bin]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:B0:D0:A4:A9:FA
          inet6 addr: fe80::2b0:d0ff:fea4:a9fa/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1547 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1424 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1319368 (1.2 Mb)  TX bytes:181558 (177.3 Kb)
          Interrupt:10 Base address:0xfc00
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:2348 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2348 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1672742 (1.5 Mb)  TX bytes:1672742 (1.5 Mb)

ppp0     Link encap:Point-to-Point Protocol
          inet addr:67.69.185.115  P-t-P:64.230.254.136
          Mask:255.255.255.255

```

```

UP POINTOPOINT RUNNING NOARP MULTICAST
MTU:1492 Metric:1
RX packets:1478 errors:0 dropped:0 overruns:0 frame:0
TX packets:1349 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:3
RX bytes:1282448 (1.2 Mb) TX bytes:147037 (143.5 Kb)

```

```
[root@fedora-core bin]#
```

Another test is to ping your local IPv6 interface:

```

[root@fedora-core bin]# ping6 ::1
PING ::1(::1) 56 data bytes
64 bytes from ::1: icmp_seq=0 ttl=64 time=0.071 ms
64 bytes from ::1: icmp_seq=1 ttl=64 time=0.057 ms
64 bytes from ::1: icmp_seq=2 ttl=64 time=0.058 ms

--- ::1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2015ms
rtt min/avg/max/mdev = 0.057/0.062/0.071/0.006 ms, pipe 2
[root@fedora-core bin]#

```

Another simple test is to open your browser to the KAME project Web site. If you see the dancing KAME, then you have IPv6 connectivity (see [Figure 1](#)).

The IPv6 Internet

The IPv6 Internet was established in 1996 as a test network for IPv6. Most of the backbone was established using tunnels over the current IPv4 Internet. Currently, the IPv6 Internet consists of both IPv6 native links and tunneled links over the IPv4 Internet. To connect to the IPv6 Internet, you need a provider that offers the service. If you cannot find one in your area, then an easy and free solution is to connect to the IPv6 Internet using the Freenet6 service.

Hexago (a spin-off of ViagÈnie) started the Freenet6 initiative to help people experiment and deploy IPv6. It offers a free and automated tunnel service that can connect any individual or organization to the IPv6 Internet. I will first discuss the concept of tunneling and Freenet6 connection models, and then provide instructions to help you connect your Linux machine to the IPv6 Internet.

Configured Tunnels

A configured tunnel is a standardized IETF transitional method to deploy IPv6 in coexistence with IPv4 by encapsulating IPv6 packets over IPv4. As a result, IPv6 hosts will be able to establish a link to the IPv6 Internet through an IPv4 connection.

An IPv6-over-IPv4 tunnel is established with both endpoints configuring the IPv4 and the IPv6 address of the other endpoint. When one of the endpoints changes its IPv4 address, then both endpoints of the tunnel must change their configuration accordingly. This is especially cumbersome when the IPv4 node has a dialup connection or changes addresses often.

Freenet6 Tunnel Server Protocol (TSP) provides an IPv6-over-IPv4 tunneling implementation that overcomes this problem. Each time the tunnel client changes its IPv4 address (for instance, at boot time if the host was configured for a DHCP service), the TSP client sends updated and authenticated information to the server, so the tunnel remains active without any reconfiguration.

Freenet6 Tunnel Server Protocol (TSP)

Freenet6 was the first public tunnel server service. It is the most used service in the world to delegate automatically one single IPv6 address to any host already connected to an IPv4 network over configured tunnels. The service is based on a client/server approach. It uses a protocol (TSP) that allows clients to request a single IPv6 address (for one host) or a full IPv6 prefix (for a network) from a tunnel server.

The TSP protocol is modeled after the tunnel broker as defined in RFC 3053. However, Freenet6 is an enhanced version where the node uses a tunnel setup protocol to negotiate the establishment of the tunnel with the server. The client node, in this case, can be a host or a router.

The TSP server of Freenet6 provides not only tunnels but also a large address space to any user of the service. The address space provided is a /48, which gives 216 subnets, each of which may have up to 264 nodes. This huge address space allows you to connect your networks to the IPv6 Internet and provides access to an abundance of addresses for servers and services that were not available with Network Address Translation (NAT) in IPv4.

Tunnel Establishment Session

Five steps take place to establish a tunnel session using the Freenet6 TSP between the tunnel server and the host client:

1. The IPv6 host, which has a connection to the Internet, sends a request for a tunnel to the TSP server through the TSP client program.
2. The TSP server receives the request, processes it, and assigns an IPv6 address to the requester.
3. The TSP server then establishes the IPv6-over-IPv4 tunnel according to the information sent in the request.
4. The host client receives the tunnel configuration information sent from the tunnel server.

5. The host client then configures its tunnel interface as well as its default IPv6 routes and will then have connectivity to the IPv6 Internet.

Connection Architectures

Freenet6 supports two architectures. The first is the single host architecture to connect a single host to the IPv6 Internet ([Figure 2](#)). The second is the multiple hosts architecture where you can connect a full network to the IPv6 Internet using the Freenet6 service ([Figure 3](#)). The difference between the two architectures is that in the second architecture, you need a machine that acts as an IPv6 router providing router advertisement for all the other hosts.

Freenet6 TSP Requirements

To use the Freenet6 TSP service, your Linux host should meet the following requirements:

1. The host must support IPv6.
2. The host must have a public IPv4 address because tunnel servers do not accept private addresses.
3. You must have root access on the host to install and configure the Freenet6 TSP client program.
4. If your Linux host is behind a firewall, to get IPv6 connectivity from Freenet6, firewalls and routers at the host side must allow protocol number 41 and TCP port 4343 between Freenet6 and end-users' network.
5. If your host is located behind a NAT gateway, it is not possible to get IPv6 over IPv4 traffic from any tunnel server except in the case where the NAT gateway handles static NAT addressing and the network administrator could map one Internet unicast globally unique IP address to your Linux host behind the NAT. This means the local network administrator controls and authorizes this special configuration for end users. Please note that Freenet6 is working on a NAT traversal technique and will be soon allow establishment of a tunnel over NAT without any modification of the NAT gateway.

Connecting a Single Host

If your Linux host meets the requirements listed above, you can follow these four steps to establish a connection to the IPv6 Internet:

Step 1: Creating a Freenet6 User Account

Go to <http://www.freenet6.net/register.shtml> and register for a Freenet6 user account in order to

receive a permanent IPv6 address for your Linux host. After registration, you will have a username and Freenet6 will generate a password for you and email it to the address you provided. You will use this information later when editing your tunnel configuration directives in Step 3.

Step 2: Installing the TSP Client Program

From <http://www.freenet6.net/download.shtml>, you can download the latest TSP package corresponding to your operating system and Linux distribution. I used `freenet6-0.9.8.tgz` for my test machine; however, you can also download the binary package or an RPM package. I will be using `freenet6-0.9.8.tgz` to demonstrate the procedure. After downloading the package into `/tmp`, you need to install it:

```
[root@fedora-core tmp]# tar -xzf freenet6-0.9.8.tgz
```

This will unpack the source package for the Freenet6 TSP. Next, switch to that directory and compile and build the binaries:

```
[root@fedora-core tmp]# cd freenet6-0.9.8
[root@fedora-core freenet6-0.9.8]# make install target=linux
installdir=/usr/local/tsp
```

This command will start the compilation process on a Linux machine, specified by the **target=linux** directive, and will automatically install the binaries in `/usr/local/tsp`, which is the destination directory (you can change this to a directory of your choice).

Step 3: Configuring the TSP Client

`tspc.conf` is located under `/usr/local/tsp/bin` or your own installation directory. It controls the configuration of the TSP client. You must edit it and add your registered userid and password as you received them from Freenet6 by email. Cut and paste the following to `tspc.conf`:

```
#
userid=username
passwd=????????
#
```

Step 4: Starting the TSP Client Program

Once you've finished editing the configuration file, you can start the TSP client to create an IPv6-over-IPv4 tunnel to the IPv6 Internet, as demonstrated below:

```
[root@fedora-core root]# cd /usr/local/tsp/bin
```

```
[root@fedora-core bin]# ./tspc -vf tspc.conf
tspc - Tunnel Server Protocol Client
```

```
Loading configuration file
```

```
Connecting to server
```

```
Using [67.68.56.66] as source IPv4 address.
Send request
```

```
Process response from server
```

```
TSP_HOST_TYPE          host
TSP_TUNNEL_INTERFACE  sit1
TSP_HOME_INTERFACE
TSP_CLIENT_ADDRESS_IPV4  67.68.56.66
TSP_CLIENT_ADDRESS_IPV6  3ffe:0bc0:8000:0000:0000:0000:0000:1e79
TSP_SERVER_ADDRESS_IPV4  206.123.31.115
TSP_SERVER_ADDRESS_IPV6  3ffe:0bc0:8000:0000:0000:0000:0000:1e78
TSP_TUNNEL_PREFIXLEN    128
TSP_VERBOSE            1
TSP_HOME_DIR           /usr/local/tsp
--- Start of configuration script. ---
Script:  linux.sh
sit1 setup
Setting up link to 206.123.31.115
This host is: 3ffe:0bc0:8000:0000:0000:0000:0000:1e79/128
Adding default route
--- End of configuration script. ---
Exiting with return code : 0 (0 = no error)
[root@fedora-core bin]#
```

Are We There Yet?

Let's examine the network interfaces after establishing the tunnel to see how the connection is identified:

```
[root@fedora-core bin]# ifconfig
eth0  Link encap:Ethernet  HWaddr 00:B0:D0:A4:A9:FA
      inet6 addr: fe80::2b0:d0ff:fea4:a9fa/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:3427 errors:0 dropped:0 overruns:0 frame:0
      TX packets:3282 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
```

```
RX bytes:2254537 (2.1 Mb) TX bytes:517149 (505.0 Kb)
Interrupt:10 Base address:0xfc00
```

```
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:2121 errors:0 dropped:0 overruns:0 frame:0
TX packets:2121 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:1730424 (1.6 Mb) TX bytes:1730424 (1.6 Mb)

ppp0 Link encap:Point-to-Point Protocol
inet addr:67.68.56.66 P-t-P:64.230.254.136
Mask:255.255.255.255
UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1492 Metric:1
RX packets:3314 errors:0 dropped:0 overruns:0 frame:0
TX packets:3157 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:3
RX bytes:2174778 (2.0 Mb) TX bytes:438083 (427.8 Kb)

sit1 Link encap:IPv6-in-IPv4
inet6 addr: 3ffe:bc0:8000::1e79/128 Scope:Global
inet6 addr: fe80::4344:3842/64 Scope:Link
UP POINTOPOINT RUNNING NOARP MTU:1472 Metric:1
RX packets:4 errors:0 dropped:0 overruns:0 frame:0
TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:416 (416.0 b) TX bytes:496 (496.0 b)

[root@fedora-core bin]#
```

The tunnel interface is sit1. The global IPv6 address assigned to this interface is 3ffe:bc0:8000::1e79, and the local scope IPv6 address is fe80::4344:3842. The ppp0 interface is my DSL connection.

You can ping6 some Web sites that support IPv6, such as the KAME project Web site or the 6bone.net site:

```
[root@fedora-core bin]# ping6 www.kame.net
PING www.kame.net(orange.kame.net) 56 data bytes
64 bytes from orange.kame.net: icmp_seq=0 ttl=54 time=296 ms
64 bytes from orange.kame.net: icmp_seq=1 ttl=55 time=292 ms
64 bytes from orange.kame.net: icmp_seq=2 ttl=55 time=293 ms
64 bytes from orange.kame.net: icmp_seq=3 ttl=55 time=294 ms
```

```
--- www.kame.net ping statistics ---  
4 packets transmitted, 4 received, 0% packet loss, time 61356ms  
rtt min/avg/max/mdev = 292.173/294.410/296.585/1.643 ms, pipe 2  
[root@fedora-core bin]#
```

Welcome to the IPv6 Internet!

Conclusion

IPv6 is a key technology and a long-term solution to build scalable, reliable, manageable, secure, and high-performance IP networks. In this article, I demonstrated how to configure a Linux machine running Fedora Core 1 to support IPv6 and connect it to the IPv6 Internet using the Freenet6 service. For more information on the subject, see the list of references.

References

Fedora Core -- <http://fedora.redhat.com>

Freenet6 -- <http://www.freenet6.net>

Hexago -- <http://www.hexago.com>

IETF -- <http://www.ietf.org>

IPv6 at Open System Lab -- <http://www.linux.Ericsson.ca/ipv6>

IPv6 Internet -- <http://www.6bone.net>

Linux IPv6 FAQ -- <http://www.linuxhq.com/IPv6>

Linux IPv6 HOW-TO -- <http://www.bieringer.de/linux/IPv6>

Linux Kernel -- <http://www.kernel.org>

RFC 3053 -- <http://www.ietf.org/rfc/rfc3053.txt>

ViagÈnie -- <http://www.viagenie.qc.ca>

Ibrahim Haddad is a Researcher in the Research & Innovation Unit at Ericsson Research in Montreal, Canada. He contributes regularly to Linux publications and also contributed to two of Richard

Petersen's books Red Hat Linux Pocket Administrator and Red Hat Enterprise and Fedora Edition: The Complete Reference (DVD Ed), published by McGraw-Hill/Osborne. He is currently a Dr. Sc. Candidate at Concordia University.

Copyright © 2001 Sys Admin, [Sys Admin's Privacy Policy](#). Comments about the Web site:
webmaster@sysadminmag.com