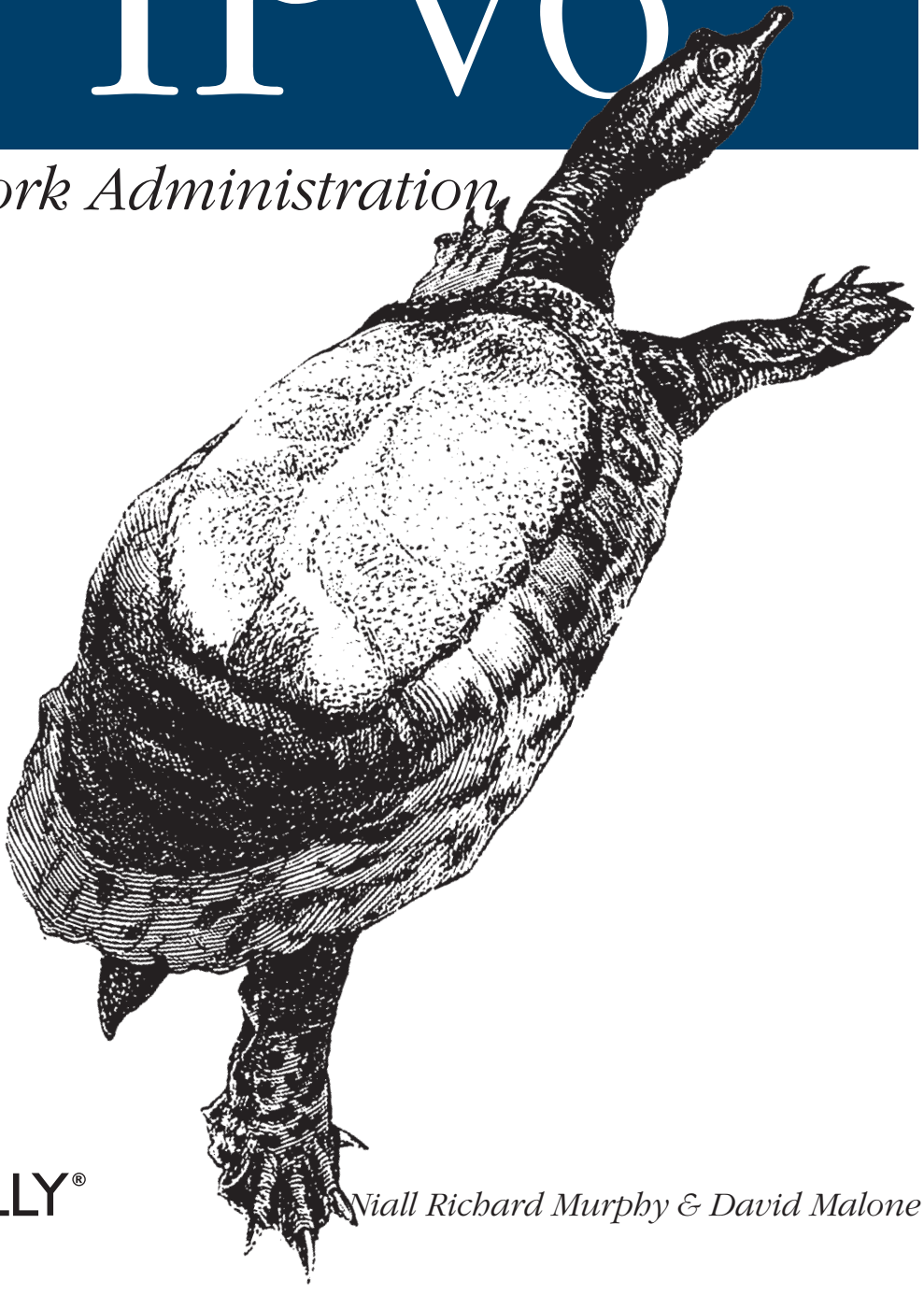
Teaching the Turtle to Dance

IPv6

Network Administration



O'REILLY®

Niall Richard Murphy & David Malone

CHAPTER 5

Installation and Configuration

Of a good beginning cometh a good end.
—Proverbes, John Heywood

We now want to look at actually configuring the IPv6 stacks on various operating platforms. First we'll describe the support present in each platform and say how to install and enable it. As the state of the art progresses, of course, the sort of instructions in this chapter should become less and less relevant, since hardly anyone needs to know how to install their IPv4 stack on their machine! Then we'll move on to the specifics of commands for testing the stack, displaying information about it and troubleshooting. This part of the chapter contains many tables showing the details of configuring the basic aspects of IPv6 on all the platforms. Tables of details rarely make exciting reading, but they are necessary because of variations between the platforms we cover. The overall aim is that, at the end of this chapter, you should have the requisite information to take a new machine from zero to hero on your IPv6 network.

We don't cover anything other than the simplest of transition mechanisms, however, so if your network relies on complicated ways to get an IPv6 connection, or if you are looking to understand how best to support IPv6 from a network manager's perspective, we advise you to look at Chapter 4, the planning chapter.

Finally, before we close the chapter we have a look at some common problems you might encounter as you take your first steps with IPv6.

Right—now it's on to the fine detail!

Workstations and Servers

In this section, we run through various workstation and server platforms, commenting on their IPv6 support and anything you may need to watch out for while enabling them. The operating systems we look at include versions of Windows, Mac OS X, and various Unix(-like) systems.

Windows

Microsoft's support for IPv6 is quite thorough, albeit relatively recent and unfortunately geared towards for their current and future products more than their past ones. Microsoft's plan for IPv6, and numerous useful articles are available at *http://www.microsoft.com/ipv6*.

Windows 2000

Windows 2000 requires the installation of the Microsoft IPv6 Technology Preview for Windows 2000, available from *http://msdn.microsoft.com/downloads/sdks/platform/tpipv6.asp*. This package creates a new protocol, unsurprisingly called IPv6, which can be manipulated and bound to various network adapters via the usual control panel interface.

The package is slightly tricky to install. Be sure to follow the instructions are included in the FAQ referenced on the page mentioned above. Note that the procedure is service pack specific and may need to be manually reinstalled after a service pack upgrade. Also, Microsoft consider the patch a technology preview and do not recommend running it in a production environment. For these reasons, Windows XP or Windows 2003 are a better choice for running IPv6 on a Windows platform.

Windows XP

Windows XP comes with IPv6 support by default, though you do need to enable it manually. Easily done: you open a command prompt and issue the command `ipv6 install`. Windows XP Service Pack 1 also supports installing IPv6 via the `Network Connections` control panel. Officially, the stack shipped with Service Pack 1 is of production quality, and the earlier versions are developer previews. Despite this, the stack shipped with Service Pack 1 identifies itself as a developers edition. This is slightly confusing but not actually harmful. Service Pack 2 extends this support even further, including an IPv6 firewall by default and Toredo which allows IPv6 through NAT.

Microsoft has some useful information about the capabilities and configuration of the stack shipped with XP available from it's IPv6 pages at *http://www.microsoft.com/ipv6* but most of the configuration can be done with the `ipv6` command, with finer control over the stack available using `netsh`. Basic testing of IPv6 connectivity can be accomplished with `ping6` and `tracert6`.

There is one peculiarity however: some versions of Windows will automatically configure routing via 6to4 if a global IPv4 address is found and no IPv6 router is present on the LAN. This has caught some people by surprise.

Windows Server 2003

Windows Server 2003 also has IPv6 support, but it goes beyond what's included in Windows XP—the IPv6 stack is a full stack nearly on a par with its IPv4 cousin. The

`ipv6` command is also being deprecated in Server 2003, and the equivalent `netsh` commands are now preferred. Microsoft provides a handy crib sheet of `ipv6` and `netsh` equivalent commands at *http://www.microsoft.com/windowsserver2003/technologies/ipv6/ipv62netshtable.mspx*.

Again, IPv6 can be enabled via the command line by running `netsh interface ipv6 install`, or from the `Network Connections` control panel (right click on a LAN interface to edit its Properties → Install → Protocol → Add → Microsoft TCP/IP Version 6).

Support for IPv6 `ping`and `traceroute` are also available in the traditional `ping` and `tracert` commands without the "6" suffix.

Other versions of Windows

If you run any variety of Windows 98 or NT 4, then, at the moment, you are out of luck. Since these products have been end-of-lifed, it is unlikely that useful IPv6 support for them will be forthcoming from Microsoft. However, third party support is available via products such as Trumpet Winsock* and Hitachi's Toolnet6.†

A stack for Windows CE.NET is available. One interesting question is whether or not we will see IPv6 on the X-Box. You would expect that non-NATed peer-to-peer gaming would be a core attraction of IPv6, but we'll have to wait and see.

IPv6 applications on Windows

As mentioned, all versions of the Microsoft IPv6 stack come with the basic diagnostic tools like `ping` and `traceroute`. Some versions also ship with a tool called pathping, which is an interesting cross between `traceroute` and `ping`. It does an initial traceroute and then calculates statistics relating to round-trip-time and loss.

The old command-line tools, `telnet` and `ftp`, also support IPv6. For applications, such as Internet Explorer, that use `wininet.dll`. IPv6 support should be essentially transparent, particularly for those applications using DNS rather than endpoints specified by explicit IPv6 address.

Microsoft's P2P update for Windows XP‡ is a platform for the development of peer-to-peer applications. It makes heavy use of IPv6 and even provides a personal IPv6-enabled firewall. There are also other sites that provide IPv6 enabled versions of Windows software, such as *http://win6.jp/*.

* *http://www.trumpet.com.au/ipv6.htm*

† *http://www.hitachi.co.jp/Prod/comp/network/pexv6-e.htm*

‡ Currently a beta version is available for download from *http://msdn.microsoft.com/library/default.asp?url=/downloads/list/winxppeer.asp*, Windows XP Service Pack 2 includes some of the features of the P2P update.

Points of interest

IPsec on top of IPv6 within Windows XP and Windows 2003 is missing one or two features currently. ESP payload encryption is not available in general, though it *is* available for tunnels. Automatic key configuration with IKE is also not available, so IPsec policies must be configured manually with ipsec6.exe, using preshared keys.

At time of writing, Microsoft's IPv6 implementations do not support mapped IPv4 addresses.* While Internet Explorer will support both protocols simultaneously, it does mean that cross-platform applications, such as some versions of Mozilla, that use mapped addresses must disable IPv6 support or lose IPv4 compatibility. For services such as Apache this is not a problem, as they can listen for IPv4 and IPv6 connections independently.

Microsoft has moved the operation of CIFS (or to use another acronym, the SMB file-sharing service) to use port TCP port 445 exclusively over IPv6.† IPv6-based SMB requests from non–on-link addresses seem to be automatically refused; this must be applauded as a useful security measure for unmanaged networks.

Macintosh (OS X and Darwin)

The Unix-like layer, Darwin, that underlies Mac OS X supports IPv6 as of version 10.2 (Jaguar) and automatic configuration is enabled by default. While many of the lower level Darwin utilities support IPv6, this has not yet percolated upwards to most familiar Mac OS applications. In essence, this means that most of the command line tools support IPv6, including `ping6`, `traceroute6`, `telnet` and so on, but things like iSync, iPhoto, etc., don't necessarily. One thing that's missing in 10.2 is IPv6 support in `ssh`.‡ Naturally, OS X's IPv6 support derives from the KAME project, so resources and documentation for KAME will apply usefully to OS X. Version 10.3 of Mac OS X is based on FreeBSD 5.x and also supports IPv6.

Panther (OS X version 10.3) extends the IPv6 support introduced in Jaguar into the OS X network control panels and also into a number of subsystems, including allowing DNS lookups over IPv6, IPv6 personal firewalling and IPv6 support in `ssh`.

As far as we're aware, you're out of luck if you want to run IPv6 on Mac OS 9 or anything earlier.

* See the "Mapped IPv4 Addresses" section in Chapter 8 to find out more about mapped addresses.

† Prototype patches allowing Sambato speak SMB over IPv6 are available from *http://v6web.litech.org/samba*, although this work has yet to be brought into a mainstream Samba release.

‡ The authors just compile a version of OpenSSH and keep it handy for IPv6-only occasions.

Linux

IPv6 in the Linux kernel has a slightly uneven history. Initial support began in 1996, with contributions by Pedro Roque, who later went on to work for Cisco. Underresourcing got the better of developer effort some while thereafter, and the stack quality suffered, with the result that a project called USAGI was started in Japan in late 2000, whose aim was to bring the kernel implementation up to spec with the reality of what the RFCs required. Thankfully things these days are a lot better; most of the Linux vendors have brought their stack into shape with the relevant USAGI patches, and if you are running a 2.4.x (or better) kernel, many of the more egregious faults with 2.2.x are no longer a problem. The USAGI patches provide things like ICMPv6 node information queries, IPsec support, and fix a number of bugs. If any of these are important to your network, you may want to investigate applying these patches.*

To get IPv6 working with Linux, you must first distinguish between the kernel and the distribution or userland that you happen to be running. All modern kernels (=>2.2, but you really want =>2.4) support IPv6—you can either compile it into the kernel statically, following the standard Linux kernel compilation instructions, or use a module. Most modern Linux software vendors will ship this as the module *ipv6.o*. The kernel module supplies the ability to actually speak the protocol; the userland tools supply the ability to work with it. It is unfortunately possible, although unlikely, to have kernel support but no userland support, and vice versa.

A lack of userland support is the easiest problem to remedy: simply download the relevant RPMs and install them.

If you are missing kernel support and the *ipv6.o* module is not provided then you will have to recompile your kernel.† Recompiling your kernel is something that your Linux distributor should provide documentation for. In general, it involves going to where your kernel sources are, generally */usr/src/linux*, typing `make menuconfig`, selecting IPv6 under `Networking Options`, saving your changes and then doing `make bzImage`, but the your vendor's documentation should be your guide here. One complication you might encounter is that IPv6 may be marked "Experimental" and hence might not be shown as a selectable option unless you indicate that you want to see experimental options under `menuconfig`'s `Code Maturity Level Options`. Note that while you are adding IPv6 support, you may also want to enable the IPv6 firewalling support (a.k.a. netfilter/iptables) as well.

* Of course the state of the art will move on, and they may make their way into mainstream kernel deployment eventually.

† Another reason you might want to recompile your kernel is to apply some of the USAGI patches from *http://www.linux-ipv6.org/*. These are for the more expert user and aren't required for normal operation.

If you are wondering whether your current kernel has IPv6 support, there are two quick tests you can do. If you are using a software vendors distribution, try the simple `modprobe ipv6` that should load the module in question (`lsmod | grep -w ^ipv6` should report the presence of the module if you want to be extra sure). *Make sure previous command has a hat-ipv6 in it.* If that doesn't work, perhaps because the kernel has it statically compiled in, then check out the contents of */proc/net/*—network protocols register their presence here when they are loaded, so *if_inet6* and *igmp6* will be present if the kernel had IPv6 compiled in.*

Of course the kernel itself has some knobs allowing you to change its IPv6 behavior more to your satisfaction. Possibly the most useful of these is being able to turn off address autoconfiguration on a per-interface basis by running `echo 0 > /proc/sys/net/ipv6/conf/eth0/autoconf` where `eth0` can be replaced by the relevant interface name. This only disables address configuration, but other information like default routes can still be learned from router advertisement packets. You can more completely disable the processing router advertisements with `echo 0 > /proc/sys/net/ipv6/conf/eth0/accept_ra`. Both of these commands have a system-wide equivalent, but we've found it simpler and more reliable to use the per-interface settings.

The first place to go to if you want to find out more is Peter Bieringer's wonderful IPv6 resources at *http://www.bieringer.de/linux/IPv6/*, which provide not only useful resources for IPv6 users under Linux, but also a wealth of information about IPv6 support in various applications and services on all Unix-like platforms.

We deal with the different distributions below.

Red Hat and derivatives

Enabling IPv6 on recent Red Hat-derived Linux systems is as easy as adding the line:

```
NETWORKING_IPV6="yes"
```

to */etc/sysconfig/network*. This should configure the boot-time scripts to load the IPv6 kernel module, *ipv6.o* if required, and enable autoconfiguration of network interfaces. Manual configuration of the interface address is covered in the "Enabling, Testing, and Troubleshooting" section later in this chapter.

Fedora Core, the community-maintained version of Red Hat, activates IPv6 in the very same way.

SuSE

Support for IPv6 varies widely across SuSE distributions. We will focus on the 8.x series here, since they were the most recent output from SuSE at time of writing.

* This provides a good way to check for IPv6 support in a script: `test -d /proc/sys/net/ipv6`.

SuSE 8.0 startup scripts do not support non-autoconfigured interfaces yet; hard-wired addresses must be put into some script executed at bootup (such as */etc/rc.d/boot.local*).

In general, one drives the SuSE 8.x distributions by editing */etc/sysconfig/network/ifcfg-eth0* (where one is attempting to configure interface `eth0`) and inserts the line

```
IP6ADDR="<IPv6 address>/<prefix length>"
```

Debian

The key configuration file for IPv6 support in Debian is */etc/network/interfaces*. We include some configuration file examples below, that serve to illustrate how IPv6 is configured:

```
iface sit1 inet6 v4tunnel
        address <your end>
        netmask <tunnel netmask>
        endpoint <tunnel broker IPv4 address>
        up ip route add 2000::/3 via <their end>
```

This brings up a tunnel between the nominated places.

```
iface eth0 inet6 static
        pre-up modprobe ipv6
        address 2001:db8:1234:5::1:1
        netmask 64
```

This is a static configuration for your local Ethernet interface.

Note that many of the examples later in this chapter use the `ip` command, which is not installed on Debian by default. To get this command you can `apt-get install iproute`.

Userland/administration support for IPv6

Simple tools like `ping6`are supplied with most modern distributions. Since they're useful for testing, if your distribution doesn't have them we would recommend that you install them from your OS vendor supplied material, or download them. Here's a list of common distributions and the names of the RPMs, together with where to get them:

Red Hat 8+

While Red Hat has shipped `ping6`since sometime around Red Hat 6.2, we'll consider version 8 onwards. The iputils RPM that is distributed with Red Hat 8 and newer contains `ping6` and `traceroute6`. Iputils also has a `tracepath` command, which is similar to `traceroute` but also provides path MTU information. Unfortunately, the Kerberos version of `telnet` and `ftp` that ships with Red Hat 8 does not seem to support IPv6. One option here is to remove */usr/kerberos/bin* from your path.

Debian

Debian also includes good userland support for IPv6. `ping6` and `traceroute6` can be found in iputils-ping and iputils-tracepath respectively, and the normal version of `telnet` supports IPv6.

SuSE 8.x

The normal networking RPMs contain all the commands that you are likely to need.

Solaris

From Solaris 8 onwards, IPv6 is included in the normal Solaris installation process, and you are asked if you want to configure IPv6 during the install. There is good coverage of both IPsec and IPv6 in the networking sections of the Solaris Administration Guide, available online at *http://docs.sun.com/*.

Sun have always been advocates of NIS, and have extended the Solaris Name services to deal with IPv6. The traditional */etc/hosts* database, which is actually a symbolic link to */etc/inet/hosts*, is only used for IPv4 addresses in Solaris. A new database, */etc/inet/ipnodes*, can be used for *both* IPv4 and IPv6 name lookups: which of these is used can be controlled with settings in */etc/nsswitch*. If the hosts database is commented out, the ipnodes database will be used for all lookups.

People familiar with Solaris may remember that the IPv4 address for an interface is stored in the file */etc/hostname.ifname*. Similarly, the IPv6 configuration of an interface is controlled by */etc/hostname6.ifname*. As with IPv4, this file can contain a numerical address, or a hostname to be looked up using the Solaris name service. It is also possible to leave this file empty, which will cause the interface to use IPv6 autoconfiguration. Autoconfiguration is managed by the `in.ndpd` daemon, which sends Router Solicitation messages and acts on the Router Advertisements received.

The `ping`and `traceroute` commands both support IPv4 and IPv6. Specifying an IPv4 address causes these commands to use IPv4. Specifying an IPv6 address causes these commands to use IPv6. Specifying a hostname causes the commands to use IPv6, if the host has an IPv6 address and IPv4 otherwise. You can explicitly choose address family by using the flags `-A inet4` and `-A inet6` respectively.

There are IPv6 patches for some earlier versions of Solaris available from Sun, however these were considered "developer" quality.

AIX

IPv6 should be available in AIX from version 4.3.3 onwards. Autoconfiguration can be enabled from AIX's SMIT configuration tool under the following menus: Communications Applications and Services → TCP/IP → IPV6 Configuration → IPV6 Daemon/Process Configuration → Autoconf6 Process → Start Using the Autoconf6

Process. In addition, you will want to enable `ndpd-host`, also available under the IPV6 Daemon/Process Configuration menu.

AIX's version of `ping` and `traceroute` includes support for IPv6. Utilities such as `telnet` also include IPv6 support. The IPv6 support in AIX is based on work at INRIA.

Although not strictly related to AIX, IBM also offer a prototype IPv6 implementation for OS/390.

Tru64

Version 5.1 of Tru64 contains basic IPv6 support. The first step is to make sure your kernel supports IPv6; if you've built or installed a kernel with support for *all* optional features, then it will contain IPv6 support. Otherwise, you'll need to configure your kernel with `doconfig -c KERNELNAME`, choose to include the IPV6 option, install it with `cp /sys/KERNELNAME/vmunix /vmunix` and reboot.

A script, `/usr/sbin/ip6_setup`, is provided to make enabling IPv6 easier. It will ask you if you have network interfaces on which you want to enable IPv6, and if you want to configure tunnels for IPv6 connectivity. As a minimum, you can tell it to configure IPv6 on your Ethernet interface, probably `tu0`, then tell it to save the changes and start IPv6 networking.

The usual `ping` and `traceroute` commands support IPv6 in Tru64, with a flag `-V 4` or `-V 6` to determine the version of IP to use (IPv6 is the default for hostnames with both types of addresses). Other base utilities such as `telnet` and `ftp` support IPv6. The version of `ssh` shipped with Tru64 5.1 seems to support IPv6, but prefers IPv4 DNS records over IPv6, so you need to give explicit IPv6 addresses on the command line, or only have quad A records in your internal DNS for servers to which you want to `ssh` using IPv6.

The Tru64 Network Administration Manual contains both an introduction to IPv6 and details of how it can be configured under Tru64.

FreeBSD

The IPv6 support in FreeBSD is based on the work by the KAME group. Initially it was available as a set of patches to FreeBSD, but IPv6 has been a shipping feature of the FreeBSD distribution for some time, and is included in the standard 'GENERIC' kernel. In fact, it is possible to install FreeBSD over IPv6 if you choose an IPv6 enabled FTP server during the setup process.

If, for some reason, IPv6 is not present in your kernel you will need to recompile it after adding the `options INET6` line to your kernel configuration (full details of how to recompile your kernel are in the FreeBSD handbook at *http://www.freebsd.org/handbook/*).

IPsec is also incorporated into FreeBSD and more recent releases include support for hardware acceleration of IPsec. However, IPsec is not part of the GENERIC kernel, and may require a kernel recompilation after the addition of `options IPSEC` and `options IPSEC_ESP` to your kernel's configuration file.

Most of the FreeBSD base applications support IPv6 including `ssh`, `telnet`, `ftp`, `sendmail`, and `inetd`. Where possible, additional software from the FreeBSD ports/packages system is compiled with IPv6 support; there's even a ports category specifically for IPv6 software!

To enable the boot-time configuration of IPv6 on FreeBSD, you must add a line `ipv6_enable="YES"` to your */etc/rc.conf* file. Other configuration options for the setting up of tunnels, routing and so on, are listed in */etc/defaults/rc.conf* under the "IPv6 options" heading.

Other Workstation/Server OSs

Naturally, the list of operating systems that now support IPv6 goes on and on. The list above is just a sample of the operating systems commonly associated with IP networking, biased by the authors' experience. Let's take a moment to glance at some of the other OSs in this area.

BSDi, NetBSD, and OpenBSD certainly warrant a mention, as they are other platforms based on the KAME IPv6 code and have supported IPv6 for several years. On these platforms, the command line utilities will be similar to those on FreeBSD and Mac OS X, though boot-time configuration knobs will differ slightly. It is also worth noting that if you require up-to-the minute IPv6 features, KAME provide *snap kits* of their development work for various platforms. These are available from the KAME web site *http://www.kame.net/* but are strictly for the courageous expert.

IPv6 support for SGI's Irix has been available as beta release for some time to people with support contracts. Since February 2003 it has been available in the normal releases of Irix 6.5.19 and above. Similarly, IPv6 is available for HP-UX 11i from the HP web site, *http://www.hp.com/products1/unix/operating/internet/ipv.html*. The IP stack shipped with VMS and Multinet both support IPv6.

Routers

In this section, we look at Cisco's and Juniper's support for IPv6. They are not the only vendors in the IPv6 market, but they are two vendors that many people will be familiar with.

Cisco

Cisco detail support for IPv6 across their various routers on their web site at *http://www.cisco.com/ipv6/*. However, IPv6 performance varies from platform to platform.

For software-based routers, such as the 2500, 2600 and 7200, it is possible to achieve respectable IPv6 performance at the current time with just an IOS upgrade. Maximum speeds are not yet on a par with IPv4, and many of the more complex features haven't been implemented for IPv6 yet, but this should improve as releases continue.

Cisco IOS support for IPv6 first appeared on the public radar with the 12.2T stream. As a "new technology" release, some readers may be familiar with the T streams if they are using other new features. Like all new technology, it has a rather large memory footprint, and may contain other new features that one may not wish to deploy in production yet.

The 12.3 stream is the first mainline IOS stream with support for IPv6. Cisco provides TAC support for the IPv6 features of 12.3. This is also a fairly hefty piece of code, and older equipment might require memory upgrades in order to support it.

An alternative is the 12.2S stream which appeared in 2004. It is aimed at service providers who wish to use IPv6 in a smaller package more tuned to their needs, or who are reluctant to make the leap to 12.3 mainline for a single feature. At the time of writing, this is also the stream on which IPv6 support for layer 3 switches, such as the Catalyst 6500/Cisco 7600, is based.

For hardware-based routers, such as the 12000 series, the situation is rather different. The 12.0S stream of code now supports IPv6 in its more recent incarnations, and a number of ISPs are using this in their backbones. However, performance of these routers is dependent on hardware support in the linecard, not the central routing processor, and this could mean expensive upgrades. Engine 3 linecards support IPv6 in hardware; Engine 0, 1, 2 and (perhaps surprisingly) 4+ linecards only support IPv6 with software forwarding, with a much lower throughput. Engine 5 linecards were promised but not yet available at the time of writing, so check with your equipment vendor for more details on those.

To discover the types of linecard already deployed in a 12000 series router, use the `show diag` command—it is listed for each slot on the line starting "L3 Engine:".

If you have a mix of cards, the throughput you may achieve depends on the input interface; if it's capable of native IPv6, you should achieve reasonable traffic rates, regardless of the output interface. If it's not, the packet will be forwarded in software, with a much lower maximum throughput. For those providers who use MPLS in their backbone, Cisco suggests 6PE, a method of transiting traffic from a dual-stacked provider edge (presumably based on 7200/7500 routers or similar) over an IPv4-only core, using MPLS tunnels. This might be an excellent workaround for an

organization already familiar with MPLS, but those who aren't already using MPLS might think twice about deploying it solely for IPv6.

IPv6 configuration on Cisco is typically straightforward, especially if one is already familiar with the procedure in IPv4. IOS commands are generally derivable from the names of their predecessors by the simple expedient of replacing "ip" with "ipv6" (`s/ip/ipv6/` for all you regexp fans). This works for commands like `show ipv6 route` and `show ipv6 interface`. However, there are ipv6 specific commands like `show ipv6 neighbors` and `ipv6 unicast-routing`.*

In BGP land, there have been some subtle changes; `sh ip bgp` becomes `sh bgp ipv6`. Also, when you configure your first BGP session over IPv6, you might get a bit of a shock when you look over your configuration; the IPv4-specific parts are moved automagically into their own section. We'll deal with this in more detail when we discuss routing in the "Routing Protocols" in Chapter 6.

Juniper

Juniper have been offering IPv6 support in JUNOS for some time; most of the features arrived in JUNOS 5.1 or JUNOS 5.2. This support covers the core parts of IPv6: the protocol itself, forwarding, IPv6 over various media and the all routing protocols you'd expect. Hardware support extends to all Juniper's platforms and interface cards.

Again, the obvious commands are fairly similar to their IPv4 equivalents—or in many cases, show IPv6 information alongside IPv4. Like Cisco, `ping` on Juniper will attempt IPv6 if it is available. Unlike Cisco, `show bgp summary` lists IPv4 and IPv6 sessions in sequence.

To configure an IPv6 address on an interface, one substitutes `family inet6` for `family inet`. Example 5-1 shows the configuration of a dual-stacked Fast Ethernet interface on a Juniper router.

Example 5-1. Configuring a Fast Ethernet interface on a Juniper

```
interfaces{
    fe-1/0/0 {
        description "HEAnet Cork PoP LAN";
        unit 0 {
            family inet {
                address 193.1.199.75/26;
            }
            family inet6 {
                address 2001:0770:0800:0003::1/64;
```

* Don't forget to turn this on. Everyone forgets to turn this on. Everyone is then surprised when the routing protocols come up but traffic isn't forwarded or router advertisements aren't sent. Some people get quite a distance into a support call before realising that they forgot to turn this on.

Example 5-1. Configuring a Fast Ethernet interface on a Juniper (continued)

```
            }
        }
    }
}
```

One important note about Juniper's IPv6 support is that if you plan to use IPv6 tunnelled over IPv4 (configured tunnels, 6to4 etc.), then you'll need to have a suitable processor to do the encapsulation and decapsulation. For example, the devolved architecture of M-series routers doesn't allow the router's CPU to get bogged down in intensive tasks like forwarding packets over tunnels. You may need an a *tunnel services PIC* or a *adaptive services PIC*. Some routers, such as the T-series, don't need extra hardware.

Enabling, Testing, and Troubleshooting

In this section we'll go through the particular steps required to enable and test IPv6 on a host, including showing tables of the relevant commands.

On some systems, IPv6-aware utilities are shipped with a suffix of "6," so `ping`becomes `ping6` and so on. On others systems, IPv6 operation is selected based on the name/address given. If you give a name that has both IPv4 and IPv6 addresses associated with it then there is usually a flag to allow you to explicitly select which protocol you want to use. There are, unfortunately, degrees of variation between systems which merely begin with the naming of commands. Consequently, these tables should serve as a useful phrase-book.

In general, utilities also live in the same directory as their IPv4 counterparts, and hence would tend to be in your `PATH` (we only include the full path to a command if it is in some unusual location).

Turning on IPv6

Table 5-1 show a summary of how to enable IPv6 at boot on the various operating systems we are considering. Once IPv6 is enabled, the boot-time behavior of most platforms is to perform autoconfiguration, unless they are explicitly configured otherwise. As it may not be easy to restart the network subsystem to initialize IPv6, Table 5-2 shows how to configure IPv6 and enable autoconfiguration while the system is actually running. This may be useful during your initial experimentation. In fact, on some systems, squeezing variations on these commands into a user-editable part of the boot sequence is the only way to introduce persistent IPv6 configuration. *Caveat configurator*.

Table 5-1. Boot time enabling of IPv6 with autoconfig

OS	Enable IPv6 at boot (with autoconf where possible)
Solaris	Create an empty */etc/hostname6.ifname*
Red Hat	Add `NETWORKING_IPV6="yes"` to */etc/sysconfig/network*.
AIX	Use `smit` or `chrctcp` to enable `autoconf6` and `ndpd-host` under: Communications Applications and Services → TCP/IP → IPV6 Configuration → IPV6 Daemon/Process Configuration.
WinXP	`ipv6 install`
Win2003	`netsh interface ipv6 install`
FreeBSD	Add `ipv6_enable="YES"` to */etc/rc.conf*.
Mac OS X	Enabled by default (see */etc/hostconfig*).
Tru64	Use `ip6_setup` to start IPv6 on an interface or edit */etc/rc.config* directly.

IOS	`conf term`
	`interface` *`if number`*
	`ipv6 enable`
	You may also want `ipv6 unicast-routing`.

JUNOS	`set interfaces` *`if`* `unit` *`no`* `family inet6 address` *`addr`*

Table 5-2. Runtime enabling of IPv6 with autoconfig

OS	Runtime IPv6 enable (with autoconf where possible)
Solaris	`ifconfig` *`ifname`* `inet6 plumb up` and then run `/usr/lib/inet/in.ndpd`.
Linux	Load kernel module with `insmod ipv6` then `sysctl net.ipv6.conf.`*`ifname`*`.accept_ra=1`.
AIX	`autoconf6 -a` followed by `ndpd-host`.
WinXP	`ipv6 install`
Win2003	`netsh interface ipv6 install`
FreeBSD	`sysctl net.inet6.ip6.accept_rtadv=1`
Mac OS X	`sysctl -w net.inet6.ip6.accept_rtadv=1`
Tru64	Ensure kernel contains IPv6, `ifconfig` *`ifname`* `ipv6 up` and then run `nd6hostd`.

IOS	`conf term`
	`interface` *`if number`*
	`ipv6 enable`
	You may also want `ipv6 unicast-routing`.

JUNOS	`set interfaces` *`if`* `unit` *`no`* `family inet6 address` *`addr`*

Of course, if you do not have an IPv6 router on your network, autoconfiguration isn't much use. You can do the initial testing with link-local addresses, but manually configuring addresses may be more satisfactory. Tables 5-3 and 5-4 show how to manually configure addresses on a variety of systems, at boot time and while they are running.*

* If you are manually configuring a system at runtime, remember that you may need to configure the loopback interface by assigning it address `::1`. If you enable IPv6 at boot time, this will usually be taken care of for you.

Table 5-3. Manual IPv6 addressing at boot time

OS	Manual assignment of address at boot
Solaris	Add hostname and IPv6 address to */etc/inet/ipnodes* and then put hostname in */etc/hostname6.*`ifname`.

Red Hat	Add
	`IPV6INIT="yes"`
	`IPV6ADDR="2001:db8::1/64"`
	to */etc/sysconfig/network-scripts/ifcfg-*`ifname`.

AIX	Use the Communications Applications and Services → TCP/IP → IPV6 Configuration → IPv6 Network Interfaces menu in `smit` to set the address, or use `chdev` to set the "netaddr6" attribute on the interface.
WinXP	`ipv6 adu` *`ifindex`*`/2001:db8::1`
Win2003	`netsh interface ipv6 add address interface=`*`ifindex`* `2001:db8::1`
FreeBSD	Add `ipv6_ifconfig_`*`ifname`*`="2001:db8::1 prefixlen 64"` to */etc/rc.conf*.
Mac OS X	No specific technique, but could use Startup Items.
Tru64	Can be set using `ip6_config` or using `IP6IFCONFIG_`, `NUM_IP6CONFIG`, and `IP6DEV_` in */etc/rc.config*.

IOS	`conf term`
	`interface` *`if number`*
	`ipv6 address 2001:db8::1/64`

JUNOS	`set interfaces` *`if`* `unit` *`no`* `family inet6 address 2001:db8::1/64`

Table 5-4. Manual IPv6 addressing at runtime

OS	Manual assignment of address at runtime
Solaris	`ifconfig` *`ifname`* `inet6 addif 2001:db8::1/64 up`
Linux	`ip addr add 2001:db8::1/64 dev eth0`
AIX	`ifconfig` *`ifname`* `inet6 2001:db8::1/64`
WinXP	`ipv6 -p adu` *`(ifindex)`*`/2001:db8::1`
Win2003	`netsh interface ipv6 add address interface=`*`ifindex`* `2001:db8::1`
FreeBSD	`ifconfig` *`ifname`* `inet6 2001:db8::1 prefixlen 64 alias`
Mac OS X	`ifconfig` *`ifname`* `inet6 2001:db8::1 prefixlen 64 alias`
Tru64	`ifconfig` *`ifname`* `ipv6` `ifconfig` *`ifname`* `inet6 2001:db8::1`

IOS	`conf term`
	`interface` *`if number`*
	`ipv6 address 2001:db8::1/64`

JUNOS	`set interfaces` *`if`* `unit` *`no`* `family inet6 address 2001:db8::1/64`

The first thing to check on a host is what IPv6 addresses are automatically configured. Either use the commands in Table 5-1 to enable IPv6 with autoconfiguration at boot, or the commands in Table 5-2 to enable IPv6 at runtime. Then display the configured addresses using the commands outlined in Table 5-5. The link-local addresses, beginning `fe80::`, should be available and the loopback address `::1` will also be available. If you see any `2001::`, `3ffe::` or `2002::` addresses and you are sur-

prised by their presence, then either a tunnel has been automatically configured, or someone has set up an IPv6 router on your network unbeknownst to you.

Example 5-2 shows the interface configuration on a FreeBSD host and Solaris host using the `ifconfig` command. They both have link-local addresses and autoconfigured `2001::` addresses provided by the local router. Note, FreeBSD assigns all the addresses to a single interface, while Solaris uses sub-interfaces and displays different families of addresses separately. These are cosmetic differences that have no real impact on the operation of IPv6. Note there are other minor differences such as how the prefix length or scope information is displayed.

Example 5-2. Displaying interface configuration

```
freebsdhost% ifconfig -a
dc0: flags=8843&lt;UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST&gt; mtu 1500
        inet 10.0.0.1 netmask 0xffffff00 broadcast 10.0.0.255
        inet6 fe80::204:e2ff:fe33:e3ac%dc0 prefixlen 64 scopeid 0x1
        inet6 2001:db8:babe:1:204:e2ff:fe33:e3ac prefixlen 64 autoconf
        ether 00:04:e2:33:e3:ac
        media: Ethernet autoselect (100baseTX &lt;full-duplex&gt;)
        status: active
lo0: flags=8049&lt;UP,LOOPBACK,RUNNING,MULTICAST&gt; mtu 16384
        inet6 ::1 prefixlen 128
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x2
        inet 127.0.0.1 netmask 0xff000000


solarishost% ifconfig -a
lo0: flags=1000849&lt;UP,LOOPBACK,RUNNING,MULTICAST,IPv4&gt; mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
le0: flags=1004843&lt;UP,BROADCAST,RUNNING,MULTICAST,DHCP,IPv4&gt; mtu 1500 index 2
        inet 10.0.0.15 netmask ffffff00 broadcast 10.0.0.255
        ether 8:0:20:72:74:9e
lo0: flags=2000849&lt;UP,LOOPBACK,RUNNING,MULTICAST,IPv6&gt; mtu 8252 index 1
        inet6 ::1/128
le0: flags=2000841&lt;UP,RUNNING,MULTICAST,IPv6&gt; mtu 1500 index 2
        ether 8:0:20:72:74:9e
        inet6 fe80::a00:20ff:fe72:749e/10
le0:1: flags=2080841&lt;UP,RUNNING,MULTICAST,ADDRCONF,IPv6&gt; mtu 1500 index 2
        inet6 2001:db8:babe:1:a00:20ff:fe72:749e/64
```

Table 5-5. Displaying IPv6 interface information

OS	Showing configured addresses
Solaris	`ifconfig -a`
Linux	`ifconfig -a`
AIX	`ifconfig -a`
WinXP	`ipv6 if`
Win2003	`ipconfig`
FreeBSD	`ifconfig -a`

Table 5-5. Displaying IPv6 interface information (continued)

OS	Showing configured addresses
Mac OS X	`ifconfig -a`
Tru64	`ifconfig -a`
IOS	`show ipv6 interface`
JUNOS	`show interfaces`

Table 5-6. Basic IPv6 diagnostic tools (including interface specifier flag for link-local addressing)

OS	ping	traceroute
Solaris	`ping -A inet6 -i` *if*	`traceroute -A inet6`
Linux	`ping6 -I` *if*	`traceroute6`
AIX	`ping`	`traceroute`
WinXP	`ping6`	`tracert6`
Win2003	`ping`	`tracert`
FreeBSD	`ping6 -I` *if*	`traceroute6`
Mac OS X	`ping6 -I` *if*	`traceroute6`
Tru64	`ping -V 6 -I` *if*	`traceroute -V 6`
IOS	`ping ipv6`	`traceroute ipv6`
JUNOS	`ping inet6`	`traceroute inet6`

Testing with ping and telnet

Initially, the most useful test you can make is to check that you can ping localhost. Check the ping command in Table 5-6 and try pinging `::1` and any link-local addresses that are configured. To ping link-local addresses, you may need to specify the interface to use. This can usually be done with an option to ping or by giving a scope ID in the address (as supported by KAME and Microsoft stacks). See Example 5-3. Scope IDs are explained as part of "Notation" in Chapter 3; see "Scope identifiers" in Chapter 3.

Example 5-3. Output from pinging ::1

```
% ping6 ::1
PING6(56=40+8+8 bytes) ::1 --&gt; ::1
16 bytes from ::1, icmp_seq=0 hlim=64 time=0.537 ms
16 bytes from ::1, icmp_seq=1 hlim=64 time=0.381 ms
16 bytes from ::1, icmp_seq=2 hlim=64 time=0.384 ms
16 bytes from ::1, icmp_seq=3 hlim=64 time=0.384 ms
^C
--- ::1 ping6 statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/std-dev = 0.381/0.421/0.537/0.067 ms
```

Of course, networking is moderately uninteresting unless there are multiple computers in the picture. If you have a second machine with IPv6 enabled, you should be able to ping that computer using its link-local IPv6 address. For example, if you have a Linux machine that has autoconfigured address `fe80::2b0:d0ff:fed7:741d` on `eth0` and a FreeBSD machine that has configured address `fe80::202:b3ff:fe65:604b` on `fxp1`, then the Linux host should be able to ping the FreeBSD host with the command `ping6 -I eth0 fe80::202:b3ff:fe65:604b` and the FreeBSD machine should be able to ping the Linux machine with the command `ping6 fe80::2b0:d0ff:fed7:741d%fxp1`*. Note that we've used an explicit flag to ping to give the interface in the Linux case, but used the KAME scope ID in the FreeBSD case.

Naturally, if there are global addresses assigned to these hosts then you should also be able to ping these without specifying any scope ID.

There is a nice trick for finding the addresses of IPv6 nodes on your network using `ping`: we can do this by pinging the all-nodes multicast address, `ff02::1`. For example, on the Linux machine mentioned above, we can ping this address on `eth0` via the command `ping6 -I eth0 ff02::1`. The output is shown in Example 5-4—here we received six responses to the ping (five of which are marked as duplicates) and the addresses of the nodes are shown.

Example 5-4. Output from pinging ff02::1

```
$ /usr/sbin/ping6 -I eth0 ff02::1
PING ff02::1(ff02::1) from fe80::2b0:d0ff:fed7:741d eth0: 56 data bytes
64 bytes from ::1: icmp_seq=1 ttl=64 time=0.062 ms
64 bytes from fe80::2b0:d0ff:fe05:fc06: icmp_seq=1 ttl=64 time=0.194 ms (DUP!)
64 bytes from fe80::206:5bff:fe68:249b: icmp_seq=1 ttl=64 time=0.224 ms (DUP!)
64 bytes from fe80::202:b3ff:fe65:604b: icmp_seq=1 ttl=64 time=0.256 ms (DUP!)
64 bytes from fe80::2b0:d0ff:fef4:c6c5: icmp_seq=1 ttl=64 time=0.334 ms (DUP!)
64 bytes from fe80::203:93ff:fe46:17a6: icmp_seq=1 ttl=64 time=0.384 ms (DUP!)

--- ff02::1 ping statistics ---
1 packets transmitted, 1 received, +5 duplicates, 0% loss, time 0ms
rtt min/avg/max/mdev = 0.062/0.242/0.384/0.103 ms
```

Unfortunately, this trick is not completely foolproof. Some versions of ping do not show duplicates and some nodes reply with an address other than their link-local address. However, it will usually even work on manually configured tunnels, which can be very useful for testing if the host at the remote end is properly configured.

This trick is not limited to pinging the all-nodes multicast address; so with very little effort, it can be used to perform small administration tasks on arbitrary multicast groups. This is very useful for working with specific, generally functional, groups of servers.

* This is of course all providing there is an appropriate communications medium between them! We cover debugging some aspects of how layer 2 things can go wrong later.

Even we can occasionally forget that ICMPv6 includes features that didn't exist in IPv4. For example, these days you can request a lot more than a simple ECHO_REPLY; using an ICMPv6 node information query you can request the addresses used by the responder, and hostnames of the destination endpoint. KAME's version of `ping6` supports these queries nicely—you can ask remote nodes what they think their host name is with the `-w` flag. We can squeeze even more out of this technique by combining it with pinging multicast groups, as shown in Example 5-5. The figure shows a node information query being sent to the all-nodes multicast group with `ping6`, which then displays the address and name of each host that replied.

Example 5-5. Output from node info query to ff02::1

```
% ping6 -w -I en0 ff02::1
PING6(72=40+8+24 bytes) fe80::203:93ff:fe46:17a6%en0 --&gt; ff02::1
39 bytes from fe80::206:5bff:fe68:249b%en0: adric
26 bytes from fe80::2b0:d0ff:fe05:fc06%en0: ace
37 bytes from fe80::202:b3ff:fe65:604b%en0: jo
40 bytes from fe80::2b0:d0ff:fef4:c6c5%en0: sarah-jane
^C
--- ff02::1 ping6 statistics ---
1 packets transmitted, 1 packets received, +3 duplicates, 0% packet loss
```

If we compare Example 5-4 and Example 5-5 we can see that not all nodes responded to the node information request. Unfortunately, these queries are not yet a full part of the standard.

If ping is working OK, then it should also be possible to telnet, even if you only get a "connection refused" message in response. Note that, most versions of `telnet` do not have an explicit option to allow you to specify the interface to be used for a link-local address. To get around this, some platforms automatically use a default interface. On platforms that support scope IDs in addresses, these can be used instead. Example 5-6 shows some examples of these in use. The first example, on Linux, shows telnetting to a unscoped link-local address and the corresponding error. In the next example, on Windows XP, it is unclear why the unscoped connect failed, but adding a scope ID allows the connection to proceed. The final example is with a Solaris host, where the LAN interface is used by default for link-local addresses.

Example 5-6. Telnetting to link-local addresses

```
linuxhost% telnet fe80::204:e2ff:fe33:e3ac
Trying fe80::204:e2ff:fe33:e3ac...
telnet: connect to address fe80::204:e2ff:fe33:e3ac: Invalid argument

C:\Documents and Settings&gt;telnet fe80::204:e2ff:fe33:e3ac
Connecting To fe80::204:e2ff:fe33:e3ac...Could not open connection to the host, on port
23: Connect failed
C:\Documents and Settings&gt;telnet fe80::204:e2ff:fe33:e3ac%4
FreeBSD/i386 (gonzo) (ttypb)
```

Example 5-6. Telnetting to link-local addresses (continued)

```
login:


solarishost% telnet fe80::204:e2ff:fe33:e3ac
Trying fe80::204:e2ff:fe33:e3ac...
Connected to fe80::204:e2ff:fe33:e3ac.
Escape character is '^]'.

FreeBSD/i386 (gonzo) (ttypc)

login:
```

Know Thy Neighbor (Before Thyself)

Pinging a node on your local network only requires Neighbor Discovery to be working correctly. The commands in Table 5-7 show how to display the neighbor cache. After pinging a host, its link-layer address should show up in the neighbor cache. If it does not, there is probably some problem with multicast—see the "Gotchas" section later in this chapter for problems we've encountered in this area. Table 5-8 shows how to display IPv6 caches.

Table 5-7. Displaying IPv6 neighbors

OS	Showing neighbor cache
Solaris	`netstat -p`
Linux	`ip -f inet6 neigh`
AIX	`ndp -a`
WinXP	`ipv6 nc`
Win2003	`netsh interface ipv6 show neighbors`
FreeBSD	`ndp -a`
Mac OS X	`ndp -a`
Tru64	`netstat -N`
IOS	`show ipv6 neighbors`
JUNOS	`show ipv6 neighbors`

Table 5-8. Displaying IPv6 routes

OS	Showing routes
Solaris	`netstat -rn`
Linux	`ip -f inet6 route`
AIX	`netstat -rn`
WinXP	`ipv6 rt`
Win2003	`netsh interface ipv6 show routes`

Table 5-8. Displaying IPv6 routes (continued)

OS	Showing routes
FreeBSD	`netstat -rn`
Mac OS X	`netstat -rn`
Tru64	`netstat -rn`
IOS	`show ipv6 route`
JUNOS	`set route forwarding-table family inet6`

Example 5-7 shows the neighbor cache on a Red Hat Linux host. The first two entries are actually for two different addresses on the same node. The first is a global `2001::` address. It is marked as stale because no communication has recently taken place using this address; the cache entry would be renewed by neighbor discovery if the address needed to be used. The second is the link-local address for the same node—we can tell this because the link-layer addresses and interface IDs are the same for both addresses. Note that this address is marked as a router. Finally, there is a global address for another node on the same network. Note that the corresponding link-local address is not in the table, indicating that these two nodes have been communicating using only the global address.

Example 5-7. Displaying the neighbor cache on Linux

```
$ ip -f inet6 neigh
2001:db8::202:b3ff:fe65:604b dev eth0 lladdr 00:02:b3:65:60:4b nud stale
fe80::202:b3ff:fe65:604b dev eth0 lladdr 00:02:b3:65:60:4b router nud reachable
2001:db8::2b0:d0ff:fef4:c6c5 dev eth0 lladdr 00:b0:d0:f4:c6:c5 nud reachable
```

Configuring Name Resolution

Name resolution is the process of turning host names into addresses and back again. Usually, configuring name resolution amounts to telling the operating system the IP address of your nameserver. Of course, now you have a choice of telling it an IPv4 or an IPv6 address.*

The easy option is to use an IPv4 address. You probably either already know the IPv4 address of your nameserver, or you don't need to know it, because it's been automatically configured by DHCP or PPP. Configuring an IPv4 address for your nameserver is quite simple because it doesn't involve any configuration other than the usual procedure for setting up IPv4 on a host, and then editing */etc/resolv.conf* on Unix-like systems or using the network control panels on Windows and Mac OS.

* You don't need to tell it both of the IP addresses of the nameserver because the DNS can resolve both IPv4 and IPv6 addresses, regardless of if you send it queries over IPv4 or IPv6.

Of course, in the long run we'll want to tell the operating system to be able to configure an IPv6 address as a nameserver. This is essential when we are configuring, say, an IPv6-only node. However, there are several obstacles to doing this from the start. First, we have to make sure our nameserver has an IPv6 address, which we discuss in "IPv6 Transport. Second, we need the operating system's resolver libraries to support the use of an IPv6 nameserver. Unfortunately, at the time of writing, many operating systems have a problem with this. Even on platforms such as Linux and FreeBSD, where this is supported, there can be unexpected problems: Chapter 5 gives an example of the sort of problem that might crop up.

Finally, most of us don't manually configure the DNS servers on every host, instead we use DHCP to manage this information centrally. In the IPv6 world the same effect can be achieved with DHCPv6, which we describe in "DHCP" in Chapter 4.

Old Dusty Libraries

A piece of commercial scientific software started misbehaving shortly after IPv6 DNS records were added for some Unix hosts. When run on some machines, its graphical interface wouldn't work if there was an IPv6 address associated with the name of the name of the X-Windows display. Strangely, the error given was "Permission Denied" or "Network Unreachable."

As a work around, the startup script for the software was changed to translate the display name into an IPv4 address and the problem was investigated further. After much system call and library call tracing, it emerged that the software used `gethostbyname`in an old version of the Linux C library. The C library then parsed */etc/hosts* using the `inet_addr` library function without checking the return value. When it encountered an IPv6 address, it returned `INADDR_NONE` to indicate an error, but this was erroneously converted to the address `255.255.255.255` and this value was returned to the application. The application then tried to make a TCP connection to this broadcast address, resulting in a "Permission Denied" or "Network Unreachable." This problem only occurred on hosts that consulted */etc/hosts* before DNS, because DNS knows how to parse both type of address correctly. The problem was resolved by having all hosts consult DNS first.

This shows why putting IPv6 addresses in files that may be parsed by old applications or libraries may not be such a good idea. Consequently, Solaris's strategy of using the *ipnodes* database may, in fact, be quite a good idea.

If you plan to do some small scale testing, you may want to add names for some of the IPv6 addresses you will be using during testing. For small scale testing, setting up DNS records (as we describe in "DNS) may be too heavy-duty, especially if the DNS server is not under your direct control. For this type of situation, it may be sufficient to add addresses to the */etc/hosts*file, or its equivalent.

Table 5-9 shows how to configure DNS resolving over IPv6, if it is available.* In some cases, you may want to use hostnames without configuring DNS, and so you may want to use a mechanism equivalent to the hosts file. Table 5-10 shows how to do this on the platforms considered in this chapter.

Table 5-9. Configuring IPv6 Resolver

OS	Enabling IPv6 transport resolver
Solaris	No support.
Linux	Edit */etc/resolv.conf*
AIX	No support.
WinXP	Use `netsh interface ipv6 add dns` *`ifnameserver IP`*
Win2003	Use `netsh interface ipv6 add dns` *`ifnameserver IP`*
FreeBSD	Edit */etc/resolv.conf*
Mac OS X	Not supported in Jaguar.. Supported on Panther through network control panel or by editing */etc/resolv.conf*.
Tru64	No support.
IOS	The `ip nameserver` command accepts IPv6 addresses.
JUNOS	`set system name-server` *`v6addr`*

Table 5-10. Static IPv6 address to hostname mapping (/etc/hosts or equivalent)

OS	IPv6 hosts file
Solaris	*/etc/inet/ipnodes*
Linux	*/etc/hosts*
AIX	*/etc/hosts*
WinXP	*C:\WINDOWS\SYSTEM32\DRIVERS\ETC\HOSTS*
Win2003	*C:\WINDOWS\SYSTEM32\DRIVERS\ETC\HOSTS*
FreeBSD	*/etc/hosts*
Mac OS X	*/etc/hosts*
Tru64	*/etc/ipnodes*
IOS	The `ipv6 host` command adds static entries to the host name cache.
JUNOS	`set system static-host-mapping` *`hostname`* `inet6` *`v6addr`*

Testing Further Afield: ping, telnet, and traceroute

As we have outlined, there are several choices for how you can connect to the IPv6 Internet. Rather than go into the details of those right now, let us assume that some-

* Some versions of Windows come preconfigured to use DNS over IPv6, with the server addresses set to be `fec0:0:0:ffff::1,2,3`. These addresses are site-local addresses that may be assigned to DNS servers. See the "DNS" section of Chapter 9 for more details.

one has provided you with a working IPv6 router and that autoconfiguration has provided you with a global address. What tests might you now perform?

Well, the `telnet` and `ping` tests listed in the "Testing with ping and telnet" section earlier in this chapter should work, but using the global addresses of local machines instead of their link-local addresses.

If routing is in place, you should also be able to telnet and ping machines out on the Internet—`www.kame.net` is probably a good machine to test with. Try `telnet` www.kame.net 80 and then typing `GET / HTTP/1.0` and then pressing return twice.* If everything works, you should be presented with the HTML for the KAME home page.†

What can go wrong here? Well, the first thing is that you'll need working DNS to get the IPv6 address for `www.kame.net`. The only situation where configuring DNS is tricky is if you have an IPv6-only host but have not yet set up an IPv6-capable nameserver. In this case you can always look up the address on an IPv4 host, using a command like `nslookup -query=aaaa www.kame.net` or `dig aaaa www.kame.net`, and then transfer it the old-fashioned way.‡

If the name is being translated to the address correctly, the next step is that the packets will need to get to your local router. Autoconfiguration should result in hosts learning the local default routers correctly, and you can check this by examining the routing table using the commands shown in Table 5-8. If the routing table is configured correctly a default route or a route for `2000::/3` should exist.

Note that the default router may advertise its link-local address, rather than a global address, so be prepared to see either as the gateway. Both router-discovery and neighbor-discovery are important here, because once a host has learned its default router's address, it may need to do neighbor discovery to learn the router's link-layer address.

If there is a routing problem, it should be possible to narrow it down using `traceroute`, as is done in IPv4. Table 5-6 shows the syntax of the IPv6 commands on our various platforms and Example 5-8 shows three traceroute examples. Each example shows tracerouting between two organizations under the same ISP. The first traceroute gets to its destination successfully; the times shown are the round trip times to each hop. The second example shows a situation where packets are being lost because a router had been powered off. Note that a "*" is shown instead of a time, to indicate a timeout. The final example shows a router returning ICMPv6 errors for an address that is not currently routable, indicated by the 'A!' after the time.

* The Windows version of `telnet` does not display the characters you type here, so you will have to type blind. It is possible to enable local echoing of what you type using `set localecho` on the `telnet` command-line.

† Note that towards the bottom of the HTML the KAME home page tells you if you are using IPv4 or IPv6.

‡ Pen and paper, or cut and paste.

Example 5-8. Traceroute examples

```
freebsdhost% traceroute6 -n 2001:db8:10:300::86e2:5103
traceroute6 to 2001:db8:10:300::86e2:5103 (2001:db8:10:300::86e2:5103) from 2001:db8:68:
ff::1, 30 hops max, 12 byte packets
 1  2001:db8:68:ff::2  0.801 ms  0.691 ms  0.669 ms
 2  2001:db8:8:9::1  6.843 ms  3.472 ms  3.457 ms
 3  2001:db8:8:3::2  4.432 ms  4.1 ms  4.166 ms
 4  2001:db8:8:4::2  4.665 ms  4.417 ms  4.458 ms
 5  2001:db8:10:100::86e2:a33  5.306 ms  4.781 ms  4.798 ms
 6  2001:db8:10:300::86e2:5103  5.369 ms  5.228 ms  5.076 ms

freebsdhost% traceroute6 -n 2001:db8:10:200::86e2:5103
traceroute6 to 2001:db8:10:200::86e2:5103 (2001:db8:10:200::86e2:5103) from 2001:db8:68:
ff::1, 30 hops max, 12 byte packets
 1  2001:db8:68:ff::2  0.779 ms  0.721 ms  0.669 ms
 2  2001:db8:8:9::1  3.719 ms  3.409 ms  3.269 ms
 3  2001:db8:8:3::2  4.527 ms  4.606 ms  3.966 ms
 4  2001:db8:8:4::2  4.649 ms  4.294 ms  4.374 ms
 5  2001:db8:10:100::86e2:a33  4.997 ms  6.483 ms  5.125 ms
 6  * * *
 7  * * *
 8  * * *
 9  *^C

freebsdhost% traceroute6 -n 2001:db8:100:300::86e2:5103
traceroute6 to 2001:db8:100:300::86e2:5103 (2001:db8:100:300::86e2:5103) from 2001:db8:68:
ff::1, 30 hops max, 12 byte packets
 1  2001:db8:68:ff::2  0.786 ms  0.701 ms  0.647 ms
 2  2001:db8:8:9::1  9.608 ms  3.649 ms  3.298 ms
 3  2001:db8:18:2:201:3ff:fe2c:960c  4.142 ms !A  3.936 ms !A  4.167 ms !A
```

As we have mentioned, path MTU discovery is an important part of IPv6 because IPv6 routers are not permitted to fragment packets. If some firewall between you and the destination does not allow ICMPv6 Packet Too Big messages through, then Path MTU discovery may not work correctly. The usual symptom is that TCP connections involving interactive or slow transfers work OK, but large or fast transfers hang unexpectedly and then time out. Some versions of `traceroute`, or variants of it such as Linux's `tracepath` can display Path MTU information, which may help diagnose this sort of problem.

Static Routing

In this section we'll have a quick look at configuring static routes. Static routes are routes that are configured by hand and don't really change often, as opposed to those routes learned from the network, which do. In the world of IPv4 we are often used to configuring a static route for the default gateway.* An IPv6 host will usually

* Though it's generally done for us if we use DHCP.

learn its default route from the network, so in the usual case the job of configuring the default route is effectively the job of configuring the local router.

However, there are reasons why you might want to configure static routes. First, you may want to configure a static route on your router, if you are not using IS-IS or OSPF to generate a routing table. Second, if you have a host connected to the IPv6 Internet via a tunnel (or some other transition mechanism) then you may not have a local router and you may need to configure your default route manually.

Table 5-11 and Table 5-12 show how a static route can be configured at boot time and at runtime. In this case, we show how to configure a route to the `2001:db8:beef::/48` network via a next hop of `2001:db8:babe::1`. Naturally, there are variants of these commands where you can add routes to a specific host or routes via a specific interface; to find out how to configure these permutations, consult your vendor's documentation.

Table 5-11. Boot time configuration of static routes: adding a route to 2001:db8:beef::/48 via 2001:db8:babe::1

OS	Configuring static routes at boot
Solaris	Create new script in */etc/init.d* and arrange for it to be run after *S*inet*, or add a command such as: `route add -inet6 2001:db8:beef::/48 2001:db8:babe::1` to the end of */etc/init.d/inetinit*.
Red Hat	Add entries of the form: `eth0 2001:db8:beef::/48 2001:db8:babe::1` to */etc/sysconfig/static-routes-ipv6*.
AIX	Use the Communications Applications and Services → TCP/IP → IPV6 Configuration → IPV6 Static Routes → Add an IPV6 Static Route menu in `smit` to add the route.
WinXP	`ipv6 rtu 2001:db8:beef::/48 ifindex/2001:db8:babe::1`
Win2003	`netsh interface ipv6 add route 2001:db8:beef::/48 ifindex 2001:db8:babe::1`

FreeBSD	In */etc/rc.conf* you can give the names of the static routes by setting:
	`ipv6_static_routes="name1 name2"`
	Then specify the routes themselves by setting:
	`ipv6_route_name1="2001:db8:beef::/48 2001:db8:babe::1"`
	and so on, also in */etc/rc.conf*.

Mac OS X	No specific technique, but could use Startup Items.
Tru64	Use `ip6_setup` or edit */etc/routes* and add a line like: `-inet6 2001:db8:beef::/48 2001:db8:babe::1`
IOS	`ipv6 route 2001:db8:beef::/48 2001:db8:babe::1`
JUNOS	`set routing-options rib inet6.0 static route 2001:db8:beef::/48 next-hop 2001:db8:babe::1`

Table 5-12. Runtime configuration static routes: adding a route to 2001:db8:beef::/48 via 2001:db8:babe::1

OS	Configuring static routes at runtime
Solaris	`route add -inet6 2001:db8:beef::/48 2001:db8:babe::1`
Linux	`ip -6 route add 2001:db8:beef::/48 via 2001:db8:babe::1`
AIX	`route add -inet6 2001:db8:beef::/48 2001:db8:babe::1`
WinXP	`ipv6 rtu 2001:db8:beef::/48 ifindex/2001:db8:babe::1`
Win2003	`netsh interface ipv6 add route 2001:db8:beef::/48 ifindex 2001:db8:babe::1`
FreeBSD	`route add -inet6 2001:db8:beef::/48 2001:db8:babe::1`
Mac OS X	`route add -inet6 2001:db8:beef:: -prefixlen 48 2001:db8:babe::1`
Tru64	`route add -inet6 2001:db8:beef::/48 2001:db8:babe::1`
IOS	`ipv6 route 2001:db8:beef::/48 2001:db8:babe::1`
JUNOS	`set routing-options rib inet6.0 static route 2001:db8:beef::/48 next-hop 2001:db8:babe::1`

If you wanted to configure a default route, rather than one to a `/48`, then you can use one of three ways to express this. The first is to add a route to `::/0`, which will catch any address that you don't have a better* route too. This may include unusual addresses, such as site-local addresses and the loopback address, so some people prefer to use `2000::/3` to configure their default route—this only covers the currently-used IPv6 global unicast space and doesn't catch unusual addresses. Finally, in the same way as you can say `route` add default in the IPv4 world, many IPv6 implementations allow you to use the keyword `default` also. This is the same as using `::/0`.

Note that an IPv6 router can only send ICMP redirects if it knows the link-local address of the next hop. If, for example, you have multiple routes out of a LAN and you want hosts to learn the best route via ICMP redirects from default router, then you must specify the next hop using its link-localaddress. This problem should not arise if you are using a dynamic routing protocol because these protocols calculate the link-local address of the next hop automatically.

Configuring Transition Mechanisms

In this section we'll talk about configuring some of the transition mechanisms. We'll give more complete descriptions for the more common ones (configured tunnels and 6to4) that are widely used to provide connectivity before native IPv6 is available.

* In other words, more specific.

Configured Tunnels

Configured tunnels are normally used to encapsulate IPv6 in IPv4 and ship it from one point in the Internet to another. To configure a tunnel of this sort you usually need 4 pieces of information: the source and destination IPv4 addresses used for encapsulation, and the source and destination IPv6 addresses assigned to either end of the virtual, point-to-point link.

The exact mechanism used to create tunnels varies a bit from platform to platform. On some platforms, the tunnel is presented as a point-to-point interface, but on others, the tunnel is created by setting the next hop to be an IPv4 compatible IPv6 address. Table 5-13 and Table 5-14 show the steps for boot-time and run-time configuration of tunnels on our operating systems.

Table 5-13. Boot time configuration of IPv6 over IPv4 tunnel

OS	Enabling a configured tunnel at boot

Solaris	Create */etc/hostname6.ip.tun0* containing the following:
	`tsrc localv4 tdst remotev4 up`
	`addif localv6 remotev6 up`

Red Hat	Create a */etc/sysconfig/network-scripts/ifcfg-sitX* where X > 0 containing the following:
	`DEVICE="sitX"`
	`BOOTPROTO="none"`
	`ONBOOT="yes"`
	`IPV6INIT="yes"`
	`IPV6TUNNELIPV4="remotev4"`
	`IPV6ADDR="localv6/prefixlen"`

AIX	Use smit to set up a tunnel using Communications Applications and Services → TCP/IP → IPV6 Configuration → IPV6 Network Interfaces → Configure Tunnel Interface.

WinXP	Interface 2 is the automatic tunnelling interface. We route packets to `2000::/3` over the tunnel.
	`ipv6 rtu 2000::/3 2/::remotev4`
	`ipv6 adu 2/localv6`

Win2003	Interface 2 is the automatic tunnelling interface. We route packets to `2000::/3` over the tunnel.
	`netsh interface ipv6 add route prefix=2000::/3 interface=2 nexthop=::remotev4`
	`netsh interface ipv6 add address interface=2 address=localv6`

FreeBSD	Add the following to */etc/rc.conf*:
	`gif_interfaces="gif0"`
	`gifconfig_gif0="localv4 remotev4"`
	`ipv6_ifconfig_gif0="localv6 remotev6 prefixlen 128"`

Mac OS X	No specific technique, but could use Startup Items.
Tru64	Use `ip6_setup` to set up a tunnel edit */etc/rc.config*directly.

IOS	`interface Tunnel0`
	` ipv6 address localv6/64`
	` tunnel source localv4`
	` tunnel destination remotev4`
	` tunnel mode ipv6ip`

Table 5-13. Boot time configuration of IPv6 over IPv4 tunnel (continued)

OS	Enabling a configured tunnel at boot

JUNOS	`set interfaces ip-1/0/0 unit 0 tunnel source localv4`
	`set interfaces ip-1/0/0 unit 0 tunnel destination remotev4`
	`set interfaces ip-1/0/0 unit 0 tunnel family inet6 address localv6/64`
	Note: the unit number should match the slot of the Tunnel/AS PIC.

Table 5-14. Runtime configuration of IPv6 over IPv4 tunnel

OS	Enabling a configured tunnel at runtime

Solaris	`ifconfig ip.tun0 inet6 plumb`
	`ifconfig ip.tun0 inet6 tsrc localv4 tdst remotev4 up`
	`ifconfig ip.tun0 inet6 addif localv6 remotev6 up`

Linux	`ip tunnel add sit1 mode sit ttl 64 remote remotev4 local localv4` `ip link set dev sit1 up`
AIX	The tunnel attributes `srctunnel4`, `destunnel4`, `srctunnel6`, and `destunnel6` can be set using `chdev`.

WinXP	Interface 2 is the automatic tunnelling interface. We route packets to `2000::/3` over the tunnel.
	`ipv6 rtu ::/0 2/::remotev4`
	`ipv6 adu 2/localv6`

Win2003	Interface 2 is the automatic tunnelling interface. We route packets to `2000::/3` over the tunnel.
	`netsh interface ipv6 add route prefix=2000::/3 interface=2 nexthop=::remotev4`
	`netsh interface ipv6 add address interface=2 address=localv6`

FreeBSD	`ifconfig gif0 create`
	`ifconfig gif0 tunnel localv4 remotev4`
	`ifconfig gif0 inet6 localv6 remotev6 prefixlen 128 up`

Mac OS X	The "gif" interface on OS X is self cloning—when you use gif0, gif1 will automatically be created, and so on.
	`ifconfig gif0 tunnel localv4 remotev4`
	`ifconfig gif0 inet6 localv6 remotev6 prefixlen 128 up`

Tru64	`iptunnel create -I ipt0 remotev4 localv4`
	`ifconfig ipt0 ipv6`
	`ifconfig ipt0 inet6 localv6`
	`ifconfig ipt0 up`
	`route add -host -inet6 remotev6 localv6 -interface -dev ipt0`

IOS	`interface Tunnel0`
	` ipv6 address localv6/64`
	` tunnel source localv4`
	` tunnel destination remotev4`
	` tunnel mode ipv6ip`

JUNOS	`set interfaces ip-1/0/0 unit 0 tunnel source localv4`
	`set interfaces ip-1/0/0 unit 0 tunnel destination remotev4`
	`set interfaces ip-1/0/0 unit 0 tunnel family inet6 address localv6/64`
	Note: the unit number should match the slot of the Tunnel/AS PIC.

After you have configured your tunnel, testing configured tunnels is like testing any other link. First, you'll want to check that you can ping all the addresses of both ends of the link, from both ends of the link. If the link is represented as an interface on your platform, then you may even be able to ping the all-nodes multicast address and get a response from both ends!

Debugging configured tunnels is slightly more tricky. Using a tool such as `tcpdump` can be quite useful. Generally, `tcpdump` allows you to attach to a specified interface and watch the packets arriving. The first thing to check is that the encapsulated IPv6 packet is being transmitted and arrives as expected. We can do this by using `tcpdump`'s `-i` flag to specify the actual interface we expect the IPv4 packet to pass through. If the IPv4 packet does not arrive, then some firewall may be filtering protocol 41. Remember also to check the hosts at both ends of the tunnel, as either of them might be running IPv4 or IPv6 firewall software.

On some platforms, we can actually run `tcpdump` on the tunnelling interface itself and see the IPv6 packet once it has been decapsulated. Seeing the decapsulated packet will confirm if there is no problem with the encapsulation/decapsulation.* Example 5-9 shows an example of running `tcpdump` on both the PPP interface (`tun0`) and then on the tunnel interface (`gif0`) on a FreeBSD host. Some versions of `tcpdump` give a warning when run on an interface with no IPv4 address configured, but this is harmless. Note that when we see the packets on the PPP interface, we can see the IPv4 addresses used for the tunnel, but when the packet gets to the tunnel interface the IPv4 addresses are stripped off.

Example 5-9. Using tcpdump to view encapsulated and decapsulated IPv6

```
# tcpdump -i tun0 -n -s0 ip proto 41
tcpdump: listening on tun0
11:34:09.181300 192.0.2.151 &gt; 192.0.2.1: 2001:db8:68:1ff:2b0:d0ff:fef4:c6c5 &gt; 2001:
db8:ccc1:1::1: icmp6: echo request
11:34:09.181486 192.0.2.1 &gt; 192.0.2.151: 2001:db8:ccc1:1::1 &gt; 2001:db8:68:1ff:2b0:
d0ff:fef4:c6c5: icmp6: echo reply

# tcpdump -i gif0 -n -s0
tcpdump: WARNING: gif0: no IPv4 address assigned
tcpdump: listening on gif0
11:35:17.736014 2001:db8:68:1ff:2b0:d0ff:fef4:c6c5 &gt; 2001:db8:ccc1:1::1: icmp6: echo
request
11:35:17.736093 2001:db8:ccc1:1::1 &gt; 2001:db8:68:1ff:2b0:d0ff:fef4:c6c5: icmp6: echo
reply
```

If the encapsulated packets are visible at both ends, but there still seem to be problems, then one possible occurrence is that the routing table is not directing all the desired IPv6 packets to the tunnel interface. Check that the routing table contains the correct routes using the commands shown in Table 5-8. The "Static Routing" section earlier in this chapter shows how to configure static routes.

Note, that some NAT systems will actually allow configured tunnels to function through NAT! In this case the NAT device will replace the destination/source IPv4

* Encapsulation/decapsulation problems should be rarer, as there are basically no configurable parameters. However, software or hardware bugs might lead to problems like this.

address of tunnelled packets on the way in/out of the NATed network. This complicates the configuration of the tunnel: the end of the tunnel inside the NATed network should use its private IP as the local IPv4 address and the end of the tunnel outside the NAT should use the NAT's public IP address as the remote end. To create the necessary NAT state and keep the connection alive, you may need to arrange for packets to be sent over the tunnel regularly (say, by running ping6 with an inter-packet time of a minute or so).

6to4 configuration

Setup of 6to4 is relatively straight-forward; in many ways it is like a configured tunnel, but you don't need to ask anyone for the local and remote IPv4 and IPv6 addresses. This makes things even simpler!

What you do need to know is your local IPv4 address and then a script like the one shown in Example 5-10 can do the rest. This script takes the IPv4 address of the host as its first argument, computes an IPv6 address for the host and configures the `stf0`, which is the 6to4 interface on KAME-derived systems. Table 5-15 and Table 5-16 show configuration details for various operating systems. The examples also show how to point the default route to the 6to4 interface, as this is a common configuration.

Note that not all the operating systems that we're considering can act as a 6to4 router. Solaris, for example, only supports it if you have the Solaris 9 4/03 Update installed. This shouldn't pose a problem though because you only need one 6to4 router to provide connectivity for a whole network.*

Example 5-10. Example 6to4 setup script

```
#!/bin/sh

IPV4=$1
PARTS=`echo $IPV4 | tr . ' '`
PREFIX48=`printf "2002:%02x%02x:%02x%02x" $PARTS`

STF_IF="stf0"
STF_NET6="$PREFIX48":0000
STF_IP6="$STF_NET6"::1

ifconfig $STF_IF inet6 $STF_IP6 prefixlen 16 alias
route add -inet6 default 2002:c058:6301::
```

* In fact, for 65536 networks, each being a `/64`!

Table 5-15. Boot time configuration of 6to4 as default route

OS	Enable 6to4 at boot
Solaris	Make */etc/hostname6.ip.6to4tun0* containing: `tsrc v4addr 6to4addr/64 up` then edit */etc/default/inetinit* and set `ACCEPT6TO4RELAY` to `YES` and check that `RELAY6TO4ADDR` is set to `192.88.99.1`.

Red Hat	Add:
	`IPV6TO4INIT=yes`
	to */etc/sysconfig/network-scripts/ifcfg-if* for the interface with the local IPv4 address and add:
	`IPV6_DEFAULTDEV=tun6to4`
	to */etc/sysconfig/network*..

WinXP	`netsh interface ipv6 6to4 set relay 192.88.99.1 enabled`
Win2003	`netsh interface ipv6 6to4 set relay 192.88.99.1 enabled`

FreeBSD	Set:
	`stf_interface_ipv4addr="v4addr"`
	`ipv6_defaultrouter="2002:c058:6301::"`
	in */etc/rc.conf*.

Mac OS X	No specific technique, but could use Startup Items and `ip6config`. Settings for `ip6config` are configurable in */etc/6to4.conf*.
Tru64	Use `ip6_setup` to setup 6to4 or edit */etc/rc.config*and */etc/routes* directly.

IOS	`interface Tunnel2002`
	` ipv6 address 6to4addr/16`
	` tunnel source if`
	` tunnel mode ipv6ip 6to4`
	`ipv6 route ::/0 2002:c058:6301::1`

Table 5-16. Runtime configuration of 6to4 as default route

OS	Enable 6to4 at runtime

Solaris	`ifconfig ip.6to4tun0 inet6 plumb`
	`ifconfig ip6to4tun0 inet6 tsrc IPv4-address 6to4-address/64 up`
	`6to4relay -e -a 192.88.99.1`

Linux	`ip tunnel add tun6to4 mode sit ttl 64 remote any local v4addr`
	`ip link set dev tun6to4 up`
	`ip -6 addr add 6to4addr/16 dev tun6to4`
	`ip -6 route add 2000::/3 via ::192.88.99.1 dev tun6to4 metric 1`

WinXP	`netsh interface ipv6 6to4 set relay 192.88.99.1 enabled`
Win2003	`netsh interface ipv6 6to4 set relay 192.88.99.1 enabled`
FreeBSD	`ifconfig stf0 inet6 6to4addr prefixlen 16` `route add -inet6 default 2002:c058:6301::`
Mac OS X	`ip6config start-stf if`

Tru64	`ifconfig tun1 ip6interfaceid ::v4addr ipv6 up`
	`ifconfig tun1 inet6 ip6prefix 6to4addr/64`
	`route add -inet6 2002::/16 fe80::v4addr -iface -dev tun1`
	`route add -inet6 default 2002:c058:6301:: -dev tun1`

Table 5-16. Runtime configuration of 6to4 as default route (continued)

OS	Enable 6to4 at runtime

IOS	`interface Tunnel2002`
	` ipv6 address 6to4addr/128`
	` tunnel source if`
	` tunnel mode ipv6ip 6to4`
	`ipv6 route 2002::/16 Tunnel2002`
	`ipv6 route ::/0 2002:c058:6301::1`

As 6to4 is another tunnelling technology, the techniques used to debug it are pretty similar to those we described for configured tunnels. One thing that you may want to do is find out where the nearest relay router is. You can do this by tracerouting to its anycast address `192.88.99.1`, which will reveal its location within the IPv4 network. Example 5-11 shows an example of this. If the relay is a long distance away, then you may want to talk to your ISP about a configured tunnel or ask them to provide a 6to4 relay for their customers.

Example 5-11. Locating your 6to4 relay

```
% traceroute 192.88.99.1
traceroute to 192.88.99.1 (192.88.99.1), 64 hops max, 44 byte packets
 1  gw-81 (134.226.81.1)  0.290 ms  0.171 ms  0.156 ms
 2  gswte1r1-vlan3.tcd.ie (134.226.1.104)  0.488 ms  0.440 ms  0.406 ms
 3  tcd.ge.link.hea.net (193.1.192.185)  0.738 ms  1.228 ms  0.615 ms
 4  Mantova-v101.Dublin.core.hea.net (193.1.196.149)  0.850 ms  0.892 ms  0.909 ms
 5  193.1.196.18 (193.1.196.18)  2.106 ms 2.960 ms  4.325 ms
```

Applications

So, you've now got your workstation talking IPv6. What's next? Well, it would be nice to be able to run some applications that use IPv6. We'll look at IPv6 support in the sort of applications many of us use regularly. We'll leave the configuration of the corresponding server-side software until Chapter 7.

Naturally, we can only survey the support available at the time of writing. As we'll see in Chapter 8, adding IPv6 support can be relatively straight-forward, so if your favorite application is listed as not supporting IPv6 then you should contact your vendor as they may have added it since we checked their software.

Web Browsers

A growing number of web browsers now support IPv6. In some cases the support varies from platform to platform; for example, some browsers have restrictions on how IPv6 web servers can be specified.

There are various sites you can visit to check if your browser supports IPv6. The standard test is to visit *http://www.kame.net/*, where the turtle at the top of this page

will dance if you requested the page by IPv6.* The KAME page also shows your IPv4 or IPv6 address at the bottom of the page.

Note that on all platforms it is possible to view IPv6 content on an IPv4-only browser by using a dual-stack proxy. We'll talk more about this in "HTTP Proxies and Caches" in Chapter 7.

Unix

Several browsers under Unix support IPv6. The best known is probably Mozilla, which has full IPv6 support on platforms with a working IPv6 stack. Similarly, browsers related to or competing with Mozilla, such as Netscape 7, Firefox, Konqueror and Galeon will also support IPv6. As of version 7.20 or so, Opera advertises experimental IPv6 support.

Nautilus, the Gnome file manager, can also be used as a browser, but in the versions of Nautilus we've tried, we've either found no IPv6 support or rather strange IPv6 support that only works for sites with both IPv6 and IPv4 DNS records.

For the console lovers amongst us, versions 2.8.4 and newer of the text based browser, `lynx`, also support IPv6.

Windows

Getting Internet Explorer to talk IPv6 is simple. First, it supports IPv6 only if the underlying core operating system also supports IPv6. In most cases, with modern editions of Windows (XP and later), the work has been done for you; Internet Explorer will initiate IPv6 connections once the IPv6 stack has been enabled. One annoying limitation of the XP version of Internet Explorer 6 is that it does not support literal IPv6 addresses in URLs (as described in the "When IPv6 Addresses Don't Fit" section of Chapter 8).

For Windows 2000, the procedure is somewhat more complicated. Some DLLs need to be replaced, and due to the way Windows Update (as well as System Protection) works, you have to be very careful about those DLLs getting wiped out by the system and removing your IPv6 capabilities.

Up to date versions of Opera, Mozilla, Firefox and Netscape also now seem to have good IPv6 support on Windows (some early versions had problems with things like IPv4 and IPv6 simultaneously, but this seems to have been resolved).

* Remember to hit reload or refresh on your browser if you have visited the page by IPv4 recently, otherwise it may have the IPv4 version of the page cached.

Mac OS

Safari, Apple's web browser, can visit IPv6 web sites. However, the API it uses under Jaguar does not currently support resolving IPv6 hostnames, so URLs have to include the address explicitly, i.e., using *http:///[::1]/* rather than *http://localhost/*. Under Panther Safari will use IPv6 to contact IPv6-only web sites and IPv4 to contact dual-stack or IPv4-only web sites, however it seems that it no longer understands IPv6 addresses in URLs. It is expected that the preference for IPv4 or IPv6 will become either user-configurable or dependent on the current network configuration. Safari can be downloaded from *http://www.apple.com/safari/* (it may come with your operating system distribution).

Mozilla on OS X will also supports IPv6, though it seems to have a similar restriction to Gnome's Nautilus in that it will not visit a server that advertises only an IPv6 address in the DNS. Servers with both IPv4 and IPv6 addresses can be contacted over IPv6. URLs with explicit IPv6 addresses also work. Firefox and Camino (formerly known as Chimera) have a similar level of support.

Internet Explorer on OS X does not currently support IPv6, and, given Microsoft's discontinuation of the product, is unlikely ever to support it.

Email Clients

Outlook Express, as shipped with Windows XP SP 1, doesn't seem to support IPv6 for POP, IMAP or SMTP. Apple's *Mail* client seems to be similarly constrained under Jaguar but has the beginnings of IPv6 support under Panther. Older Unix mail programs such as elm and mh do not yet support IPv6, though work is under way for some of the more recent ones like `pine`, kmail, and Evolution.

Lotus

Lotus Domino supports IMAP, POP, SMTP, LDAP and HTTP over IPv6 on AIX, Solaris and Linux. You should be able to add `TCP_EnableIPV6=1` to *NOTES.INI*.

Mozilla

We mentioned Mozilla above as a browser, but it also includes a mail reader. Again, on any Unix platform that supports IPv6, the Mozilla mail client should support IPv6.

Mutt

Mutt has supported IPv6 for some time. The use of IPv6 addresses can be controlled by the `use_ipv6` configuration variable in your *.muttrc*, but it defaults to yes, so no additional changes should be needed.

Sylpheed

As an example of a less well-known mailer that supports IPv6, we'll mention Sylpheed, a nippy GTK+ based mailer available from *http://sylpheed.good-day.net/*.

SSH

Now that `telnet`has been thoroughly discredited, SSH tends to be the remote access service of choice, especially for people who care about security. Many of the systems we talk about ship with the portable release of OpenSSH, from *http://www.openssh.com/*, which supports IPv6 if your system provides the standard IPv6 APIs. OpenSSH provides `-4` and `-6` flags for restricting operation to IPv4 or IPv6. It also accepts IPv6 addresses on the command line. The `scp` command uses a colon as a separator between hostname and filename, so if you want to use the an IPv6 address with `scp`, it needs to be enclosed in square brackets, for example `scp "user@[2001:db8::a00:2]:/etc/ipnodes" /tmp`.

For Windows users, a version of Simon Tatham's popular PuTTY program, compiled with IPv6 support is available from *http://unfix.org/projects/ipv6/*. Some versions of IPv6 PuTTY would *only* connect to hosts that have IPv6 DNS records, but up-to-date versions should also connect to hosts with IPv4-only DNS. Which protocol is preferred can be configured in PuTTY's `connection` panel. At some time in the future, IPv6 support should be rolled into the standard version of PuTTY.

Miscellaneous

Full support for IPv6 is currently in development by Wipro Technologies for the Gnome Desktop and Developer Platform, and should be available for Gnome 2.4.

Naturally, a prerequisite for IPv6 on Unix desktops is IPv6 support in X11. This work has been well underway and the first release of this code was in version X11R6.7. Details of the design of the IPv6 support can be found at *http://www.x.org/IPV6_Review.html*. Release 4.4.0 of XFree86 and X.org's 6.7.0 release both use this code on platforms that support IPv6. From an end user's point of view there should be no noticeable change, though you can now prefix a display name with `inet/` or `inet6/` to force a connection to be made over IPv4 or IPv6. Thus, you can say:

```
xclock -display desktop.example.com:0 xclock -display inet6/desktop.example.com:0
```

or even:

```
xclock -display ::1:0
```

Gotchas

Once a computer has IPv6 enabled it is likely to begin to find records relevant to IPv6 in the DNS. In an ideal world, this would cause no problems, even if the device was

not connected to the IPv6 Internet. However, a bug in some DNS servers has caused then to respond with a "host does not exist" message, rather than a "no record of this type" message. The best known occurrence of this led to IPv6 users not being able to connect to `news.bbc.co.uk` unless they first looked up its IPv4 address, though this problem has since been resolved. Others have had problems with `ad.doubleclick.net`, where some of its servers do not respond to queries for IPv6 addresses.*

Native IPv6 over Ethernet uses multicast at the link-level for a number of things, and thus is sensitive to the correct operation of multicast in Ethernet drivers. There have been several reports of vendors discovering that Ethernet multicast is broken only when users complain that IPv6 does not work correctly.

The usual manifestation of this is that Neighbor Discovery does not work correctly. One way to test this is to run a tool such as `tcpdump` that puts the Ethernet interface into *promiscuous* mode. This means that the Ethernet interface examines *all* packets, thus working around incorrect filtering of multicast packets. If IPv6 seems to work correctly while the interface is in promiscuous mode, there's probably a multicast problem. You will need to contact your vendor for a fix.

One other confusing thing that can happen is that router solicitation and advertisement do not properly occur, but `ping6` appears to work fine if the host is already in the neighbor cache! This is of course because `ping6` is unicast and the ND/RA protocols rely on multicast. Again, this is indicative of a underlying multicast problem.

We have also seen switches that have trouble forwarding IPv6 multicast packets if features such as IGMP snooping are enabled. In this case, while pinging the all nodes multicast address `ff02::1` from one node, we saw no packets at all arriving at another node. Using the all nodes address here is useful because it does not require neighbor discovery, which depends on multicast anyway. In a similar way, some wireless access points do not forward Ethernet multicast or require special configuration to do so. They would exhibit similar symptoms to above.

We have also seen one strange problem where IPv6 would not operate correctly between a router and a switch using ISL encapsulation for a VLAN trunked port. Switching the encapsulation to 802.1Q resolved the issue. Don't ask us—we just work here.

Summary

We've gone into the details of how to do the basic configuration of IPv6 for a variety of devices you might encounter on an IP network, covering the basic details for each OS and the commands you'll need to use. We've covered the most common compli-

* Considered a feature by some...

cations you're likely to face, but at the end of the day, complications can arise in almost any procedure. Probably the best approach, if you anticipate a tricky install, is to use this chapter as reference material for the install, and as a list of pointers of what to do if you have problems. Remember, the vendor documentation can be quite detailed and it is worth revising or keeping open beside you as you work.

Finally, if something isn't working, take heart, there is very probably a way to do what you want to do—it's often a case of lateral thinking, and using Google or similar search engines to look for the exact error you've been getting.