The Internet, and many smaller, private networks, use the Internet Protocol version 4, IPv4, as the network layer upon which other protocols, such as TCP, reside. Developed in the late ’70s, it became apparent by the early ’90s that IPv4 was reaching the end of its useful life.

The Internet Software Consortium’s annual survey of Domain Name Servers indicates incredible growth on the Internet. The latest numbers represent an annual growth rate of over 65%, and that includes only those systems that are publicly advertised. Private nodes are not included in this count. In fact, it was the fear of running out of IP addresses that first prompted the look into a new version of IP.

Changes
The need for a new set of Internet protocols was first described in RFC 1287, “Towards the Future Internet Architecture,” published in December 1991. By early 1996, the first proposed standards for IP version 6, IPv6, were published.

Four major changes were made in version 6:

- Larger address space
- Simplified header
- Flows
- Authentication and privacy

Larger address space
The most highly publicized change is the larger address space. IPv4 addresses are 32 bits in length. The number of connected networks has continuously doubled...
in less than 12 months for much of the past 10 to 15 years. With the explosion of Internet devices, PCs, PDAs, and even cell phones, the Internet is expected to run out of IPv4 address spaces sooner than later. In contrast, IPv6 has 128-bit addresses.

One way to understand how many addresses 128 bits represents is to think of a book that describes every possible Internet node address with 100 node descriptions per page. If each page was 0.1mm thick and was printed double-sided, the book to describe all possible IPv4 addresses would be 2,000 meters thick.

This may sound like a big book, but the book describing all possible IPv6 addresses would be \(2 \times 10^{16}\) light-years thick. To date, the farthest sight seen by the Hubble telescope is less than \(2 \times 10^{13}\) light-years.

You may recall that IPv4 uses a “dotted decimal” notation. For example, 123.45.67.89. IPv6 addresses are instead represented by hexadecimal 16-bit quantities separated by colons, for example, FF02:0:0:0:0:1:200E:8C6C (with the exception noted below). A double colon (::) can be used as a type of shorthand for a string of zeroes. The address FF02::1:200E:8C6C is, therefore, equivalent to the previous IPv6 address. The double colon can only be used once in an address.

There is also the notion of an IPv4-compatible address, that has the 96 high-order bits set to zero. This can be represented as 0:0:0:0:0:0:0:123.45.67.89 or ::123.45.67.89.

Whenever there is a case where a colon would cause confusion, a literal IPv6 address can be quoted using square brackets. For example, a URL may look like http://[fec0::55:a00:20ff:fe90:58f8]:80/index.html.

**Simplified header**

IPv6 headers consist of eight equal length fields. This is in contrast to IPv4, which has variable length fields, some of which are optional. IPv6 handles options by placing them in sepa-
rate extension headers that follow the actual IP header. These changes make IPv6 packets more efficient to route since there are fewer fields to examine and process.

**Flows**

A “flow” is defined by RFC 1883 as a sequence of packets that require special handling by the intervening routers.3 Flows may be used to control quality of service to support “real-time” applications like voice and video.

**Authentication and privacy**

Although security was considered when IPv4 was designed, e-commerce has stressed the need for built-in security. IPv6 uses two extension headers for security: the authentication header described in RFC 18264, and the encapsulating security payload described in RFC 18275.

The authentication header provides strong integrity and strong authentication for packets. The encapsulating security payload provides confidentiality by encrypting the payload of the packets.

**Autoconfiguration**

IPv6 addresses are longer than IPv4 addresses, and much more awkward for people to work with. One of the goals of the IPv6 standards was to simplify the management of addresses.

Toward that end, IPv6 offers autoconfiguration, which gives IPv6 machines plug-and-play connectivity. Two types of autoconfiguration are available: stateless and stateful.

With *stateless autoconfiguration*, hosts automatically compute their own IP address by combining the network prefix obtained by the local router with their own Ethernet MAC address. Stateless autoconfiguration is described in RFC 2462.6

Because some people view the use of a MAC address as an invasion of privacy, a new draft proposes a way for hosts to compute their own IP address based on a random number.7 Using a

---

**LISTING 1** Using `sockaddr_in6`

```c
struct sockaddr_in6 sin;
struct hostent *hp;
...
(void) memset(&sin, sizeof(sin), 0);
sin.sin6_family = hp->h_addrtype;
sin.sin6_flowinfo = 0;
sin.sin6_port = htons(port);
(void) memcpy(&sin.sin6_addr, hp->h_addr, hp->h_length);
if (connect(sock, (struct sockaddr *)&sin, sizeof(sin)) == -1) {
    ...
}
```

**LISTING 2** Using `gethostbyname()` vs. `getaddrinfo()`

```c
a)
int myconnect(char *host, int port)
{
    struct sockaddr_in dest;
    struct hostent *hp;
    int ret;
    int sock;
    hp = gethostbyname(host);
    if (hp == NULL ||
        hp->h_addrtype != AF_INET ||
        hp->h_length != 4 {
        /* handle error */
    }
    dest.sin_family = AF_INET;
    dest.sin_port = htons(port);
    bcopy(hp->h_addr,
        &dest.sin_addr, 4);
    sock = socket(AF_INET,
        SOCK_STREAM, 0);
    if (sock == -1) {
        /* handle error */
    }
    ret = connect(sock,
        (struct sockaddr *)&dest, sizeof(dest));
    if (ret == -1) {
        /* handle error */
    }
    return(sock)
}

b)
int myconnect(char *host, int port)
{
    struct addrinfo *res;
    struct addrinfo *aip;
    int error;
    int sock;
    error = getaddrinfo(node,
        service, NULL, &res);
    if (error != 0) {
        /* handle error */
    }
    for (aip = res; aip != NULL;
        aip = aip->ai_next) {
        sock = socket(aip->ai_family,
            aip->ai_socktype,
            aip->ai_protocol);
        if (sock == -1) {
            /* handle error */
        } else {
            if (connect(sock,
                aip->ai_addr,
                aip->ai_addrlen) == -1) {
                close(sock);
                continue;
            }
            break;
        }
    }
    freeaddrinfo(res);
    return(sock);
}
```
random number makes it harder for eavesdroppers and other information collectors to identify machines.

Stateful autoconfiguration is done with the DHCPv6 protocol.8,9 This is similar to the current DHCP protocol that is used to parcel out addresses and service information to individual machines.

One advantage of using DHCPv6, instead of stateless autoconfiguration, is that DHCPv6 gives you the ability to configure dynamic updates to DNS. Another advantage is security. With stateless autoconfiguration, any node can connect to your network and start running. With DHCPv6 servers, you can maintain tighter control over which nodes can be configured for the network. The only nodes that can be configured through the DHCP server are those that you have explicitly authorized.

Converting from IPv4
RFC 2553, which includes portions of POSIX 1003.1g, defines the new sockets API for IPv6.10,11 The good news is that the new API is protocol-independent. It doesn’t matter if your protocol is IPv4, IPv6, or IPv—it will work with no changes.

This means you can code for IPv6 today, even if you still only use IPv4! The API, and your application, will still work. When you enable IPv6 in your network, the applications should continue to run with no changes.

The changes described in RFC 2553 for IPv6 cover four areas:

- Core socket functions
- Address data structures
- Name-to-address translation functions
- Address conversion functions

Core socket functions
The original core socket functions, those that set up and tear down connections and send and receive packets, were designed to be protocol-independent. These functions do not have to change for IPv6, but they do require...
some new IPv6 data structures, which are described in the next section.

Note that this applies only if you are using a sockets API, which may not be the case in an embedded system. Similarly, the application code changes I’m about to describe will only make sense if you’re using a sockets or sockets-like networking API. If you are using such an API, you may be able to take advantage of a “socket scrubber” developed at Sun (www.sun.com/software/solaris/ipv6).

Address data structures
A protocol-specific data structure is defined for each protocol that the socket functions support. You normally cast the protocol-specific data structure to sockaddr when calling the core socket functions. For example, in IPv4, the protocol-specific data structure is sockaddr_in which you cast to sockaddr when calling connect(). The protocol-specific data structure for IPv6 is sockaddr_in6.

IPv6 also has a new address family, AF_INET6, a new protocol family, PF_INET6, and a new structure for holding a single IPv6 address, in6_addr. These are used in a way that is similar to IPv4’s AF_INET, PF_INET, and in_addr.

Name-to-address translation functions
The most common function for translating names to addresses in IPv4 is gethostbyname(). It’s still retained for backwards compatibility, but is inadequate for IPv6. A common complaint has been that gethostbyname() is not thread-safe, but a bigger problem is that there’s no way to specify anything about the types of addresses desired (IPv4, IPv6, and so on). Adopted from POSIX 1003.1g, getaddrinfo() solves these problems.

There’s also a function to perform the reverse. Given a node name, getnameinfo() looks up an IP address.

Address conversion functions
Typically, the functions inet_addr() and inet_ntoa() are used to convert
In reality, "dual-stack" is a misnomer. Actually, only one IP module handles both IPv4 and IPv6.

an IPv4 address between binary and text form. The analogous IPv6 functions are called inet_pton() and inet_ntop(). inet_pton() converts a text address to its binary equivalent; inet_ntop() does the reverse, converting a binary address to printable text. Since both of these functions have an parameter for specifying the address family, they can be used to convert both IPv4 and IPv6 addresses.

6bone

The 6bone is an IPv6 testbed. It's run as a world-wide informal collaborative project with oversight from the NGtrans (IPv6 Transition) working group of the IETF. There are 462 sites in 42 countries.

You can use the 6bone for testing purposes, or to gain experience with IPv6 before releasing your product commercially. To join the 6bone, you will need the following:

- A router that supports IPv6
- A 6bone site you can connect to
- A nameserver that support IPv6 AAAA records

You can find more information about the 6bone at www.6bone.net.

The easiest way to connect a single workstation to the 6bone is to use Freenet6 by Viagènie. Freenet6 is a server that will generate a configuration for you to download for your specific operating system. After downloading and installing the configuration, you’re on the 6bone. See www.freenet6.net for more information.

The state of IPv6 today

When the IETF started working on IPv6, one of the requirements was to allow for a slow migration from IPv4 to IPv6. Obviously, an abrupt cutover from one protocol to another would be impossible. To make IPv6 successful, it would have to work on the Internet simultaneously with IPv4.


Dual-stack nodes

A dual-stack node supports both IPv4 and IPv6 simultaneously. When communicating with an IPv6 node, or another dual-stack node, the dual-stack node will use IPv6. When communicating with an IPv4 node, the dual-stack node uses IPv4. Both protocols are handled gracefully, unknownto the applications.

In reality, “dual-stack” is a misnomer. Actually, only one IP module handles both IPv4 and IPv6. The socket layer, based on the AF_INET or AF_INET6 parameter, knows which protocol is being used. If IPv4 is being used, it constructs an IPv4-mapped address, where the 80 high-order bits are zero, the next 16 bits are 0xffff, and the 32 low-order bits are the IPv4 address, for example, ::FFFF:123.45.67.89. Thus, the IP module always receives a 128-bit address from the upper layer.

When the IP module receives an IPv4-mapped address, it assembles the IPv4 headers. When it receives a non-IPv4-mapped address, it assembles the IPv6 headers. Either way, the same module does the work of both protocols, so there really is only one stack.

Tunnels

What happens if two IPv6 nodes are separated by an IPv4 network? How do they communicate? By using dual-stack routers, a tunnel can be dug across the IPv4 network. Tunnels are dug by encapsulating an IP packet within the payload of another packet.
The dual-stack router on one end of the communication takes IPv6 packets from the sender, encapsulates them within IPv4 packets, then forwards the packets across the IPv4 Internet. Another dual-stack router on the other side receives the IPv4 packets, extracts the IPv6 packets inside, and forwards the IPv6 packets to their proper destination.

Configuring tunnels can be a little tricky. To make life easier, the IETF draft, "IPv6 Tunnel Broker," describes a way to automatically manage tunnels. A tunnel broker is a dual-stack machine that creates, modifies, or deletes tunnels for you. To dig a tunnel, you simply give the tunnel broker the following information:

- The IPv4 address of the tunnel’s client side
- A nickname to be used for the DNS registration of the global IPv6 addresses assigned to both sides of the tunnel
- The client function: standalone host or router

See the IETF draft or more details on how a tunnel broker works.

Communicating with IPv4 machines
Several mechanisms have been defined by the IETF for IPv6 applications to talk with IPv4 applications. See the referenced documents for more detailed information.

SIIT\textsuperscript{15}

The Stateless IP/ICMP Translation Algorithm (SIIT) translates between IPv4 and IPv6 packet headers in separate translator “boxes” in the network.

NAT-PT\textsuperscript{16}

A combination of Network Address Translation (NAT) and Protocol Translation (PT) can be used to provide transparent routing. This does not mandate dual-stacks or special purpose routing requirements on end nodes.

TCP/UDP Relay\textsuperscript{17}

This mechanism translates packets at the TCP layer. A transport relay translator box sits between an IPv6-only node and an IPv4-only node, handling the protocol translation.

DSTM\textsuperscript{18}

The Dual Stack Transition Mechanism (DSTM) provides a method to assign temporary global IPv4 addresses to IPv6 nodes. It also uses dynamic tunnels within an IPv6 network to carry IPv4 traffic.

6to4\textsuperscript{19}

This mechanism uses an IPv4 network...
as a unicast point-to-point link layer. IPv4 sites are assigned a unique IPv6 address prefix. Packets are then encapsulated using this prefix and transmitted over IPv4. No configured tunnels are required. 6to4 is typically implemented almost entirely in border routers.

6over4
This RFC allows isolated IPv6 nodes, located on a physical link that has no directly connected IPv6 router, to run IPv6 by using an IPv4 multicast domain as their virtual local link. The IPv6 hosts do not require IPv4-compatible addresses or configured tunnels in this approach.

Do your systems need to support both IPv4 and IPv6, or just IPv6? That depends. If your systems will run in a private network, IPv6 alone is enough. The extra memory required by the IPv4 portion of the stack can be eliminated if your systems will never talk to IPv4-only applications. Even if your applications connect to the Internet, they can use tunnels to communicate with other IPv6-only applications. The dual-stack can be relegated to the routers connecting your private network to the Internet.

In practice, the memory consumed by the IPv4 portion of the stack is relatively small, approximately 15KB for most implementations. Including it makes a nice insurance policy, especially since the majority of the Internet will still be IPv4 for a while. With the dual-stack approach, you will be guaranteed that your applications will work today and tomorrow, regardless of the mix of IPv4 and IPv6 machines on the network.

Not if, but when
The big question these days is not if IPv6 will be widely deployed, but when. IPv6 is already being rolled out in Japan and parts of Europe, where IPv4 addresses are increasingly difficult to obtain. In North America, where IPv4 addresses remain more plentiful, IPv6 will deploy more slowly. Nevertheless, it’s coming in the next few years. Will your systems be ready for the change?

Stephen Harpster is the engineering manager for IPv6 at Sun Microsystems Inc. He earned his BS in computer science at the Georgia Institute of Technology in 1982, and his MS in computer science/electrical engineering at Northwestern University in 1987. You can reach him via e-mail at stephen.harpster@sun.com.

References
All of the RFCs and drafts can be retrieved for free from the IETF at www.ietf.org.

Additional IPv6 resources
This set of web pages provides good generic information on IPv6: playground.sun.com/pub/ipng/html/ This document outlines the business and technical case for IPv6: www.ietf.org/internet-drafts/draft-iat-case-for-ipv6-05.txt
This is a geographic map that gives an idea of worldwide deployment: noc.ntua.gr/~adamo/6bone/
The IPv6 Forum is a world-wide consortium of Internet vendors: www.ipv6forum.com/
These are slides from a good tutorial: www.viagenie.qc.ca/en/ipv6forum.pdf