

Getting Ready for IPv6

Starting from Current IP

Networked computers are becoming the norm instead of the exception today. This is a dramatic shift from the so called “standalone” desktop. But, as the demand for services, features, and information grows, the majority of computer owners opt to pay for connectivity to the Internet. If you are reading this article, you probably got it off the Internet.

Connectivity requires certain policies and interfaces. The computer on your desk must, of course, have the capability to connect to the network either through a network interface card (NIC) or through a modem. The computer also needs a unique method of identification. Most NICs have a built-in 6-byte ID called the Media Access Control (MAC). Modems do not have this feature since they were designed to be a point-to-point connection *not for networking*. Still, because the network cannot depend on the MAC, it is unsuitable for networking addressing.

The Internet Protocol (IP version 4 or *IPv4*) defines a 4-byte ID that can include routing information. The address is divided into separate domains with each domain responsible for routing, decoding subnets, and mapping to the destinations’ MACs.

IPv4’s 4-byte allocation has enough addresses for 4 billion hosts. You’ve probably worked with the dot notation like 129.56.230.4. However, because of current allocation techniques, less than the 2% of the allocated addresses are actually used. Also, the accepted IP definition sets apart several addressing subranges for special purposes, for example, 127.xxx.xxx.xxx is “this computer” – over 16 million addresses. These factors, along with the explosive growth of the Internet, have led experts to predict that IPv4 will run out of addresses within a few years.

The growth of the Internet has also forced another unexpected need – unified networks. The Internet is really a collection of different networks, some more compatible than others. Some are different enough that the networks require address mapping from the foreign address to the IP address.

To encompass different networks while breathing life on IPv4’s fading embers, the addressing must change – expanded and clarified.

Widening the Network with IPv6

A worldwide network has to be flexible enough to support disparate addressing, subranges for local networks, and enough breathing room for millions of more computers. The next generation of IP, *IPng* or *IPv6*, defines an address that is four times the size of IPv4. It also sets aside several ranges for multicasting and foreign network addressing.

IPv6’s 16-byte address can handle about 3.4×10^{38} which when compared to the IPv4’s address space appears to give you as the programmer and/or administrator more flexibility. The allocation of IPv6’s addresses will probably continue the same approach as now done. This means that we can expect many years of longevity.

Getting Ready for IPv6

Naturally, several things had to change to support the new protocol – not the least of which is the IP packet. The IP packet, the most fundamental block of data that the program may work with, must carry at least the destination address, the source address, and the length in bytes. The new protocol simplifies the packet and reduces the potential overhead load. Instead of IPv4's 20-byte header information, IPv6 now uses 40 bytes.

The representation had to change as well. To distinguish the familiar IPv4's dot notation (and perhaps to increase readability a little), the new protocol uses eight 16 bit hexadecimal numbers separated by colons. For example, you may get an address like 2FFF:80:0:0:0:32C:2356 (or 2FFF:80::32C:2356, the two colons implying all zeros). You can even represent an IPv4 address using 0:0:0:0:FFFF:XXXX:XXXX (or ::FFFF:XXXX:XXXX) where the XXXX:XXXX is the IPv4's address expressed as hexadecimal. The new protocol really does encompass the old.

Another added benefit of the new IP header is its ability to handle huge packets. The current IP packet cannot handle any packet bigger than 65535 bytes. On the newer, faster networks, that limitation is very constricting and is prone to fragmentize the network (especially Ethernet). IPv6 offers a *jumbo payload* that allows for up to 4 Gbytes!

The benefits of moving to IPv6 include better addressing, better multicasting (another topic), larger capacity messaging, and multi-networks. But before you can actually use it, you need support for it.

IPv6-Supporting Languages/Networks

With all the great capabilities that IPv6 offers, you naturally might want to try it out. You face three obstacles: operating systems, networks, and languages. Linux and most current operating systems actually support IPv6 right out of the box. Some you may have to add support for it.

The next obstacle is the network itself. IPv6 is becoming very popular in Asia and Europe, so if you live in those countries you're in luck. However, the United States has not jumped completely on the bandwagon. The network routers there still use the older protocols and don't even support *dual stacks* (coupling IPv4/IPv6 together making processing seamless to the program). If you find that you are in an area that has very limited access to IPv6 support, you can hook in with IPv6-over-IPv4 (or *6bone*). Your operating system may attach to 6bone without your intervention.

The last obstacle in programming IPv6 is language support. Most languages do not support it. Java has a very good networking package, but it does not yet include the needed class interfaces. The Sun's website implies support in the future but makes no promises. C is really the only language that does include the structures and system calls; and since C++ is built on top of C, it tacitly includes support as well.

C-Language Programming Changes

The C language has all the pieces you need to write socket-based programs in whatever network or protocol you desire. You are only limited to network accessibility and operating system support. The basic system call to create an IPv4 socket is:

```
int sd;
```

Getting Ready for IPv6

```
sd = socket(PF_INET, SOCK_STREAM, 0);
```

You use this call to start the process of connecting computers together across the network. `PF_INET` requests a IPv4 network, and the `SOCK_STREAM` requests a TCP connection. You may use this call for either the client or the server.

The IPv6 equivalent is very similar:

```
int sd;
sd = socket(PF_INET6, SOCK_STREAM, 0); /* Req. IPv6 protocol */
```

A server offers services through a published port. While setting up the port, the server program must convert the addresses into equivalent machine-readable format. Typically, you may use `inet_ntoa()` and `inet_aton()` to convert addressing from/to machine-readable form. For example, to set up a `sockaddr` (for binding an address to a published socket or connecting to a server) you may use:

```
struct sockaddr_in addr;
addr.sin_family = AF_INET; /* request IPv4 */
inet_aton("16.32.64.127", &addr.sin_addr); /* Convert address */
```

Or, you can get the address of the connecting client and print the results:

```
struct sockaddr_in addr;
int client, len=sizeof(addr);
client = accept(sd, &addr, &len); /* Await connection */
printf("Connected: %s\n",
       inet_ntoa(addr.sin_addr)); /* print address */
```

The IPv6 equivalents are very close to these. Instead of using `inet_ntoa()` and `inet_aton()`, use `inet_ntop()` and `inet_pton()`, respectively. Using the previous examples, you can represent them in IPv6 like this:

```
struct sockaddr_in6 addr;
addr.sin_family = AF_INET6; /* request IPv6 */
inet_pton(AF_INET6, "::FFFF:1020:40FF", /* (hex of above) */
          &addr.sin6_addr); /* Convert address */
```

Again, printing the client's IPv6 info, you use `inet_ntop()`:

```
struct sockaddr_in6 addr;
int client, len=sizeof(addr);
client = accept(sd, &addr, &len); /* Await IPv6 connection */
printf("Connected: %s\n",
       inet_ntop(AF_INET6, addr.sin6_addr)); /* print address */
```

These steps change the socket setup and client connection. From that point, if you have written your code carefully, you may not have to change anything. The operating system and network does the rest for you. You can delve into the extended socket options which can increase your throughput tremendously.

You can get the complete listings of all the program examples from <http://www.cs.utah.edu/~swalton>.

Getting Ready for IPv6

Preparing Your Code Today for Tomorrow

Even if you don't start working on the extra features of IPv6, you can take steps now to make your code easier to change (or even transition smoothly) later. Network programming is a very challenging and important skill. So, certain habits and techniques make it easier to avoid problems and enhance success.

First, be careful about return values. The examples above do not check the return values for easier reading. Go to the <http://www.cs.utah.edu/~swalton> website for examples of how to catch errors.

Second, don't assume that the user is going to give you one address or another; test it instead. While `inet_pton()` and `inet_ntop()` have a parameter for the network type, it cannot auto-detect the network. You may have to do that yourself.

Finally, note, document, and centralize the sections that may need changing later. This can save your skin over and over again. It will make your life (and those maintain product) easier.

Summary

IPv6 expands the network addressing range considerably making the Internet more viable for the future. It also adds greater packet capacity, up to 4 Gbytes. Finally, it simplifies the messaging protocol while giving more power over multicasting and delivery.

The network is moving towards IPv6: some segments a little slower than others. Still, it is going to happen. Already, you can get several standard networking tools (like telnet and ftp) which are IPv6 ready. Being aware now can help you be more productive later.

If you would like to know more about this topic see the *Linux Socket Programming* book by Sean Walton or visit the <http://www.cs.utah.edu/~swalton> website.