Top 10 Tasks for IPv6 Application Developers

Things to consider when creating dual-protocol applications

By Scott Hogg *on Sun, 01/23/11 - 7:03pm.*

Many IT people, who are unfamiliar with IPv6, believe the responsibility for IPv6 deployment falls on the network-teams. However, those who are knowledgeable about IPv6 realize the migration to IPv6 will involve any system that uses an IP address. As the network teams prepare the infrastructure for the addition of IPv6 we should alert our application developers and make sure they are ready for the challenge that awaits. This article contains some of the key issues that application developers will need to know as they make their applications function properly in a multi-protocol world.

Even though network engineers place a high level of importance on the network we need to keep things in perspective. We network engineers realize that the network's role is to support end-to-end communications between computers running applications. The way the applications, operating systems, and computers interface with the network tends to fall within the realm of the system administrators and the application developers. As we become familiar with IPv6 we should share our knowledge with others in our organizations. That means that virtually everyone in the IT organization will be involved with the deployment of IPv6. We should share the following concepts with our application developers so they are aware of the nuanced of creating applications that will operate over current IPv4 networks and in the near future as IPv6 is added.

1. Assessing Current Code for IPv6 Capability

When application developers prepared for Y2K they took a hard look at their code for anywhere the year was held in only 2 characters/digits. Back then, Y2K programming code assessment utilities looked into the lines of code looking for data structures that contained either a 2-digit or 4-digit year. With the addition of IPv6 we need to look at our TCP/IP application code for anywhere an IP address may be entered, stored, accessed, or used. If we are using a data structure that allocates only 32-bits then the code will need to be modified to hold 128-bit addresses.

We could look at the application code file-by-file and module-by-module looking for such data structures or we can use an automated software application. Thankfully, there are several tools for automated assessment of application source code for IPv4/v6 calls. For hosts that are written on Microsoft platforms we can use Checkv4.exe to identify if the source code is IPv6-ready. There is a SourceForge tool called PortToIPv6. There are also two other programs that can help you with your IPv6 code assessment and these both work with C, Java, Perl and Python. The EUChinaGrid project has created an IPv6 code checker and the EGEE (Enabling Grids for E-science) group has created their IPv6-care utility. A fall-back option would be to use grep or other parsing tool to look through the source code.

2. Creating Code that is Address-Family (AF) Independent and Backward Compatible with IPv4

Applications must remain backward compatible with IPv4 as we deploy IPv6. Creating two versions of applications is not feasible. For example, it would not be prudent to create one version of an application that works for IPv4 and one version for IPv6 (e.g. program4.exe & program6.exe). Furthermore, end-user should not need to know which IP version is being used and then chose the correct application. However, advanced users may want to control how connections are made. Therefore, applications should be made to be Address Family (AF)-independent. That means that no matter which Address Family (IPv4 or IPv6) is being used, the same application will work in all cases.

3. Having the Same Application Work on IPv4-only, Dual-protocol, and IPv6-only Operating Systems With These Connectivity Methods

Along those same lines, AF-independent applications should work on any type of operating system that

runs on a computer that could be connected to a network that operates with IPv4-only, dual-protocol or IPv6-only connectivity. The single program should handle DNS queries for either protocol and be able to communicate using either protocol version. The application should be able to perform proper DNS queries and receive either A or AAAA responses and create a connection based on either protocol. If IPv6 connectivity is not possible then the application should automatically fall back to IPv4.

4. Storing of 128-bit IPv6 Addresses in Memory and Compatibility with 32-bit IPv4 Addresses
A 32-bit IPv4 address string is displayed in quad-dotted-decimal format which can be 16 characters (4 sections of 3 decimal digits + 3 periods + one null character). A 128-bit IPv6 address string is displayed as 8 sections of 4 hex characters separated by ":"s (up to 46 characters). The application must use data entry fields that can accept IPv4 addresses, IPv6 addresses, hostnames and fully-qualified hostnames. The bounds must be checked to validate the input and the addresses must be stored in the proper data structures of the appropriate size. Just like when boarding an airplane there are people who far too often try to shove 10 pounds of stuff into a 5-pound overhead compartment. You too should not try to squeeze 128 bits of IPv6 address into a 32-bit memory space.

5. DNS Queries for IPv6 and IPv4 Using Either IPv6 or IPv4 Transport
As mentioned previously, applications should properly handle DNS queries because with 128-bit IPv6 addresses we will need to rely even more on DNS to keep IP addresses straight. The IETF RFC 4074 Common Misbehavior Against DNS Queries for IPv6 Addresses gives guidance on how to do this properly. "There is some known misbehavior of DNS authoritative servers when they are queried for AAAA resource records. Such behavior can block IPv4 communication that should actually be available, cause a significant delay in name resolution, or even make a denial of service attack." This means that IPv6-capable nodes should perform an AAAA query first and then an A query. Furthermore, the authoritative name server should return an RCODE=0 if no AAAA exists, then return the A record response.
Applications could perform DNS queries to a DNS server that has IPv4 or IPv6 network connectivity. Applications should also correctly perform forward and reverse DNS lookups when necessary. Many of the root name servers have IPv6 addresses. Last year the "I" root name server received an AAAA record and as such your DNS servers should be using the latest updated named.root file. Applications should also correctly perform forward and reverse DNS lookups when necessary.

6. Handling Input of FQDN Hostname or IPv4/IPv6 Address and Output Using Correct Format
Furthermore, IPv4 and IPv6 addresses can be used in a URL directly. We all know that we should leverage DNS and use FQDNs, but sometimes we just can't help it. In a URL, an IPv6 address can be enclosed in brackets like: http://[2001:DB8:1001::BEEF]:8080/index.html. This may be cumbersome for users and is a technique that would be mostly for diagnostic purposes as needed. Regardless, applications should follow the updated RFC 3986 Uniform Resource Identifier (URI): Generic Syntax.
When converting an IP address from its storage in memory to output as a string your application should also handle this properly for IPv4 and IPv6. If your code is written using the functions inet_aton(), inet_addr() or inet_ntoa(), then you will have problems. Your code should be using the inet_pton() and inet_ntop() whenever possible to make your code work in all circumstances and have the correct output format.

7. Making Socket Connections With IPv6 and IPv4
Applications that are using a computer and operating system that are dual-protocol capable and are on a network that has both IP version addresses should prefer IPv6 when possible. Applications should try making a connection using IPv6 first and then fall back to IPv4 if IPv6 connectivity does not exist. IETF RFC 4038 Application Aspects of IPv6 Transition gives guidance on creating dual-protocol applications.

This RFC tells us that there are really only 4 cases that should be considered.
Case 1: IPv4-only applications in a dual-stack node. IPv6 protocol is introduced in a node, but applications are not yet ported to support IPv6.
Case 2: IPv4-only applications and IPv6-only applications in a dual-stack node. Applications are ported for IPv6-only. Therefore there are two similar applications, one for each protocol version (e.g., ping and ping6).
Case 3: Applications supporting both IPv4 and IPv6 in a dual stack node. Applications are ported for both IPv4 and IPv6 support. Therefore, the existing IPv4 applications can be removed.
Case 4: Applications supporting both IPv4 and IPv6 in an IPv4-only node. Applications are ported for both IPv4 and IPv6 support, but the same applications may also have to work when IPv6 is not being used (e.g., disabled from the OS).
The first two cases are not interesting in the longer term; only few applications are inherently IPv4- or IPv6-specific, and should work with both protocols without having to care about which one is being used. Therefore, the focus should be on creating dual-protocol applications that can run on a network that uses either protocol or can operate on an IPv4-only network as exist mostly today.

8. Leverage Higher-Level APIs That are IPv6-Capable: BSD Socket API, Perl Socket6, IO::Socket::INET6, Python Socket Module, Java Sockets
The IETF has created several RFCs to give guidance on C-language IPv6 socket programming.
RFC 3493 - Basic Socket Interface Extensions for IPv6
RFC 3542 - Advanced Sockets Application Program Interface (API) for IPv6
RFC 4038 - Application Aspects of IPv6 Transition
RFC 5014 - IPv6 Socket API for Source Address Selection

Perl is a classic example of how to do this the wrong way. There is no IPv6 support in Perl CORE. You can use either Socket6 and IO::Socket::INET6 or Socket and IO::Socket::INET. No single API will work for IPv4-only, dual-protocol, and IPv6-only and due to this fact, it is not easy to create AF-independent code. Therefore, you must re-write your code to get IPv6-support. You can easily check to see if your version of Perl has IPv6 support using these following commands.
$ perl -MSocket6 -e1
$ perl -MIO::Socket::INET6 -e1

If you are writing Java code, then you have been blessed with IPv6-support for many years. IPv6 support was added in J2SDK/JRE 1.4. Java supports address types Inet4Address and Inet6Address and address-family independent sockets. However, there is something that you need to be aware of. With Java, the IPv6 stack is preferred by default but Java prefers IPv4 addresses returned by DNS. The following two settings control how Java uses either IP version.
java.net.preferIPv4Stack= < true|false > # false by default
java.net.preferIPv6Addresses= < true|false > # false by default
Therefore, you will want to add these lines to your java options.
-Djava.net.preferIPv4Stack=false
-Djava.net.preferIPv6Addresses=true

If you are using Python, then you are in luck. There is good IPv6 support in Python versions 2.3 and 2.6. The newest version of Python 2.7.1 has a few bug fixes that may make your life easier. Python has many address-family independent functions such as: socket.AF_INET, socket.AF_INET6, socket.getaddrinfo, socket.getnameinfo, socket.socket([family[, type[, proto]]]), and socket.inet_pton, socket.inet_ntop. There is a simple way to check if your version of Python has IPv6 capabilities. Just enter the following lines.
python

```
>>> import socket
>>> socket.has_ipv6
True
```

9. Working Well With Path MTU Discovery (PMTUD)
Another thing to remember about IPv6 networks is that IPv6 routers do not perform fragmentation of IPv6 packets. All IPv6 network interfaces have a minimum IPv6 Link MTU of 1280 bytes. When a router is asked to forward a packet that is too large for the link MTU it will drop the packet and send back to the source an ICMPv6 Packet Too Big message (Type 2). Therefore, the end nodes must perform Path MTU Discovery (PMTUD) and the source must create packets of the proper size. Path MTU Discovery for IP version 6 was defined in RFC 1981. The fragmentation extension header will be added to fragmented packets (next-header 44)(RFC 2460).
Many applications today do not perform PMTUD properly and this will need to change with the introduction of IPv6. Tunnels are pervasive in IPv6 so PMTUD is needed even more than with IPv4. Firewalls should also not filter PMTUD messages (FYI, ICMPv6 uses IPv6 extension header #58 (RFC 2463)).

10. Perform Extensive Testing of Your Applications
Test of application software can never be overdone. After you have created your address-family independent application then you should be testing it using a variety of operating systems. These computers should have network connectivity using IPv4-only, dual-protocol and IPv6-only. You should also test DNS using A and AAAA records over TCP and UDP port 53 over IPv4 and IPv6 and also perform reverse lookups. You should test, test, test, and then do some more testing. Believe me, this will pay dividends with fewer support calls and an application that you can rely on.

Additional Resources and References
There are many sources of information on creating address-family independent applications. One of my favorite books is Unix Network Programming, Volume 1, Third Edition, Addison-Wesley, W. Richard Stevens, Bill Fenner and Andrew M. Rudoff, 2004, ISBN-10: 0131411551. This book has a web page dedicated to it where you can download the code examples and experiment yourself.
Other books you may want to check out are the book TCP/IP Sockets in C, Second Edition: Practical Guide for Programmers (The Morgan Kaufmann Practical Guides Series), by Michael J. Donahoo and Kenneth L. Calvert, 2009, ISBN-10: 0123745403. These authors also have an earlier book TCP/IP Sockets in C#: Practical Guide for Programmers (The Practical Guides), by David Makofske, Michael J. Donahoo, and Kenneth L. Calvert, 2004, ISBN-10: 0124660517. If you write code in Java you should check out these same author's book TCP/IP Sockets in Java: Practical Guide for Programmers, 2nd Edition, Morgan Kaufmann, by Kenneth L. Calvert and Michael J. Donahoo, 2008, ISBN-10: 0123742552. Another useful book is IPv6 Network Programming, First Edition, Digital Press, by Jun-ichiro itojun Hagino, 2004 ISBN: 1555583180.

There are also numerous resources available on the Internet. As they say, "If you haven't Googled it, you haven't looked!" Owen Delong's of Hurricane Electric has a web page that contains useful information. Derek Morr of Penn State has a nice page on IPv6 Programming. Eva M. Castro also has a useful page on how to port applications to IPv6.

No matter where you start learning about how to create address-family independent applications, the important part is that you start working on this now. That way you will have longer to perform the conversion and have more time to be relaxed and thoroughly test your applications.

Scott