IPv6 Developer's Checklist

- ✓ **Address Data Structures (size, type, quantity)**
- ✓ **Network Operations**
- ✓ **Name-to-Address Translations (DNS)**

IPv6 Developer's Guide

Impact on Applications: One way to understand the impact on applications is to examine the differences between IPv4 and IPv6:

1. The size of addresses is different. Therefore the data structures (e.g. sockaddr_in vs. sockaddr_in6) used to communicate about addresses with the network stack are different. If the application uses addresses internally, say as names for hosts or nodes, those internal data structures need to change. If the application uses addresses in network protocols (which is a reasonable thing to do under many circumstances), those protocols need to change also.

2. Use "struct sockaddr_storage" for general purpose socket addresses. These can be cast to (struct sockaddr_in) or (struct sockaddr_in6) as necessary in socket calls. The sockaddr_storage struct allows enough room for the largest sockaddr_<type>.

3. When looking for parts of the code that deal with addresses, IPv4 addresses can be hard to find. They may be stored in (int), (unsigned int), (long), (unsigned long), char[4], (struct in_addr), etc. If your application has many references to IP addresses in this way, consider defining a separate, application-specific address structure such as (for the snort application):

```
typedef struct _SnortAddr {
                int SA_type;                /* AF_INET, AF_INET6, etc */
                int SA_length;              /* A little redundant, give type
                above, but useful */
                unsigned char SA_addr[sizeof(struct in6_addr)];
                                    /* struct in6_addr is largest addr to date */
} SnortAddr;
```

   Using this type of structure along with a few utility functions to get/set/compare address structures will allow you to change variable types and function prototypes which make finding those last few IPv4 address references easier.

4. Also, consider changing the names of data structure elements if the element changes from holding an IPv4 address to an IPv6 address or generic address. This, again, will let the compiler help you identify all the references to the IPv4 address so that you can check them and verify that they are still appropriate.

5. Use inet_ntop() and inet_pton() for address-to-string and string-to-address conversion respectively. Most applications will require very few changes to support both IPv4 and IPv6 addresses. Changes should be IP version agnostic and allow for nearly seem-less handling of IPv4 and IPv6 addresses.

6. There are different DNS records used for IPv4 and IPv6 addresses (A and AAAA, respectively). Since the old APIs assumed IPv4, there are new APIs for IPv6 address lookup. For compatibility with IPv4-only hosts or hosts on IPv4-only networks, the new APIs can also return IPv4 addresses.

7. Many IPv4-only applications expected or only used one address that was returned by DNS. Expect multiple addresses to be returned from a DNS lookup (IPv4 and IPv6) and use them in the order that the DNS provides. It is expected that DNS will return all the addresses for a host lookup in the order of preference. During the potentially uncertain transition period from IPv4 to dual-stack (to all IPv6?), falling back to the N^{th} address returned by DNS may be required.

8. There are several different kinds of unicast IPv6 addresses, including link-local addresses, public addresses, "privacy" addresses, "local" addresses, 6to4 addresses, and Teredo addresses. With the exception of link-local addresses all of these are globally unique and globally scoped. (Note: "site-local" addresses have been deprecated.) Some applications may need to use one kind of address in preference to another and the choice of which address to use might need to vary from one site to another. Wise configuration of networks may minimize the burden on applications, but this subject is not well-understood at the present time by IPv6 designers (much less network operators) and may not be well understood for many years.

9. There are some handy C pre-processor macros defined in netinet/in.h to help identify link-local, multicast, v4-mapped, v4-compatible, etc addresses:

   IN6_IS_ADDR_LINKLOCAL()
   IN6_IS_ADDR_V4MAPPED()

   See /usr/include/netinet/in.h for details.

10. There were also different kinds of unicast addresses in IPv4, notably public addresses and "private" addresses (RFC 1918), but in the IPv4 world, applications were less likely to be expected to be aware the differences between these kinds of addresses.

11. For the most part, developers should never have to worry about non-global addresses. DNS should never have link-local addresses for hosts and hosts on the same link should always refer to each other using globally-scoped addresses. An application that uses an IPv6 socket and connects to a IPv4-mapped address will connect via IPv4.

12. IPv6 hosts will routinely have multiple IPv6 addresses, and applications need to be able to deal with this. This happens in part because IPv6 networks are expected to be able to renumber periodically in order to keep routing efficient, but there are other reasons also. This potentially impacts a few long-running applications that expect host-to-address bindings to be stable.

13. As with IPv4, there is the notion of INADDR_ANY (= 0.0.0.0 for IPv4). An application wishing to bind to all interfaces on the local machine should bind to "in6addr_any" ("::"). Note that this is an externally defined constant data structure as opposed to a preprocessor macro (because of its size). Applications that include <netinet/in.h> will have access to the structure. Applications which bind and listen to the in6addr_any address will also get IPv4 connections, where the IPv6 address of the peer is an IPv4-mapped address.

14. The same goes for the loopback interface (::1), which uses "in6addr_loopback".

15. Because a host may have multiple IPv6 addresses it is necessary when establishing communications between two hosts to choose a source address (if the source host has multiple addresses) and a destination address (if the destination has multiple addresses). Normally this is done by APIs and is transparent to the application; however, the APIs can make poor choices. The problem is that in general there is no way for a host to know whether a particular (source, destination) address pair is better than another - since different choices may result in different routings through the network, and the host is not aware of network topology.

16. One penalty for poor choices may be that the application takes longer to connect - as it must try each of several (source, destination) address pairs before finding one that works. Another penalty may be reduced performance, if the API or application chooses an address pair that results in a slower or lossier network connection than would have been obtained by using a different address pair.

17. There may be circumstances where the application would do better to provide its own address selection code, particularly when the application (for whatever reason) is already maintaining its own estimates of reachability and connection quality.

18. There may frequently be circumstances where applications would do well to measure throughput via multiple address pairs and to choose the one that works the best. It may also be wise to design applications so that they can easily transition a session from one address pair to another, say by migrating from one TCP connection to another.

Comments:

- If you think this is broken, you're right. The current IPv6 architecture is really asking the host or application to do things that the network should be doing, and the result is to add an undesirable amount of complexity to some applications. Again, wise network configuration can significantly ease the burden on applications and the incidence of failure - but it may be dangerous for application maintainers to assume that networks will be configured wisely.

- The network should provide adequate connectivity regardless of which local address you are using. If a host has one source network that is better than another, then the same will go for IPv4 as it will for IPv6. This shouldn't be of concern to the application at all. If there is any preference for a particular local interface when connecting to a remote node, the OS is tasked with making the best decision. There is no difference in this case between IPv4 and IPv6.

IPv6-Only Operation vs. Mixed IPv4-IPv6 Operation:

1. It's one thing to modify an application to run only on IPv6, or to run in either of two modes - one where all nodes in the conversation use IPv4 and another where they all use IPv6. It's quite another thing for an application to support a mixture of IPv4-only nodes and IPv6-only nodes in the same conversation. While it's the case that many applications that only involve two nodes at a time, this is not the general case. Furthermore, even applications that only involve two nodes at a time can have the problem. For instance, users of an SMTP server that is only accessible from IPv6 will not be able to receive mail from users whose SMTP servers only support IPv4 unless special arrangements are made. On a web server which supports IPv4, but whose pages contain links to IPv6-only sites, users of IPv4-only browsers will not be able to follow those links.

2. One way to think of the mixed addressing problem is to treat it as a generalization of the IPv6 address pair selection problem. However, while it is generally feasible to configure IPv6 networks so that the default address selection algorithms produce acceptable results under most conditions, it is less feasible to provide IPv4 and IPv6 connectivity to all hosts in an application - particularly when those applications cross administrative boundaries.

3. An application that needs to communicate between IPv4 and IPv6 may therefore need some additional infrastructure or configuration beyond that required by a pure IPv4 or pure IPv6 application. For instance:

    - All electronic mail that will end up on an IPv6-only network (or a network isolated from the public IPv4 network) still needs to be reachable via an SMTP sever which accepts IPv4 traffic, and which is advertised as an MX (mail exchanger) for the recipient domain. All electronic mail originated from an IPv6-only network needs to be relayed through an SMTP server that can forward traffic to IPv4 SMTP servers.

    - All web sites hosted on IPv6-only networks should be reachable via IPv4 addresses if they expect to be usable by the public. One way to do this is to set up a TCP tunnel that listens on an IPv4 address and forwards incoming connections to the IPv6 address. The IPv4 address of the tunnel endpoint may then be associated with the DNS name of the web server. With HTTP/1.1 it is possible to have a proxy that listens on a single IPv4 address fronting for several IPv6 servers, using the Host: request header field to determine which of the IPv6 servers should handle the request.

Comments

- The default APIs tend to favor IPv6 over IPv4. This is often a mistake, as present-day IPv6 connectivity may be worse than that for IPv4 even between the same pair of hosts. This is particularly true for communications between IPv6 hosts using 6to4 and IPv6 hosts using native addresses, since these communications must usually travel through one of a small number of public "relay routers" resulting in very sub-optimal routes and increased potential for congestion.

- Favoring a remote node's IPv4 or IPv6 address is generally done by the order received in the DNS lookup (assuming a DNS lookup versus static addressing). If anything, IPv6 should encourage the use of DNS versus static IP configuration because addresses can be easily mis-typed or can change with IPv6 network re-mapping. If the IPv6 route is less reliable today, then the network needs to be fixed or ordering in DNS should reflect the preference. Applications really shouldn't worry about which addresses to choose.

- There will be very few IPv6-only applications/hosts out there. Everything built today should be dual-stacked and support both IPv4 and IPv6. Issues of connectivity between IPv4-only nodes and IPv6-only nodes can be addressed by proxy devices, but application developers really shouldn't have to worry about them.

Naming Issues:

1. If you chose to set up your DNS such that one name points to more than one host, all applications on those hosts have to support that configuration. If you are going to distribute an application among multiple hosts, the application must have its own way of synchronizing its state. This is really a general issue for distributed applications and has little to do with IPv4 versus IPv6.

2. Most applications use DNS names to refer to hosts, and this is often a convenient and technically acceptable convention. In a distributed application, this can be dangerous, as a DNS name does not necessarily refer to a single host. For instance, a DNS name can be associated with multiple addresses, each of which refers to a different host. It is common practice to do this for heavily used web and mail servers, so as to share load between several CPUs and also (on some occasions) to allow the service to continue operation in spite of local network outages. IPv6 increases the need to associate multiple hosts with a DNS name. For instance, the IPv6 addresses associated with a DNS name might be "really" bound to the host, whereas the IPv4 addresses associated with the same DNS name might be bound to some sort of proxy.

3. Say you're writing a distributed computation program and you'd like for Node A to be able to say to Node B: the result of computation X will be stored on Node C. What name do you use to refer to node C? It's risky to assume that the DNS name that either Node A or Node B associates with Node C uniquely refers to Node C, since there may be multiple IP addresses associated with Node C. What if Node B picks a different IP address for "Node C" than Node A picked, and reaches a different host? The data which was stored on the Node C that Node A picked may not be there.

4. For that matter, it can also be dangerous to use IP addresses to name hosts. Because of renumbering, there's no guarantee that an IP address that was once bound to Node C will continue to be bound to Node C.

5. These issues exist for both IPv4 and IPv6, but the two cases are somewhat different. IPv4 has widespread use of private addresses and NATs which makes addresses ambiguous if an application spans more than one private address realm. IPv6 does not have the equivalent of IPv4 private addresses (since site-local addresses were deprecated) and IPv6 addresses are unlikely to be ambiguous - however there may be a greater potential in IPv6 for a host-address binding to change while an application is running. As with IPv4, if you change an address of a host, existing connections may die. Don’t know if it is more likely with IPv6, but from the network admin’s perspective, it’s just easier to do with IPv6. I don’t think that is an issue that application developers really need to worry about.

6. Applications that are currently using DNS names to refer to host should re-examine that decision to see if it is still appropriate in an IPv6-only world or a mixed IPv4/IPv6 world.

7. Applications that are currently using IP addresses (either v4 or v6) to refer to hosts or nodes should consider defining their own internal identifiers to _name_ those hosts or nodes, and using IP addresses merely as potential ways to reach those hosts or nodes.