How Software Engineers can make their apps IPv6 Ready

26th June 2016 on Software Development, IPv6, Programming, IP by Christopher Demicoli

Fortunately for Software Engineers, the introduction of IPv6 is going to be less of an inconvenience than for Network Engineers. If you still need to understand the basics of IPv6, before this read my other blog post: Beginner's Guide to IPv6

That being said, it is becoming increasingly important, especially on mobile devices, to have your apps support IPv6. For example, on May 4th 2016, Apple declared that as of June 1st, all apps submitted to the AppStore **must** support **IPv6 only networking**.

The reason for Apple (and other tech giants) enforcing this is now clearer than ever; **some** mobile **ISPs have simply run out of IPv4 space** and are giving out only IPv6 addresses or Carrier Grade NATted IPv4.

Deployment Scenarios

An IPv6 ready app doesn't mean it has to work in IPv6 only networks. There are going to be various transition mechanisms to help the transition to IPv6 go smoothly. Your app needs to handle:

- **Networks with only IPv4**: Some networks still are IPv4 only and will remain so for a long time. You need to make sure that your apps continue to work even in IPv4 only networks.
- **Networks with only IPv6**: Some ISPs have already run out of IPv4 space. While some have opted to give Carrier Grade NAT to their customers, some have simply opted to deploy only IPv6. **The only way for these customers to connect to IPv4 networks is to pass through a 6to4 Tunnel** which essentials uses IPv6 as a transport layer for IPv4. The most obvious disadvantage is **a degraded experience to your end users**. Your app (especially hosted ones) should **accept IPv6 natively** for the best experience for these customers.

- **Networks that Dual Stack**: This is **the best transition mechanism possible**. In networks that have still enough IPv4 space the ISPs will start to give out both IPv4 and IPv6 connectivity at the same time. **You need to understand how the application behaves in a dual stack world**. In that situation, there are **two possible ways to connect**. A DNS name will resolve in at least two addresses. And you have multiple local addresses to choose from when setting up a connection. As long as things work fine, everyone will be happy. But if one of the networks has a problem, there will be long timeouts unless you consider this situation from start. If you are interested in reading more about this, search for **Happy Eyeballs** which **can make dual-stack applications more responsive to users**, avoiding the usual problems faced by users with imperfect IPv6 connections or setups.

  IPv6 is going to start to appear in EMail Headers, SIP, DNS, etc... Your application needs to be able to understand these addresses in URLs, header fields and data. And **you need to make sure that applications doesn't crash when meeting these 128 bits or longer text strings** for the first time.

Log Files, Databases, Regular Expressions and Operating Systems

For web servers, such as IIS, nginx, Apache, etc... you will start seeing IPv6 addresses pop up in your log files, referrers, remote IPs etc... It is very common to find **regular expressions to match IPv4 strings**; these will no longer work with IPv6. In fact, it is **no longer recommended to parse IPv6 addresses using Regex** given that IPv6 address can have very different formats. Ideally, you should use a library to do this job for you, for example the `IPAddress` in .NET and and `InetAddress` in Java.

Is your database schema ready to accept IP addresses of 128 bits in length instead of 32? Are you deploying your applications on **Operating Systems that support Dual Stacking**?

Libraries

Make sure that the libraries that you are using are also IPv6 Ready. For example, if you are using a library to parse User Input to IP address, the library may not have been yet updated. As Microsoft notes to their developers, **IPv6 addresses are unlike IPv4 addresses in that their length is unpredictable**. This has ramifications for nearly every type of user interface. Where a fixed space was previously allocated in a user interface for IPv4 addresses, that fixed space would necessarily not be sufficient for IPv6.

URLs

URLs (which are a kind of URI) are often used by network applications. For example, many mobile apps use URLs for RESTful communication to server-based applications, and those server-based applications might use URLs to communicate with LDAP services (for centralized authentication), databases, or other RESTful services.

As IPv6 addresses are textually expressed with colons (`:`) and the host and port are separated by a colon, placement of IPv6 addresses in URLs requires that they be escaped by square brackets (`[` and `]`). **The following is the correct way to formulate an HTTP URL with an IPv6 address: `https://[2001:500:4:13::125]:443/`**

DNS

If an application needs to connect to some service by name, it will use a DNS name resolver to translate that name into a list of destination addresses. It is worth mentioning that **the version of the IP protocol used to carry the DNS queries is independent from the protocol version of the addresses included in the DNS data records**. Which means that **you can connect through IPv4 to a DNS server in order to query for information related to IPv6 and the other way around**. Remember though: **being able to translate a name server into an IPv6 address doesn't mean that the application running on that server supports IPv6.**

In IPv4, the DNS record used to translate a hostname into an IPv4 address is called an A record. In IPv6, as IPv6 addresses are four times longer than IPv4 addresses, the new record used to translate a hostname into an IPv6 address is called the **AAAA record (sometimes called a "quad A" record). A single hostname could have both an A record and an AAAA record, therefore that hostname could be translated into an IPv4 address and into an IPv6 address.** Happy Eyeballs says that if you receive both an A and an AAAA record, you should try contacting the host using both addresses and measure the response time. Then you will establish a connection to that host using the IP address that responded first.

IP Geolocation

It is quite common for server-based applications to tailor services based on the location of the client, and one such method used is to locate the client geographically using the client's IP address (IP geolocation). In this case, changing of the address type from IPv4 to IPv6 may have some implications.

IP Geolocation works by taking the IP address of the system being geographically located and looking it up in a database. How the database is created and how accurate it is depends on the provider of the database. Depending on which IP geolocation database is in use, IPv6 may not be supported.

For some IP geolocation systems, **the API into the database has changed to support IPv6**, and **for the feed-based systems, which enable customers to locally store the data in a relational database such as MSSQL, there are likely schema changes needing to be made**.

Loopback and localhost

Some client applications send data or create connections specifically to other processes on the same node using IP sockets. Some server applications have special processing rules, such as authorization policies, that change if the software recognizes the

connection is from another process on the same node. This communication happens using an IP address known as the loopback address, usually referred to as `localhost`.

In IPv4, the loopback address can be any in the range between `127.0.0.0` and `127.255.255.255`, though `127.0.0.1` is almost always the address used. With **IPv6 there is only one loopback address, `::1`**. For clients, the most portable way to address a local process is to resolve the `localhost` hostname instead of using an IP address.

Testing

When you *think* you are finished, test, test and test.

If you have a Windows application that uses socket-level programming **you can use Microsoft's checkv4.exe Utility**. The Checkv4.exe utility is designed to provide you with a code porting partner; a utility that steps through your code base with you, identifies potential problems or highlights code that could benefit from IPv6-capable functions or structures, and makes recommendations.