IPv6: It's Not Your Dad's Internet Protocol

PAUL EBERSMAN

Paul Ebersman first worked on UNIX and TCP/IP for the Air Force in 1984, serving at the Pentagon. He was one of the original employees at UUNET and helped build AlterNet and the modem network used by MSN, AOL, and Earthlink. He has maintained his roots in the Internet and the open source community, working for various Internet infrastructure companies, including the Internet Systems Consortium and Nominum. Paul currently works in the Infoblox IPv6 Center of Excellence as a technical resource, both internally and to the Internet community. pebersman@infoblox.com

Functionally, IPv4 and IPv6 seem pretty similar. They connect a bunch of virtual wires or local networks together so that we can all run our favorite Internet apps. But once you start looking under the hood, there are some obvious differences. There are lots more subnets and addresses available in IPv6, and they are really big and ugly and hard to type (try "ssh 2001:db8:a32:8f3e:3c32:abcd:829:1a" without a typo).

But there are some more subtle twists and turns that may surprise you, and some old habits and "conventional wisdom" rules that we will all need to rethink and relearn. There are also some changes in how addresses are configured, which will change how we all design and build our networks.

Let's crack open the hood and take a peek.

Don't Get Tied Up in IPv4-Think

The Vast Reaches of Space

We are so used to constructing our networks from a patchwork of IPv4 ranges that we've begged, borrowed, or acquired that we've forgotten that a subnetting plan can actually make our lives *easier*. You can get enough IPv6 space to allow this.

Think about how you organize your company, define security zones, and delegate control. Then subnet in a way that follows those boundaries, your process, and your security model. It may be by department, function, or geography. Figure 1 provides a simple example of a location-based subnet plan.

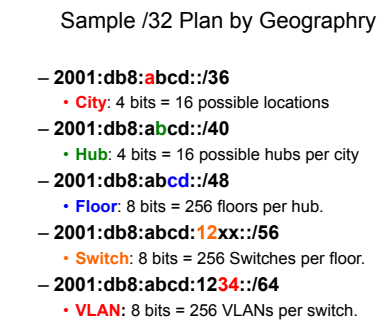

Figure 1: A subnetting scheme that takes advantage of the enormous addressing potential of IPv6

With the example in Figure 1, I can easily make ACLs at whatever level makes sense—by city, VLAN, etc.—because my subnet boundaries match my delegation scheme.

You can make your life easier by subnetting on nibble (4-bit) boundaries, just as we used to do subnet IPv4 address on 8-bit boundaries in classful subnets.

In IPv4, the 32 bits are written as four decimal characters, separated by dots: e.g., 192.168.1.1. Each one of those decimal characters is 8 bits. If you do subnets at 8-bit boundaries (where the prefix length is divisible by 8, such as /24, /16, /8), you don't have to "split" within any of those decimal characters.

With IPv6 and the written representation of the 128 bits as 32 hexadecimal characters (2001:0db8:0100:0000:1111:2222:3333:0001), if you use prefix lengths divisible by four (i.e., on nibble boundaries), you don't have to "split" within any of those 32 hexadecimal characters.

There's no technical reason you couldn't use non-nibble prefixes; it's just way easier for humans to not have to split any of those hexadecimal characters.

Hosts No Longer Require a Single Hard-Coded Default Route

With IPv6, host network stacks now deal gracefully with having multiple addresses on the same interface, with addresses coming and going without any user-visible disruption of most apps. Interfaces can have multiple default routes at the same time. And assignment of default routes is all done with router advertisement messages; there's no need to run an active routing protocol on each host. You may even be able to simplify your current HSRP/VRRP setup by having redundant default routes.

Changing My IP Address No Longer Breaks My Connections

In the old days of modems, you got users off a modem pool by letting existing calls stay up but not taking new calls. As active users hung up, you wound up with an empty modem pool for your maintenance window without having to hang up all of your user calls. We called this quiescence.

With IPv6, each host can have multiple addresses per network interface, choosing which to use based on RFC 3484 [1] and any local rule changes. Addresses come and go, with old addresses kept open for existing connections but not used for any new ones. As with modems, this quiescence allows new addresses to be used by the host without any user-visible disruption in existing connections.

RFC 1918: Can We Please Kill It?

The "conventional wisdom" is that using private addresses is "more secure," because you hide multiple users behind one single public IP address. The bad guys can't tell how many users you have and can't target one particular machine from the outside.

While this does add a layer of security, the reality is that we've convinced ourselves that RFC1918 [2] space is a security feature because we actually ran out of enough IPv4 addresses years ago, and using private address space and NAT let the Internet keep running. We've been trying to make the best of a bad situation.

With the abundance of IPv6 addresses, you can use public addresses in IPv6 for everything. Use ACLs to prevent accidental or malicious traffic from getting into

or out of your network. You know that everything has a unique address, so merging networks isn't nearly as painful. And you do have the stateful firewall option if you like things slow and complicated.

If you're bound and determined to use private addresses, IPv6 does have Unique Local Addresses (ULA). However, they have the same issues as RFC 1918 (not guaranteed to be unique, must be NAT'ed to get global connectivity, and break end-to-end apps).

DHCP: Not Just for Servers Anymore

In IPv4, you either statically configure all hosts, or you centrally manage your network and network policy via DHCP server. With IPv6, you can still statically configure hosts, but it's sure painful. The two new choices are SLAAC (StateLess Address AutoConfiguration) or DHCPv6.

With SLAAC, you push all the configuration to the edges. Each host is given certain information via router advertisement (RA) messages, such as default route and what network prefix(es) the host should create itself addresses from. Then the host configures its own network interfaces.

RAs can come from either a router, a switch port configured to send RA messages, or a server running RA daemon software (such as rtadvd).

With DHCPv6, much like with DHCPv4, the DHCP server provides the client host with most of the information it needs to configure itself. DHCPv6 can give you addresses, rDNS server, TFTP server, and lots of other configuration information, but it can't (currently) give you a default route.

RA messages give you default routes and prefixes but, until recently (RFC 6106 [3]), couldn't give the client a recursive DNS server. Even with RFC 6106, rDNS via RA support in DHCP clients is not yet widespread. And you still can't get NTP server, TFTP server, and other information via RA messages.

Another big change from DHCPv4 is that clients don't make DHCPv6 requests unless they are told to via RA message flags using the M (ManagedAddress) flag and/or O (OtherConfiguration) flag.

The end result is that you have to run both an RA and a DHCP server for most installations. That means implementing network policy requires both DHCP server configuration and router or switch configuration on every subnet with clients.

Mac Address vs. DUID

DHCPv4 uses the client MAC (Media Access Control) address as the unique identifier to the DHCPv4 server. The problems with MAC addresses are well known:

- Not guaranteed to be unique
- Can be changed or forged
- Doesn't identify a host with multiple network interfaces clearly

In spite of all these problems, there wasn't a better answer in IPv4, and we've all learned to deal with the problems. Most end hosts on the Internet use DHCP and MAC address-based identification.

For IPv6, the IETF decided to try to tackle the flaws with MAC-based identification by using a DUID (DHCP Unique IDentifier), one per DHCPv6 client and one per DHCPv6 server.

For every network interface on a host, there is an Identity Association (IA) that is the collection of all the addresses for that interface. The combination of DUID and IA for any given host/network interface uniquely identifies that interface. That allows you to use DHCPv6 for every interface, something that a MAC address-based system can't do.

DHCPv6 is a Layer 3 protocol (the client has a valid IPv6 link local address and uses a multicast IPv6 address to reach a DHCP relay/server). The client no longer has to send a broadcast using its MAC address; it sends the DUID/IA pair instead.

Because DHCPv6 clients don't need to broadcast with their MAC address in the DHCPDISCOVER packet, and DHCPv6 doesn't have the htype/claddr option that DHCPv4 uses to pass a MAC address through DHCP relays, there is no longer a way for the DHCP server to get the client's MAC address. This makes it hard to determine that a particular IPv4 and IPv6 address represent the same host.

Another problem is that you don't generate the DUID until the first time you start up DHCP (as server or client). It is not based directly on the MAC address, nor can you know in advance what the DUID will be until you contact DHCPv6 for the first time. This makes reserved addresses or host statements hard to pre-provision.

There is a proposal at the IETF to address this [5], but it will take time before these become standards and are available in production client and server code.

Failover

DHCPv4 failover was an attempt to deal with two issues: IPv4 client address changes tend to be user-visible, and the shortage of IPv4 addresses made pool sizes small/scarce. The failover protocol used by ISC was one attempt to solve these problems.

In IPv6, both issues are no longer relevant. Any IPv6-compliant network stack will deal gracefully with address changes with no user-visible impact. And a /64 subnet has 4 billion times 4 billion addresses. If you are running out of addresses in a network with that many to use, DHCP and failover aren't the problems you need to solve.

With IPv6, just assign half of the /64 to one DHCP server and half to the other. Yes, if server A is down and a client of server A needs an address, it will wind up getting a new IPv6 address from server B. But we've already established that new addresses aren't an issue for IPv6 clients. And 2 billion times 4 billion addresses is probably still sufficient for the subnet. If you want more redundancy or a remote server as backup, split the pool into three or four chunks and relay as needed.

This solves the problem of always having a working DHCP server on a subnet and avoids much of the fragility and complexity of trying to synchronize a lease file across a failover pair. The interim IETF draft that ISC based its failover on was never finalized in the IETF, due to these complexities. The odds are good that an IPv6 failover draft will not be finalized any time soon.

ICMPv6

Another place where the IETF has really taken lessons learned from IPv4 and improved things is ICMPv6. Function is much more precisely defined, making filtering by type much more accurate.

But ICMPv6 is more than just improved with IPv6; it's now a vital part of making things work. In IPv4, if you filter all ICMP regardless of type, you make your NOC's life miserable but things still mostly work. With IPv6, if you filter ICMPv6, you break:

- Duplicate Address Detection (DAD), the way you make sure two hosts aren't using the same address on the same LAN segment
- SLAAC RA messages
- DHCPv6
- Path MTU Discovery (PMTUD), the replacement for fragmentation of large packets
- Connectivity testing (echo request/response)
- Other network errors

There is a very useful RFC (RFC 4890 [4]) which has best current-practice recommendations on what to filter with ICMPv6.

Security

Meet the New Security Holes, Same as the Old Security Holes

While there are some complexion changes between IPv4 and IPv6 (arp cache corruption vs. neighbor cache corruption, rogue DHCP server vs. rogue router advertisements), most problems with IPv6 are the same as we've been living with in IPv4. Most issues with IPv6 are going to be due to lack of familiarity leading to misconfigurations.

There are some administrative issues—double work in ACLs in a dual stack environment, twice as many ports to check, etc.—nothing that says we should avoid IPv6 completely.

The big problem seems to be the impression that we don't already have IPv6 on our networks. If you're running a current OS X, Linux variant, FreeBSD, or Windows7/Vista/W2008, you probably have hosts using IPv6. If you specifically disable IPv6 completely in Win7/Vista/W2008, Microsoft considers this an unsupported configuration.

The best thing you can do for security is to turn on enough IPv6 that you can at least monitor and see it. Validate your firewalls with your vendor to ensure that they can detect and monitor IPv6, particularly various tunneling technologies. Get a lab up with IPv6 so that your staff starts to build experience and test procedures and toolsets.

IPsec Is Built into IPv6

Yup. Sure is. Per RFC, if you want to have a network stack that claims to support IPv6, you must support IPsec. Sadly, OS vendors have chosen to interpret this as meaning "We ship it with IPsec. If you want to actually enable it, knock yourself out."

The missing piece of IPsec is key management. There is no standard PKI infrastructure. This leaves you with the same situation as in IPv4. You have to configure the end nodes and the VPN/tunnel termination point with a shared secret. No worse than IPv4, no different from IPv4, just no better than IPv4 (at the moment).

The ray of hope here is Microsoft. Since Microsoft Windows authentication already has a built-in PKI, they can use the standard Windows login and credentials to configure a client with no extra effort on the user end. DirectAccess uses this to create an IPsec-secured tunnel from the end user to a W2008 server. The user logs in and is able to securely see any shared resources on the W2008 server as if on the local net, through firewalls, hotel NATs, etc.

If Microsoft can do this, surely the open source community can do it too.

Getting IPv6 on Your Network

How do you eat an elephant? One bite at a time.

By now, you're probably itching to get started implementing IPv6 in your network. The good news is that you don't have to do it all at once. You can do the phases as distinct projects and take time between them as you need to.

Start with an inventory of your IP address usage, subnetting, hardware, software, apps, and make sure everything can support IPv6. Upgrade or replace what can't (or figure out what transition technology might keep it limping along). Then decide on a new subnetting plan and network architecture, if needed.

You will next want to get the IPv6 addresses you need, working with your ISP(s) or applying directly to an RIR (Regional Internet Registry). Also make sure your ISP provides full IPv6 connectivity and is willing to route your new prefix (if you got it directly from the RIR).

For most companies, the easiest place to start is their external Internet site:

- Ensure thatyour firewall/IDS is IPv6 ready.
- Get IPv6 connectivity to your site.
- Create a test site where you can run IPv6 experiments separate from your current Internet-exposed production servers.
- Test in IPv4 to validate it reproduces your production site.
- Convert the test site to IPv6 only, put NAT64/DNS64 in front of the test site, and see if it all still works.
- Remove the NAT64/DNS64, go to IPv6 only, and see what was using IPv4 that you thought was using IPv6.
- Do a beta version of your production site using a subdomain (ipv6.example.com).
- When everything works for your beta testers, add the AAAA record for www .example.com.

Once you know that your production servers can work with IPv6, and have implemented IPv6 on your external Internet presence, you can work on adding IPv6 on your internal (or more complicated) environment:

- Ensure that your firewall/IDS is IPv6 ready (if it's different from your production site).
- Get IPv6 connectivity to your site.
- Get IPv6 running on just your core networking gear and switches.
- Make sure your network monitoring and logging infrastructure can monitor IPv6.
- Simulate your internal site in a lab as you did with your external site. Do the IPV4/IPV6-NAT64-DNS64/IPv6-only dance.
- Work with your vendors on any bugs or lack of features.
- Rinse and repeat until it all works.

Conclusion

These are some of the issues you will encounter as you implement IPv6. There will probably be others, although I'd expect most of those to be hidden in internally written code rather than in OS stacks or large commercial products. But the techniques and experience you'll gain dealing with the topics discussed here, especially as you run pilot IPv6 operations in labs, will make you much better prepared to find them in your own code.

IPv6 requires learning about the differences between the more familiar IPv4 and IPv6, as you will undoubtedly be using both for years to come. And, in some ways, it is 1994 all over again: you will be venturing into an unfamiliar networking technology, with a new set of warts to figure out how to overcome.

References

[1] http://tools.ietf.org/html/rfc3484.

[2] http://tools.ietf.org/html/rfc1918.

[3] http://tools.ietf.org/html/rfc6106.

[4] http://tools.ietf.org/html/rfc4890.

[5] http://tools.ietf.org/html/draft-halwasia-dhc-dhcpv6-hardware-addr-opt-00.txt.