Autoconfiguration of IPv6 Networks

Mark Kamichoff

Internet Protocols

ECSE-6600

April 15, 2004

Contents

1 Internet Protocol Version 6

1.1 Introduction to IPv6

At the time when the Internet was created, nobody predicted that it would become the de facto medium for business transactions, communication, shopping, and entertainment. The original ARPANET and NSFNET networks were created to facilitate communication between military personnel and educational institutions. The IPv4 protocol, specified in RFC 791, was standardized in 1981, and provided 32 bits of address space. As corporations and home users rapidly jumped on the "information superhighway," it was clear that this implementation was not going to work. Within time, CIDR and NAT were developed to more efficiently utilize the IPv4 address space, but these were only short-term workarounds, not solutions. Additionally, with over 26% of the IPv4 allocations belonging to the United States, the world is definitely in need of a new protocol.

The IPv6 protocol, first standardized in RFC 1883 (now obsoleted by RFC 2460) and written in 1995, brings 128 bits of address space to the Internet, in addition to built-in multicast routing capabilities and privacy extensions. IPv6 headers are of a fixed length, with flexible extension headers replacing the need for variable header fields that were present in IPv4. Neighbor discovery and ICMPv6 replace ARP, IGMP, and router discovery, making it easier to locate and communicate with hosts sharing the same link.

1.2 Deployment and Transitioning Issues

Even though there are numerous benefits of this new protocol, many organizations and countries have been reluctant to put forth a serious effort and begin the migration from IPv4. Since IPv6 was slow in development, many "hacks" and additional features were added to IPv4, such as CIDR and NAT, to more optimize the use of the slowly decreasing address space. Due to these advancements in IPv4, the imminent address shortage has been temporarily averted, and more organizations have time to procrastinate about moving to IPv6.

Although the IPv6 protocol has been standardized in RFC 2460, DNS records are still in somewhat of a flux. Since the current DNS was not capable of handling 128-bit addresses, new records, AAAA and A6 were developed to solve this problem. An AAAA record is similar to an IPv4 A (address) record, but handles four times the bytes. A6 and DNAME records, specified in RFC 2874, on the other hand, are an alternate replacement for IPv4's A records, but facilitate network prefix renumbering, assuming the change of ISP or similar. Instead of an A6 record holding a complete IPv6 address for a host, it only contains the 64-bit (can be variable) interface identifier of the host. Queries must follow the chain of A6 records from the hosts domain name to the TLA ID. A number of prominent authors such as D.J. Bernstein, creator of djbdns, have spoken out against this method, due to its plethora of lookups and possibility of loops. The IETF has supposedly reconsidered their decision to implement this system with A6 and DNAME records.

Reverse DNS for IPv6 has been a rocky ride as well, due to the (still) competing methods for solving the 32 to 128-bit transition. The `IP6.INT` domain was created and delegated to the 6bone and production IPv6 Internet. This method works similarly to how IPv4 reverse-lookups are made, reversing the order of the nibbles and appending IP6.INT at the end. However, `IP6.ARPA` was created in RFC 3152 and bitstring labels were introduced along side A6 and DNAME. The following examples show how each of the potential queries would look for the IPv6 address of `2001:470:1f00:181::1`.

```
IP6.INT method:

1.0.0.0.[snip].1.8.1.0.0.0.f.1.0.7.4.0.1.0.0.2.ip6.int.  IN PTR

IP6.ARPA method:

1.0.0.0.[snip].1.8.1.0.0.0.f.1.0.7.4.0.1.0.0.2.ip6.arpa.  IN PTR

IP6.ARPA with bitstring labels:
```

```
\[x200104701F000181000000000000000001/128].ip6.arpa. IN PTR
```

It is not clear at this point whether `IP6.ARPA` or `IP6.INT` will end up becoming the standard.

A number of systems were developed and provided to provide for the major transitioning hurdle from IPv4 to IPv6. Most of these techniques use IPv6-in-IPv4 tunnels (IP protocol 41) to allow IPv4 hosts to connect to the 6bone or production IPv6 Internet. 6to4, a method that is primarily intended for single hosts, uses the `2002::/16` IANA-assigned prefix and the hexadecimal representation of the IPv4 address to create a globally scoped IPv6 address. For example, an IPv4 address of `128.113.151.16` would become `2002:8071:9710::` with a prefix length of 48 bits. This IPv4 host then communicates with a public 6to4 relay using the IPv4 anycast address of `192.88.99.1`, which is advertised via BGP from multiple locations. 6to4 does allow for routing multiple hosts on a LAN through a router running a 6to4 virtual interface, but this does not seem to be very popular, considering that latencies to 6to4 relays may be less than adequate. Another method, involving tunnel brokers, allows a user to setup an IPv6-in-IPv4 tunnel to an ISP which provides an endpoint and some IPv6 address space. The address space given to the user can be assigned to a LAN and be globally routable, potentially with its own reverse-DNS delegation. Hosts on these LANs may use stateless autoconfiguration, a method used to automatically assign IPv6 addresses in a quick and simple manner. This, as well as stateful autoconfiguration, is discussed in greater detail in the next section.

Regardless of the rampant deployment issues, most modern operating systems and router software now fully support the IPv6 protocol. Router manufacturers such as Cisco Systems, Juniper Networks, Foundry Networks and Extreme Networks and operating systems such as Microsoft Windows XP, Linux, Unix, and Macintosh OS X all completely support the protocol as well as stateless autoconfiguration. For home users running Windows XP, a command as simple as “ipv6 install” from the command line will install the IPv6 stack, bring up a virtual interface, and obtain a 6to4 address for immediate use.

2 IPv6 Autoconfiguration

2.1 Introduction to IPv6 Autoconfiguration

Since IPv6 has been developed to completely replace IPv4 for every host on the planet, is follows that there be a method for hosts to obtain IPv6 addresses automatically. At this point, IPv4 addresses can be autoconfigured using DHCP, which is a stateful method and uses the client/server model. However, IPv6 addresses can be autoconfigured using both stateless and stateful (DHCPv6) systems, often simultaneously.

While stateless IPv6 autoconfiguration has been quite popular on dual-stacked hosts, it is rather limited, and was arguably a “temporary” solution while DHCPv6 was being developed. Since stateless autoconfiguration is prevalent in IPv6 LANs today, DHCPv6 is capable of providing extended parameters and values for hosts that have already received a IPv6 address through other means. The next two sections will detail the two protocols available for assigning IPv6 addresses.

2.2 Stateless Autoconfiguration

IPv6 stateless autoconfiguration (documented in RFC 2462) is a fairly simple protocol, but not a very complete one. Most of the work is actually done by the end host that wishes to obtain an IPv6 address on the network. A host appears on the network, creates an interface identifier, then obtains a network prefix that is prepended to the interface identifier. Using the link-local address of the router, the host has instant IPv6 connectivity. Most of the communication in stateless autoconfiguration uses the ICMPv6 protocol, which has been greatly enhanced for these functions, compared to ICMPv4.

The first task in stateless autoconfiguration is for the host to create a unique interface identifier to be used for any link-local, site-local, or global addresses. This interface identifier, called EUI-64, is 64 bits that are adapted from the device's layer two address, typically MAC-48 in LANs. The IEEE dictates that the conversion from MAC-48 to EUI-64 be carried out by inserting a value of `0xfffe` after the first 24 bits of the MAC address. Assuming `0xX` (the company ID) and `0xY` are hexadecimal values of the original MAC-48 address, the EUI-64 identifier would be computed as `XX-XX-XX-FF-FE-YY-YY-YY`. Most systems would then insert a binary one at bit seven (counting from the right) to indicate a global scope. The following example illustrates this.

```
MAC-48: 00:01:03:69:8B:CF

EUI-64: 0201:03ff:fe69:8bcf
```

Using a MAC-48 layer two address is only one example of computing an interface identifier. A EUI-64 identifier can be created for layer two addresses up to 118 bits, and for addresses larger than that, autoconfiguration fails.

The interface identifier is then appended to the prefix `fe80::/10`, which is reserved for link-local addresses. Before this, or any other unicast address, can be assigned, the end host must perform duplicate address detection, or DAD, to ensure the address is unique to the link. The host then joins the all-nodes and solicited-node multicast addresses using the tentative address. With neighbor solicitation and advertisement messages, the host can determine if the address is unique to the link. The following tcpdump output illustrates a case where DAD finds an address conflict (`ff02::1` is the all-nodes multicast address).

```
15:02:53.862965 :: > ff02::1:ff1c:bb3a: icmp6: neighbor sol: who has

3ffe:b80:d6e:2:250:56ff:fe1c:bb3a

15:02:53.863213 3ffe:b80:d6e:2:250:56ff:fe1c:bb3a > ff02::1: icmp6:

neighbor adv: tgt is 3ffe:b80:d6e:2:250:56ff:fe1c:bb3a
```

If no neighbor advertisements are received, the host assumes the address is unique. DAD is required to be used on all new unicast addresses not derived from the same interface identifier. Additionally, DAD can be disabled (DupAddrDetectTransmits ¡= 1) via the operating system, since some administrators believe it generates unneeded overhead, and MAC-48 addresses *should* be unique. In this situation, an address conflict could possibly be undetected, and communication between hosts might mysteriously fail. If not using DAD, a prudent administrator should create a registration system to check MAC-48 addresses before they are allowed on the network.

Once the host has build a unique interface identifier (and a link-local address), the host then uses the link-local address to send a router solicitation to `ff02::2`, the all-routers multicast address. The router sends a router advertisement to the all-nodes multicast address with a network prefix, preferred and valid lifetimes, and a MTU for the link. The preferred lifetime is the time until the address is deprecated in favor of a new preferred address. This is most often used in conjunction with IPv6 privacy extensions, covered later in this section. The prefix is prepended to the interface identifier and a new, often globally routable, IPv6 address is added to the interface. Since the interface identifier is the same used in the link-local address, DAD is not performed.

Stateless autoconfiguration is available on many devices today, including most Unix systems, routers, and Windows-based machines. Additionally, software such as Radvd (router advertisement daemon) and the Zebra/Quagga software routing suites can provide router advertisements for a link.

One may note that stateless autoconfiguration does not provide the many common options that DHCPv4 (or DHCPv6) provides, such as DNS/NTP servers or a domain/host name to be used by the client. For this reason, stateless autoconfiguration is popular for dual-stack hosts where there is already an IPv4 network, and stateful IPv4 autoconfiguration has been performed.

Even though stateless autoconfiguration is quite simple, it is not without security issues. There is no authorization to obtain an address on the network, and this has supposedly been addressed in RFC 2402, which states that neighbor discovery packets can be authenticated. If DAD is not used, the host can silently become a part of the network, without one packet being transferred. This is possible, although unlikely, because many routers send out router advertisements on a periodic interval, regardless of whether a router solicitation is received. A DoS with DAD is always a possibility, if neighbors respond to a neighbor solicitation indicating a duplicate address, even though no such conflict exists. The victim's operating system will, in some cases, invalidate what it believes to be a duplicate address, and the host will lose connectivity. However, this is not a particularly new problem, since certain ARP spoofing techniques can achieve the same goal with IPv4.

According to Microsoft, since interface identifiers do not change when a mobile host moves between subnets on a network, privacy can be an issue for anyone snooping network traffic. RFC 3041 was developed, which details privacy extensions for use with stateless autoconfiguration. In addition to the IPv6 address generated via the interface identifier, a new IPv6 address with a "scrambled" interface identifier is suggested, using MD5 hashing. Using a 64-bit history value, which can be held in stable storage or generated at random, the MD5 hash is computer for the history value plus the interface identifier. Taking the leftmost 64 bits of the hash and setting bit six to zero creates the *anonymous* address. Since the interface identifier is new, DAD is used. This anonymous address is changed frequently, depending on usage.

Using privacy extensions helps solve the privacy problem with constant interface identifiers, but can make it more difficult to diagnose network problems. When a host's IPv6 address changes randomly, it is almost impossible to troubleshoot the network when there are no set addresses for outgoing connections from hosts. Additionally, some Internet services will refuse access to hosts without PTR records, which anonymous addresses will end up lacking.

2.3 Stateful Autoconfiguration (DHCPv6)

Due to the various limitations of stateless autoconfiguration, a new DHCP standard was produced to work with IPv6. The DHCPv6 RFC, submitted in July of 2003, proposes an (almost) entire rewrite of DHCPv4, complete with authentication and interoperability with stateless autoconfiguration. DHCPv6 is called "stateful" since there is bidirectional and (somewhat) reliable communication between the client and server.

As is similar in DHCPv4, DHCPv6 also uses UDP messages to communicate with clients and servers, claiming ports 546 and 547, respectively. Clients, instead of broadcasting, communicate with the DHCPv6 servers via reserved multicast addresses. `ff02:1:2` is the link scoped address for `All_DHCP_Relay_Agents_and_Servers` and `ff05::1:3` is the site scoped address for `All_DHCP_Servers`. This assumes each client has a working link-local address and has performed some address collision detection, usually DAD, prior to communicating with the DHCPv6 server.

In DHCPv4, clients are uniquely identified by their MAC address, a DHCP client identifier, or some other means. In certain environments DHCPv4 servers may randomly generate addresses for clients, not saving any identifiers. However, DHCPv6 is more stringent about such client identifiers. A DHCPv6 unique identifier (DUID) is required on the client side when negotiating options and addresses from a DHCPv6 server, and is passed to the server as a variable-length option. Being smaller than 128 octets, a DUID can be generated using a combination of the link-local address and a timestamp, a vendor-assigned unique ID based on some enterprise number, or solely from the link-local address if the client lacks stable storage.

DHCPv6 is capable of being used under two circumstances, with or without stateless autoconfiguration. If stateless autoconfiguration is already used on the link, a DHCPv6 client can be used to just obtain DHCPv6 options only, such as DNS and NTP servers. If DHCPv6 is used to obtain options, addresses, and routes, the process becomes slightly more complex, and uses link-local and well-known multicast addresses to communicate with the DHCPv6 servers.

All DHCPv6 messages are required to be either multicast or unicast. Clients send messages to `All_DHCP_Relay_Agents_and_Servers` and then can communicate to a specific server using unicast or continue using multicast. As in DHCPv4, the burden of retransmission is put on the client. To alleviate a possible denial of service of the DHCPv6 server in the case of a power failure where a large amount of hosts are powered up at the same time, messages from the client are delayed by a random time before being sent. All client and server messages use the same format, an 8-bit message type with a 24-bit transaction ID and a variable length of options. The transaction ID is used to synchronize server responses to client messages, and must be fairly unique to minimize security issues.

If DHCPv6 is to be used beside stateless autoconfiguration, only two messages (assuming a lossless link) are required. The messages originating from the client must use the link-local address instead of any other address obtained previously from stateless autoconfiguration. The client then sends an information request with its DUID to `All_DHCP_Relay_Agents_and_Servers` and gets a reply with the DHCPv6 options.

In the case where a client needs an address in addition to DHCPv6 options, four messages are required. A solicit message is sent to `All_DHCP_Relay_Agents_and_Servers` initially, and a DHCPv6 server should respond with an advertise message. Choosing a server, the client proceeds to send a request message, complete with a DUID and IA identifier, and the server replies with an IA (identity-association) and options. An IA contains IPv6 addresses for the client, and can be applied to exactly one interface. The client's IAID is required to be consistent across DHCPv6 client restarts.

2.4 Design Considerations

When deploying IPv6 in an organization, a number of decisions about autoconfiguration must be made. Some factors that could potentially influence the decision include the amount of machines, heterogeneity of operating systems, and frequency of mobility. In general, organizations will either opt to use DHCPv6, stateless autoconfiguration, or a combination of the two.

For most organizations that already provide IPv4 connectivity and autoconfiguration for their hosts, stateless autoconfiguration is most likely the best choice. Almost all modern operating systems with an IPv6 stack support stateless autoconfiguration, and the required configuration is usually minimal, if it exists at all. However, since stateless autoconfiguration is required to use a `/64` or larger prefix, this can be quite wasteful or impossible for organizations with a small number of hosts per subnet or ones that have been only delegated a `/64` by their ISP. However, such a waste of abundant resources in networking is quite common, since the scarce resources are optimized.

If an organization wishes to only use IPv6 connectivity, DHCPv6 starts to look like a better choice, due to the limits of stateless autoconfiguration. However, in this case, both methods will require some configuration changes on each host on the network, either updating DNS server settings or installing a DHCPv6 client. This method will allow smaller prefixes to be used on subnets, therefore making more efficient use of the address space.

A third method of autoconfiguration that promises to scale the best is a combination of DHCPv6 and stateless autoconfiguration. Most of the hosts on a LAN would use stateless autoconfiguration, while newer hosts could start migrating to DHCPv6 without needing any changes in network services.

Even though DHCPv6 software is still young, there are two projects that show some promise. The “Dibbler” project aims at providing a portable DHCPv6 implementation, currently providing binaries for Linux 2.4/2.6 and Windows XP. Proposed ports include *BSD and Windows 2000.

The other project, entitled simply “DHCPv6” is developing a client and server implementation for Linux, and is hosted at SourceForge. Once this software has matured, vendors will most likely start integrating DHCPv6 clients and servers into operating systems, paving the way for IPv6-only networks gaining in popularity.

3 References

- http://www.faqs.org/rfcs/rfc2460.html

  RFC 2460 - Internet Protocol, Version 6 (IPv6) Specification

- http://www.faqs.org/rfcs/rfc2462.html

  RFC 2462 - IPv6 Stateless Address Autoconfiguration

- http://www.faqs.org/rfcs/rfc3041.html

  RFC 3041 - Privacy Extensions for Stateless Address Autoconfiguration in IPv6

- http://www.faqs.org/rfcs/rfc3315.html

  RFC 3315 - Dynamic Host Configuration Protocol for IPv6 (DHCPv6)

- http://klub.com.pl/dhcpv6/

  Dibbler, a portable DHCPv6 implementation

- http://dhcpv6.sourceforge.net/

  DHCPv6