IPv6 Internals

by Iljitsch van Beijnum

This article discusses some of the protocol details you should be aware of when planning a transition from IPv4 to IPv6. Although it is not intended as a complete step-by-step guide, this article explains the differences between IPv4 and IPv6 as they relate to actually operating a network. Vendor- and operating system-specific details can be found in the book from which this text was adapted, and further information is available in the references.

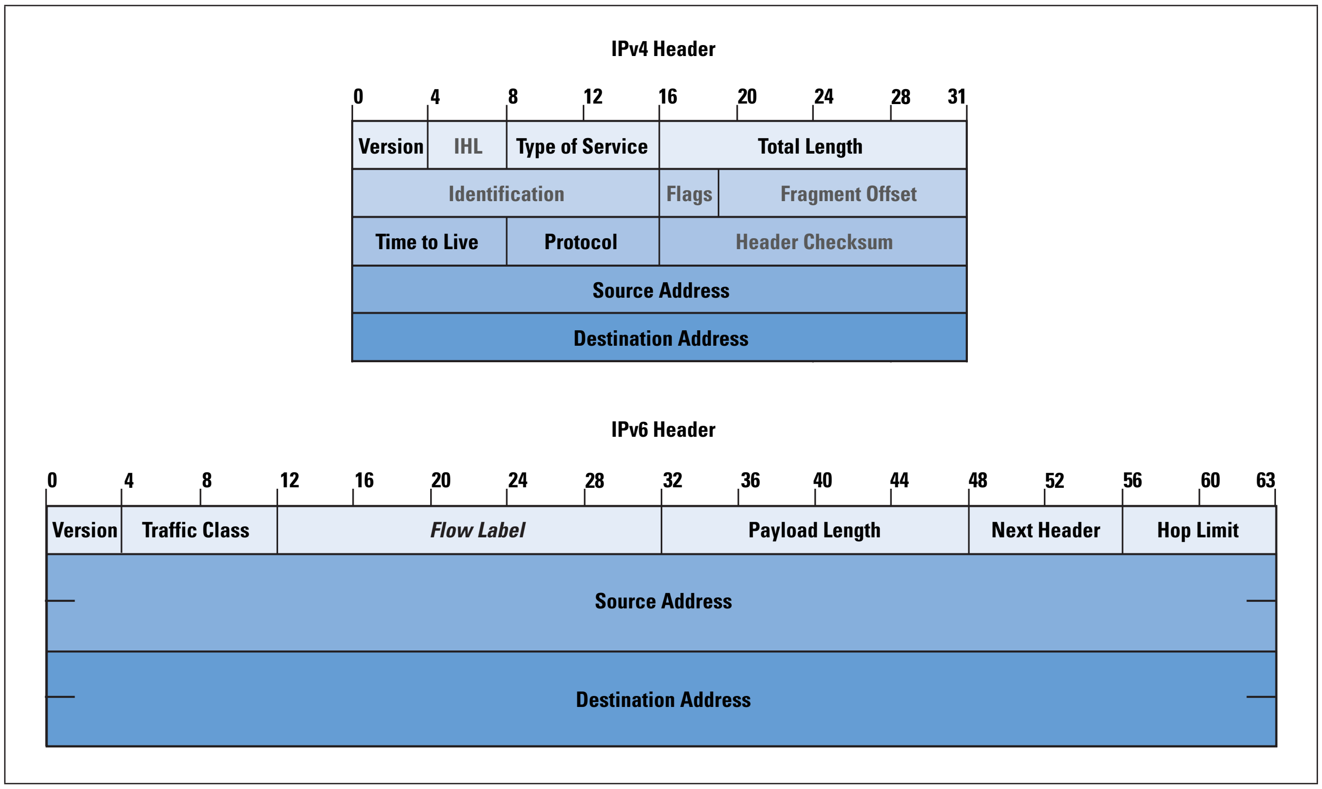
The easiest way to observe—in action—the mechanisms discussed in this article is to set up an IPv6 router on the local subnet and enable IPv6 on the operating system of your choice, if it is not enabled by default. If "native" IPv6 connectivity is not possible, you can set up automatic IPv6 tunneling or use a manually configured IPv6-in-IPv4 tunnel. Getting portable IPv6 address space from a *Regional Internet Registry* (RIR)[1] is a topic worthy of its own article, but *6to4*[2] creates 65,536 IPv6 subnets from a single IPv4 address, and service providers that provide IPv6 connectivity—either natively or over manually configured tunnels—are usually quite generous with IPv6 address space. However, you need to renumber when changing *Internet Service Providers* (ISPs), or when changing IPv4 addresses with 6to4. Most router vendors currently support IPv6 routing, but all widely used general-purpose operating systems can also route IPv6.

When you have IPv6 connectivity, the browser that comes with your system should be able to work over IPv6 (visit **`http://www.kame.net/`**), and there are v6 versions of *ping* and *traceroute* (called *ping6* and *traceroute6*) to determine IPv6 connectivity. More and more applications work over IPv6, but many still do not.

Differences Between IPv4 and IPv6

All knowledge about IPv6 begins with studying the IPv6 header format and the ways in which it is different from the IPv4 header format. Even though at the time the IPv6 specifications were written 64-bit CPUs were rare, the IPv6 designers elected to optimize the IPv6 header for 64-bit processing. For this reason, I have drawn the IPv6 header 64 bits wide in Figure 1, a little different from the way it is usually depicted. Because 64-bit CPUs can read one 64-bit-wide memory word at a time, it is helpful that fields that are 64 bits (or a multiple of 64 bits) wide start at an even 64-bit boundary. Because every 64-bit boundary is also a 32-bit boundary, 32-bit CPUs aren't affected negatively by 64-bit optimization. The IPv4 header is presented in the usual form that highlights its 32-bit background.

Figure 1: The IPv4 and IPv6 headers



The fields in the IPv4 header that are not present in the IPv6 header have gray text; the field that is present in IPv6 but not in IPv4 is shown in italic. The changes from IPv4 to IPv6 follow:

- *Version* now always contains 6 rather than 4.
- The *Internet Header Length* (IHL) field that indicates the length of the IPv4 header is no longer needed because the IPv6 header is always 40 bytes long.
- *Type of Service* is now *Traffic Class*. The original semantics of the IPv4 Type of Service field have been superseded by the *diffserv* semantics per RFC 2474[3]. However, in IPv4, both interpretations of the field are in use (although most routers either cannot or are not configured to look at the field anyway). The IPv6 RFCs do not mandate a specific way to use the Traffic Class field, but generally the RFC 2474 diffserv interpretation is assumed.
- The *Flow Label* is new in IPv6. The idea is that packets belonging to the same stream, session, or flow share a common flow label value, making the session easily recognizable without having to look "deep" into the packet. Recognizing a stream or session is often useful in *Quality of Service* mechanisms. Although few implementations actually look at the flow label, most systems do set different flow labels for packets belonging to different TCP sessions. A zero value in this field means that setting a flow label per session is either not supported or not desired.

- The *Total Length* is the length of the IPv4 packet including the header, but in IPv6, the *Payload Length* does not include the 40-byte IPv6 header, thereby saving the host or router receiving a packet from having to check whether the packet is large enough to hold the IP header in the first place—making for a small efficiency gain. Despite the name, the Payload Length field includes the length of any additional headers, not just the length of the user data.
- The *Identification, Flags,* and *Fragment Offset* fields are used when IPv4 packets must be fragmented. Fragmentation in IPv6 works very differently (explained later), so these fields are relegated to a header of their own.
- *Time to Live* (TTL) is now called *Hop Limit*. This field is initialized with a suitable value at the origin of a packet and decremented by each router along the way. When the field reaches zero, the packet is destroyed. This way, packets cannot circle the network forever when there are loops. Per RFC 791[4], the IPv4 TTL field should be decremented by the number of seconds that a packet is buffered in a router, but keeping track of how long packets are buffered is too difficult to implement, regardless of buffering time. The new name is a better description of what actually happens.
- The *Protocol* field in IPv4 is replaced by *Next Header* in IPv6. In both cases, the field indicates the type of header that follows the IPv4 or IPv6 header. In most cases, the value of this field would be 6 for TCP or 17 for the *User Datagram Protocol* (UDP). Because the IPv6 header has a fixed length, any options such as source routing or fragmentation must be implemented as additional headers that sit between the IPv6 header and the higher-layer protocol such as TCP, forming a "protocol chain."
- The IPv4 *Header Checksum* was removed in IPv6.
- The *Source Address* and *Destination Address* serve the same function in IPv6 as in IPv4, except that they are now four times as long at 128 bits.

All IPv6 hosts and routers are required to support a maximum packet size of at least 1280 bytes. For lower-layer protocols that cannot support a *Maximum Transmission Unit* (MTU) of 1280 bytes, the relevant "IPv6 over ..." standard must have a mechanism to break up and reassemble IPv6 packets so that the minimum of 1280 bytes can be accommodated. In IPv4, the official minimum size is 68 bytes—too low to be workable.

Checksums

In IPv4, the IP header is protected by a header checksum, and higher-layer protocols generally also have a checksum. The checksum algorithm for the IPv4 header, *Internet Control Message Protocol* (ICMP), ICMPv6, TCP, and UDP is the same one's complement addition, except that in IPv4, UDP packets may forego checksumming and simply set the checksum field to zero. In IPv6, this practice is no longer allowed: UDP packets must have a valid checksum.

The TCP, UDP, and ICMPv6 checksums are computed over a "pseudoheader" and the TCP, UDP, or ICMPv6 header, and user data, respectively. The pseudoheader consists of the source and destination addresses, the upper-layer packet length, and the protocol number. Including this information in the checksum calculation ensures that TCP, UDP, or ICMPv6 do not process packets that were delivered incorrectly, for instance, because of a bit error in the IP header.

IPv6 no longer has a header checksum to protect the IP header, meaning that when a packet header is corrupted by transmission errors, the packet is very likely to be delivered incorrectly. However, higher-layer protocols should be able to detect these problems, so they are not fatal. Also, lower layers almost always employ a *Cyclic Redundancy Check* (CRC) to detect errors.

Extension Headers

To allow special processing along the way, IPv4 allows extension of the IP header with one or more options. These options are rarely used today, both because they do not really solve common problems and because packets with options cannot be processed in the "fast path," and many routers and firewalls block some or all options. Not unlike the checkout counters at a grocery store, many routers have several "paths" that packets may follow: a fast one, implemented in hardware or highly optimized software, that supports only the most common operations (no checks), and one or more slower paths that use more advanced but slower software code that supports less common operations such as looking at IP options. However, many modern routers have only a fast path, so using additional features does not lead to a performance penalty.

Because the header is of fixed length in IPv6, options cannot be tagged onto the IP header as in IPv4. Instead, they are put in a header of their own that sits between the IPv6 header and the TCP or UDP (or other higher-level protocol) header. The most common extension headers follow:

- *Hop-by-Hop Options:* See the section that follows.
- *Routing:* This header is similar to the *Source Route* option in IPv4.
- *Fragment:* This header is used for fragmentation; see later in this article.
- *Authentication:* This header authenticates the user data and most header fields.
- *Encapsulating Security Payload* (ESP): This header encrypts or authenticates user data.
- *Destination Options:* See the section that follows.

The Hop-by-Hop Options and Destination Options headers are container headers: they have room for multiple suboptions. The Hop-by-Hop Options are processed by all routers along the way. All other options are normally ignored by routers and processed only by the destination. Obviously firewalls, or routers configured to perform filtering, may also look at these options. The Hop-by-Hop Options, Routing, Fragment, and Destination Options extension headers are defined in RFC 2460[5]. The Authentication and ESP extension headers are part of *IP Security* (IPsec).

Note that there is no standard extension header format, meaning that when a host encounters a header that it does not recognize in the protocol chain, the only thing it can do is discard the packet. Worse, firewalls and routers configured to filter IPv6 have the same problem: as soon as they encounter an unknown extension header, they must decide to allow or disallow the packet, even though another header deeper inside the packet would possibly trigger the opposite behavior. In other words, an IPv6 packet with a TCP payload that would normally be allowed through could be blocked if there is an unknown extension header between the IPv6 and TCP headers.

ICMPv6

The IPv6 version of the ICMP generally serves the same purposes as its IPv4 counterpart, but there are some changes. In IPv4, when a router or the destination host cannot process the packet properly, it sends back an ICMP error message along with the original IP header and the first 8 bytes of the higher-layer header. For UDP and TCP, this is enough for the source of the original host to see which TCP session or UDP association generated the offending packet. Because IPv6 supports an arbitrary number of extension headers between the IPv6 header and the higher-layer header, ICMPv6 returns as much of the original packet as will fit in the minimum MTU size of 1280 bytes. In addition to error messages, which are recognizable by an ICMP type of 127 or lower, there are also informational messages, with a type of 128 or higher. Because informational messages are not the result of an error, they do not include an original packet or part thereof. The most common ICMPv6 message types follow:

1:	Destination unreachable
2:	Packet too big
3:	Time exceeded
4:	Parameter problem
128:	Echo request
129:	Echo reply
130:	Multicast listener query
131:	Multicast listener report
132:	Multicast listener done
133:	Router solicitation
134:	Router advertisement
135:	Neighbor solicitation
136:	Neighbor advertisement
137:	Redirect message

ICMP and ICMPv6 messages also include a "code" that indicates the exact nature of the ICMP message within a certain type. As with ICMP, ICMPv6 calculates a checksum over the control message, but unlike ICMP, the ICMPv6 checksum calculation also includes a pseudoheader. Another departure from IPv4 is the fact that hosts and routers are required to limit the number of ICMPv6 messages they send. So if a router receives 100 packets per second toward an unreachable destination, it is not supposed to send back ICMPv6 packets at the same rate of 100 per second. The ICMPv6 redirect message works slightly different from the ICMP redirect message in IPv4. Like its IPv4 counterpart, the ICMPv6 redirect can be used by a router to inform a host that it should use a different router to reach the destination in question. But routers can also use the IPv6 Redirect to tell a host that the destination is reachable on the local subnet. Thus two hosts that have addresses in different prefixes can communicate directly after receiving redirects from a router.

Neighbor Discovery

When a system wants to send an IPv6 packet to another system connected to the same subnet or link, it needs to know what MAC address (or "link address" in the new IPv6 terminology) it should address the packet to, unless the interface in question is a point-to-point interface. Neighbor discovery allows systems to discover each other's MAC addresses, similar to *Address Resolution Protocol* (ARP) on Ethernet with IPv4.

Each IPv6 system joins the "solicited node" multicast group that corresponds to each of its addresses. Because the solicited node group address consists of the prefix **`ff02:0:0:0:0:1:ff00::/104`** followed by the bottom 24 bits of the address in question, addresses in different prefixes based on the same interface identifier (including the link-local address) all map to the same solicited node address.

Whenever a system needs to find out the link address for another system residing on the same link, it sends a neighbor solicitation to the solicited node address to which the IPv6 address of the remote system maps. The source host includes its own MAC address in the neighbor solicitation, so the neighbor knows where to send the reply.

Neighbor Unreachability Detection

RFC 2461[6] specifies a procedure for neighbor unreachability detection. IPv6 hosts and routers actively track whether their neighbors are reachable by periodically sending neighbor discovery messages directly to the neighbor. If the neighbor answers, it is reachable; if it does not, there must be some kind of problem, and the system discards the neighbor's MAC address and tries a regular multicast neighbor discovery procedure, allowing IPv6 systems to detect dead neighbors and neighbors that change their MAC address. But it is most useful to detect dead routers. On a subnet with more than one router, a host can simply install a default route toward another router when the router that it has been using becomes unreachable.

If a router loses its IPv6 address and no longer runs IPv6, Windows XP, Linux, MacOS, and FreeBSD all switch over to another router without incident. However, turning off the active router has much more severe effects: at the very least, ongoing downloads stall for a while, and in some cases, the session breaks. I have no explanation for this difference in behavior.

Stateless Address Autoconfiguration

Hosts and routers always configure link-local addresses on every interface on which IPv6 is enabled. The link-local address is nearly always derived from the interface MAC address, but to guarantee uniqueness, it is necessary to perform *Duplicate Address Detection,* which is discussed later.

When a host has a link-local address, it can obtain one or more global IPv6 addresses by using RFC 2462[7, 12], *Stateless Address Autoconfiguration*. IPv6 routers send out *Router Advertisement* (RA) packets (ICMPv6 type 134) periodically and in response to router solicitations. The information in RAs includes:

- An 8-bit *cur hop limit* field that tells hosts what value to use in the Hop Limit field of outgoing IPv6 packets
- The *managed address configuration* (M) flag—This flag is not well-defined, but the basic idea is that when it is set, hosts use a stateful mechanism (presumably *Dynamic Host Configuration Protocol Version 6* [DHCPv6]) to configure their addresses, and when the flag is not set, they use stateless address autoconfiguration.
- The *other stateful configuration* (O) flag—This flag is similar to the M flag, but indicates that the host should use a stateful mechanism to discover nonaddress configuration information.
- A 16-bit *router lifetime* value in seconds—This value tells hosts how long the default route that was created as the result of this RA should remain valid.
- The 32-bit *reachable time* value in milliseconds—This value indicates how long a neighbor should be considered reachable after receiving a "reachability confirmation," which is generally a neighbor advertisement but could be any packet.
- The 32-bit *retrans timer* value in milliseconds—The retrans timer tells hosts how long they should wait before retransmitting neighbor solicitation messages when there is no answer.

When fields that determine a value are set to zero, this means the value is not specified in the RA, so hosts must discover that value through other means. In addition to the preceding, router advertisements may also contain one or more options, such as:

- *Source link-layer address,* the router MAC address
- *MTU,* the maximum packet size that should be used on this subnet
- *Prefix information,* which specifies the prefixes used on the subnet and their properties

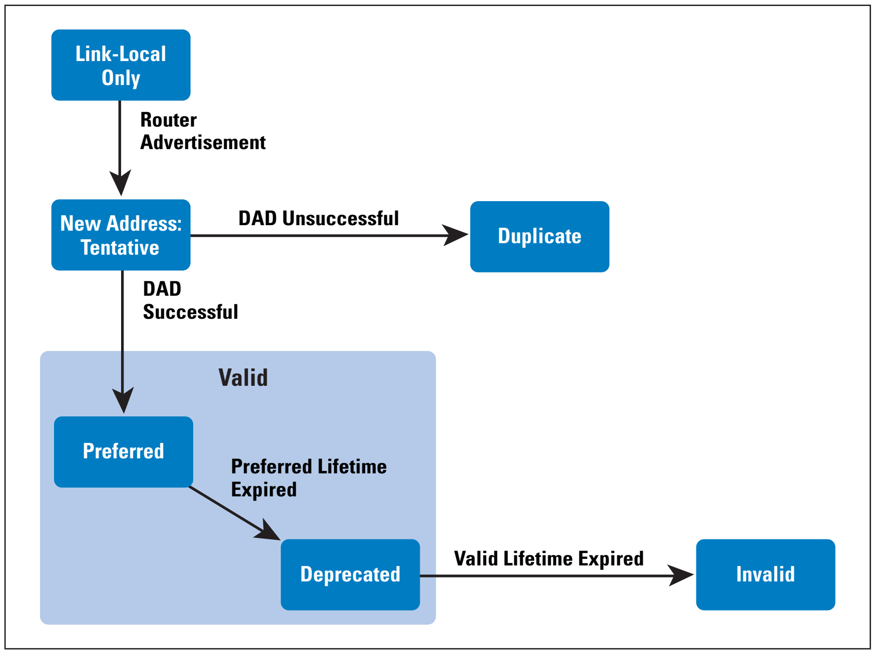
The prefix information option, in turn, has its own list of attributes:

- The *address prefix* itself and its length—For stateless address autoconfiguration to work, the prefix must be 64 bits long.
- The *on-link* flag—This flag tells hosts that the prefix is “on-link,” so systems with addresses within this prefix are reachable on the subnet in question without help from a router.
- The *autonomous address configuration* flag—This flag tells hosts that they can create an address for themselves by combining this prefix with an interface identifier.
- A 32-bit *valid lifetime* in seconds—This value indicates how long the prefix should be considered on-link and how long autoconfigured addresses using the prefix can be used.
- A 32-bit *preferred lifetime* in seconds—This flag tells hosts how long autoconfigured addresses using this prefix are preferred.

Duplicate Address Detection

To avoid the situation where two IPv6 systems use the same address, systems perform *Duplicate Address Detection* for (nearly) all new IPv6 addresses before they are used. Duplicate address detection is done for global unicast addresses—and not just for those created using stateless address autoconfiguration, but also for link-local addresses. For obvious reasons, there is no duplicate detection for anycast addresses, because the whole point of anycast is that multiple systems have the same address.

Figure 2: The Lifecycle of an IPv6 Address



As depicted in Figure 2, a host starts with only a link-local address. Duplicate address detection is also done for the link-local address, but this is not shown in the figure.

When a host receives a router advertisement that contains one or more prefixes with the autonomous address configuration flag set, the host creates addresses with interface identifiers derived from the IEEE *64-bit Extended Unique Identifier* (EUI-64) and possibly also a randomly generated one, if the host uses RFC 3041[8] address privacy. The host marks the resulting addresses as "tentative" and proceeds to execute the duplicate address detection procedure by joining the solicited node multicast group for the address in question and sending out one or more neighbor solicitation messages for the address. (If the number of duplicate address detection retries is configured to be zero, no duplicate detection is performed.) Only when there is no answer is the address used. If there is a conflict, the system is supposed to log the error and wait for manual intervention.

Address Lifetime

After successfully maneuvering past the duplicate address detection hurdle, addresses configured through stateless address autoconfiguration can be used until the "preferred lifetime" from the router advertisement message expires. In most cases, the lifetime does not expire because new RAs refresh the timers. But if there are no more RAs, eventually the preferred lifetime elapses and the address becomes "deprecated." New sessions should not use deprecated addresses but should choose "preferred" (nondeprecated) addresses, if available. However, existing sessions will continue to use the deprecated address. Eventually, the "valid lifetime" also runs out, and the deprecated address is removed from the interface, breaking any sessions that are still using the address.

Renumbering

Having different preferred and valid timers for the router advertisement itself and also for any prefixes contained in it makes it possible to do two things: renumber easily and cause more problems. It is even possible to do both at the same time. With stateless autoconfiguration, renumbering is easy: you simply give the router an address in the new prefix and set the preferred lifetime for the old prefix to zero, making hosts create one or more new addresses and deprecate any existing ones in the old prefix as soon as they receive the resulting router advertisement. After that, all new communication should start using the new address immediately. Existing TCP sessions and UDP associations continue to use the same address as before. After some time, all communication that started before the change should have stopped so that the old addresses can be removed safely.

This process is slightly more complex than it seems at first glance: as a precaution against attackers, hosts are not supposed to trust a valid lifetime of less than 7200. So make sure that the hosts have received at least one RA after setting the valid lifetime to 7200, and then set both the lifetimes to zero and remove the autonomous address configuration flag for the prefix. Two hours later, all hosts should have removed the addresses in this prefix, so you can remove the prefix from the router.

Beware that when you renumber because you are switching from one ISP to another, it is unavoidable that at some point, packets with source addresses in address space from ISP A end up at ISP B, or the other way around. If ISP B employs antispoofing or ingress filtering, it will not allow these packets through, so reduced connectivity will result. You can ask one ISP to remove the filters temporarily and then send out all your outgoing traffic over that ISP (or one that did not filter in the first place). However, do not expect too much cooperation from your ISP unless you are a valued customer.

Address Prefix and Router Lifetime Mismatch

Earlier, I mentioned the potential for causing more problems because router advertisements and the prefixes they contain have independent lifetimes. This scenario allows for four permutations:

- The RA lifetime is valid, and the prefix lifetime is valid: IPv6 works.
- The RA lifetime is invalid, and the prefix lifetime is invalid: IPv6 is disabled.
- The RA lifetime is valid, but the prefix lifetime is invalid: The system has an IPv6 default route but no global IPv6 address.
- The RA lifetime is invalid, but the prefix lifetime is valid: The system has a global IPv6 address but no IPv6 default route.

When a host has no global addresses but does have an IPv6 default route (case 3), it cannot reach the rest of the IPv6 Internet. Unfortunately, FreeBSD and MacOS hosts do not know that: they try anyway, with long delays as a result. Only after trying all the remote destination IPv6 addresses and timing out, the system falls back on IPv4 (for applications that try more than one address). Linux, on the other hand, does not install the IPv6 default route or ignores it when no global IPv6 addresses are present, so the timeout is immediate.

Windows XP does install the default route but magically manages to avoid lengthy timeouts anyway. On the other hand, Windows XP suffers timeouts when it has an IPv6 address but no default route (case 4) because Windows implements the on-link assumption: it first performs neighbor discovery on the local subnet for any IPv6 addresses. Only after neighbor discovery times out does Windows revert to IPv4. FreeBSD and MacOS, however, do not implement the on-link assumption, so they immediately notice that the IPv6 destination address is unreachable and revert to IPv4—if an IPv4 address is available and the application cycles through all addresses. With Linux, the default route does not seem to expire even though the timers eventually reach zero and lower. But addresses do expire and are removed when the lifetime for the associated prefix times out.

Address Selection

Choice is good, but it comes with problems of its own. The explicit support for multiple addresses in IPv6 requires the system or applications to choose which address to use for a given communication session. The coexistence of IPv4 and IPv6 in the same host makes this situation even more pressing. RFC 3484[9] provides guidelines in this area—it lists no fewer than 10 rules for choosing a destination address and 8 rules for selecting a source address. Most of these rules are fairly obvious, such as preferring a nondeprecated address over a deprecated one and not using a link-local source address to communicate with a destination that has a global address. It gets more interesting with the "policy table." On systems that support this mechanism, such as Windows XP and FreeBSD 5.4, the administrator can instruct the system to prefer certain address ranges over others.

Path MTU Discovery and Fragmentation

Because routers cannot fragment IPv6 packets, *Path MTU Discovery* (PMTUD) is mandatory in all cases where links with MTUs larger than 1280 bytes are used for IPv6, so it is imperative that routers generate ICMPv6 packet-too-big messages and that these messages make it back to the source of the offending packet. Filtering out these ICMPv6 messages makes it impossible to communicate reliably.

If you decide that you must filter ICMPv6 packet-too-big messages, you *must* use an MTU equal to the IPv6 mandatory minimum of 1280 bytes across your network so there is no need for PMTUD.

Upon reception of a packet-too-big message, TCP reduces its packet size to accommodate the smaller MTU on the path in question. However, protocols that run over UDP often cannot arbitrarily reduce their packet size. In IPv4, UDP packets are generally sent without the "don't fragment" bit set, so routers fragment them if necessary. In IPv6, this setup is not possible; if the packet is too large, the source host has to fragment it. The source host does this by first splitting the packet into unfragmentable and fragmentable parts. The IPv6 header and any headers that must be processed by routers along the way make up the unfragmentable part; the payload data and any headers that have to be processed only on the destination host are the fragmentable part. The fragmentable part is then split into as many parts as required to fit in the path MTU, and each part is transmitted as a packet containing the unfragmentable part, a fragment header, and one of the fragments of the fragmentable part.

The fragment header is 8 bytes, and except for a "next header" field and two reserved fields, it contains the same fragment offset, more fragments, and identification fields as the IPv4 header. The identification field is now 32 bits long and is used to indicate which fragments belong to the same original packet. All fragments except the last one have the "more-fragments" bit set and are multiples of 8 bytes.

After receiving the first fragment (which is not necessarily the first fragment of the original packet), a host waits up to 60 seconds for all other fragments to come in and, if they do, reassembles the original packet by combining all the fragments with the same source and destination addresses and identification field into a single packet. If one or more fragments is lost, the packet cannot be reassembled, so the entire packet is lost.

Note that IPv6 fragmentation has the same problem as IPv4 fragmentation: the TCP or UDP port numbers are available only in the first fragment, making it hard for firewalls and the like to filter fragmented packets. Common solutions are to reassemble the packet prior to filtering or to discard all fragments.

DHCPv6

DHCPv6 (RFC 3315[10]) is the IPv6 version of the DHCP. Because IPv6 has stateless address autoconfiguration, DHCP occupies a very different part of the landscape in IPv6 compared to IPv4. Although the details are different in the by-now-expected places (address length, use of multicasts, some streamlining), the DHCPv6 protocol itself is quite similar to the IPv4 version of DHCP. The more important differences are the way in which the protocol is used. DHCPv6 has three purposes:

- *Address configuration:* Giving out addresses to individual hosts
- *Nonaddress configuration:* Giving out other configuration information, such as DNS resolver addresses and domain search lists
- *Prefix delegation:* Giving out entire prefixes to routers (RFC 3633[11])

A DHCPv6 client interested in an address or other configuration information sends out a *solicit* message indicating its needs to the link-local scope multicast address **`ff02::1:2`**, port 547. (Server-to-client messages are addressed to port 546.) DHCPv6 servers that receive the *solicit* message either directly or forwarded by a relay and can accommodate the request respond with an *advertise* message. The client considers the offers in the various *advertise* messages and directs a *request* message to the server of its choice. The server then replies with a *reply* message, confirming the address or configuration information. Alternatively, if the client wants to receive only configuration information and no addresses or prefixes, it can send a *request-information* message, and the server immediately sends back a reply message, so only half the messages are exchanged and the whole process completes much faster. The client can also use the "rapid commit" option to indicate that it wants to use the expedited procedure for address or prefix assignment if it is fairly certain that it will take up the offer from the first DHCPv6 server that responds.

As expected, IPv6 addresses assigned with DHCPv6 come with a preferred and a valid lifetime. Sometime before this timer expires, the client sends a *renew* message, asking the server if it can continue to use the address. When it has no more use for the address, the client sends a *release* message. Less common situations have other messages.

To allow servers to recognize clients, each device that implements DHCPv6 has *DHCP Unique Identifier* (DUID). In IPv4, DHCP clients use a MAC address or user-supplied string as a Client Identifier. In DHCPv6 this client identifier is always the DUID. Devices can create their DUID in various ways, as long as the DUID is unique and not subject to change, if at all possible.

DHCPv6 supports an authentication mechanism that allows clients and servers to interact in a secure way, so third parties cannot inject false DHCP messages or modify legitimate ones. However, this mechanism must be preconfigured manually on all servers and clients, partially negating the advantages of DHCP over manual configuration.

An interesting use of DHCPv6 is *Prefix Delegation* (PD). With DHCPv6 PD, routers request a prefix that they then use to number one or more of their interfaces, supporting stateless address autoconfiguration for hosts connected to that interface. By creatively borrowing the DHCP timers and reusing them in router advertisements, a whole site can be renumbered by changing a single setting in a DHCPv6 configuration on a DHCPv6 server or a router functioning as a DHCPv6 PD server.

Ed.: This article is adapted from chapter 8 of *Running IPv6* by Iljitsch van Beijnum, published by Apress in 2005, ISBN 1590595270. The article differs from the chapter in that it has been edited for size and the vendor-specific examples have been removed. Used with permission. For information about the book, see:
`http://www.apress.com/book/bookDisplay.html?bID=10026`

References

[1] Karrenberg D., Ross G., Wilson P., and Nobile L., "Development of the Regional Internet Registry System," *The Internet Protocol Journal,* Volume 4, No. 4, December 2001.

[2] Carpenter, B., Fink, B., and Moore, K., "Connecting IPv6 Routing Domains Over the IPv4 Internet," *The Internet Protocol Journal,* Volume 3, No. 1, March 2000.

[3] Nichols, K., Blake, S., Baker, F., and Black, D., "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," RFC 2474, December 1998.

[4] Postel, J., “Internet Protocol,” RFC 791, September 1981.

[5] Deering, S. and Hinden, R., “Internet Protocol, Version 6 (IPv6) Specification,” RFC 2460, December 1998.

[6] Narten, T., Nordmark, E., and Simpson, W., “Neighbor Discovery for IP Version 6 (IPv6),” RFC 2461, December 1998.

[7] Narten, T. and Thomson, S., “IPv6 Stateless Address Autoconfiguration,” RFC 2462, December 1998.

[8] Narten, T. and Draves, R., “Privacy Extensions for Stateless Address Autoconfiguration in IPv6,” RFC 3041, January 2001.

[9] Draves, R., “Default Address Selection for Internet Protocol Version 6 (IPv6),” RFC 3484, February 2003.

[10] Droms, R., Ed., Bound, J., Volz, B., Lemon, T., Perkins, C., and Carney, M., “Dynamic Host Configuration Protocol for IPv6 (DHCPv6),” RFC 3315, July 2003.

[11] Troan, O. and Droms, R., “IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) Version 6,” RFC 3633, December 2003.

[12] François Donzé, “IPv6 Address Autoconfiguration,” *The Internet Protocol Journal,* Volume 7, No. 2, June 2004.

ILJITSCH VAN BEIJNUM holds a Bachelor of Information and Communication Technology degree from the Haagse Hogeschool in The Hague, Netherlands. In 1995, he found himself in the emerging Internet Service Provider business. There he learned about system administration, IP networking, and especially routing. After first starting a small ISP with four others and working as a senior network engineer for UUNET Netherlands, he became a freelance consultant in 2000. Not long after that, he started contributing to the IETF Multihoming in IPv6 working group. He wrote the book *BGP: Building Reliable Networks with the Border Gateway Protocol,* ISBN 0-596-00254-8, published by O’Reilly in 2002, and *Running IPv6,* ISBN 1590595270, published by Apress in 2005. E-mail: **`iljitsch@muada.com`**

Book Reviews

Electronic Brains

Electronic Brains, Stories from the Dawn of the Computer Age, by Mike Hally, ISBN 0-309-09630-8, Joseph Henry Press, 2005.

Electronic Brains is a personal account from the early days of computing that describes the childhood of a technology that is little more than 50 years old. The book originated as a BBC radio programme, still accessible at **`http://www.bbc.co.uk/radio4/science/electronicbrains.shtml`**. Mike Hally traveled over the globe looking for the first "computors" and the stories from the dawn of a new age. This book contains the results of the investigation, giving a first-hand testimony of hard work, passion, and amazing developments that shaped the second half of the last century.

Organization

Chapter 1, "From ABC to ENIAC," presents the development of what is commonly accepted as the first computer, the ENIAC, a computer that replaced calculating machines and people making the operations in ballistic trajectories analysis by hand. Credit is given to John Atanassof and Clifford Berry, the developers of ABC, possibly the first operational computer in the world.

Development of the UNIVAC, the computer famed by predicting the result of the 1952 U.S. presidential election, is presented in Chapter 2. Designed by Eckert and Mauchly, the developers of ENIAC, UNIVACs were commercial computers used for processing census data and so well marketed that the term "UNIVAC" was used as a synonym for "computer."

Chapter 3 looks at the development of the Rand 409, maybe the first mass-produced computer. The 409 was a medium-sized computer, with a price tag of US$100,000 that compared favorably against UNIVAC's $1 million, achieving a sell rate of one per week.

"Computing in Great Britain" is the focus of Chapter 4, where credit is given to Maurice Wilkes and Alan Turing. A worthy detail that gives a glimpse of the technical difficulties overcome is the description of memory based on mercury delay-lines, where binary data was stored using sound pulses on tubes filled with mercury engineered in such way that the delay from transmitter to receiver allowed the electronics to do the calculation before the data in memory was needed at the receiver side.

Perhaps the strangest computer development is set forth in Chapter 5. The *Lyons Electronics Office* (LEO) was a computer developed by a large catering company to expedite its clerical operations. LEO was possibly the first commercial computer in the world, so successful that the catering company began to produce and sell it to other corporations.

Chapter 6 describes the efforts by USSR scientists to develop computing technology. More than one development was made; it is not clear which was the first soviet computer, and the developments were secret—in some cases very specialized, such as a computer with ternary logic instead of the currently used binary logic (ENIAC used decimal logic).

Chapter 7 focuses on computing developments in Australia, work that did not last because the funds were scarce and sometimes the budget was assigned to other sciences, such as radiophysics. Here we can see that computers were used for purposes totally different than their uses in cold-war countries; for example, they were used to answer crossword puzzles—strange if we consider that the disk had a capacity of 3 KB.

A strange computer, formally known as *Hydraulic Economics Computer,* is described in Chapter 8. It was not a typical computer—it was a system developed to show the interrelation between macroeconomic variables using colored water, pumps, and valves. Universities, central banks, and Ford bought the computer, and four of them survive in different parts of the world. The emergence of IBM is the subject of Chapter 9, which presents IBM as a late adopter of computing technology that eventually became the leader of the computer age. We learn that the first computer produced by IBM was the IBM 701; after that came the IBM 1401 and then the IBM 360—the system that consolidated IBM as the ruler in the computing world.

Summary

From the ABC to the well-known ENIAC and UNIVAC, *Electronic Brains* is a testimony to the people who worked day and night to accomplish something that few others understood. Motivated mainly by passion and with little to no economic support, team spirit is a common factor in all the computer developments: "...it was like a brotherhood! We would help each other in case someone got stuck on a particular activity. I would have gone anywhere with those guys. I've never had such unified job environment. We knew we were pushing back the frontiers."

Electronic Brains is an enjoyable book that I recommend to any person with interest in computers and technology. Computer historians could scoff at the rather simple analysis of technical details, but this is not a technical book. The value of *Electronic Brains* is the first-hand account of early undertakings and the multiple-country investigation that is presented. With many anecdotes, this book will serve as a witness to the pioneers of a new era, the computing era.

—Claudio Gutiérrez
claudio.gutierrez.m@gmail.com