;login:
THE MAGAZINE OF USENIX & SAGE
April 2003 • volume 28 • number 2

inside:

USENIX & SAGE
The Advanced Computing Systems Association &
The System Administrators Guild

IPv6 configuration on Solaris 9 and freeBSD-4.x

by Timo Sivonen

Timo Sivonen is a principal consultant working for Greenwich Technology Partners. His professional activities circulate around operating systems, internetworking, and information security.

tljs@greenwichtech.com

The basic setup to enable IPv6 on the Solaris 9 and FreeBSD-4.x operating systems is discussed. The text will begin with a quick overview of the IPv6 packet format and various addressing types, before continuing to stateless address autoconfiguration for a host and a router. The text will also deal with the current and transition issues of the Domain Name Service for IPv6. Finally, there is discussion about IPsec and how to use IPsec for inbound authentication and to secure communication between two hosts.

Introduction

"Is there a customer requirement?" and "Have you been asked about migration?" were the first comments I received when I said that I had started to work on IPv6.[1] Undeterred by reality, my incentive to play with IPv6 was to experiment with the protocol to see how it works. Although the buzzwords "next generation Internet" and "migration" have been flying around for the past eight years, no one really knows whether this will ever happen. Instead, I decided to take a fresh look at IPv6 and view it as a completely new network protocol that just happens to improve IP further.

IPv6's availability has improved in the past few years. It is no longer a network protocol for only researchers to play with, but the emergence of IPv6 stacks, commercial and open source, means that IPv6 is getting serious. If IPv6 is in the operating system already, enabling it will cost nothing. IPv6 is a turnkey protocol as SPX/IPX was, and you don't have to request an IP address.

In the following text we will discuss how to set up IPv6 on Solaris 9 and FreeBSD-4.x. We will briefly cover the various addressing types of IPv6 and what they mean to an average user who wants to get his/her feet wet. A quick overview of the two recommendations for IPv6 DNS will be presented. Finally, we will discuss the use of IPSec, try a very basic setup, and test interoperability between Solaris and FreeBSD.

Getting Started — Header Format

No discussion of a networking protocol would be complete without displaying the actual packet header (Figure 1), as in RFC-2460.[1] The IPv6 header is 40 octets long, as opposed to the 20 octets of the IPv4 header. The IPv6 header is much simpler, which will speed up its processing on routers and end-nodes.

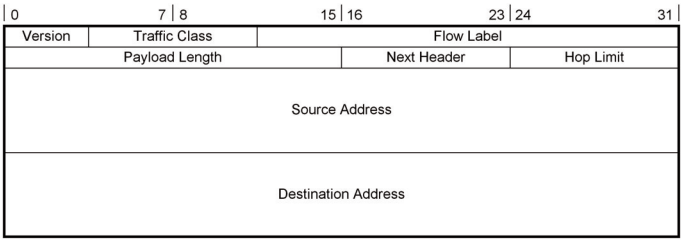

Figure 1. IPv6 Header

The fields of the IPv6 header are as follows:

Field	Bits	Description
Version	4	Internet Protocol header version. Has value of "6".
Traffic Class	8	Used by routers to recognize different traffic classes or priorities.
Flow Label	20	Requests special handling of the IPv6 packet, e.g., real-time quality of service.
Payload Length	16	Length of the payload, i.e., packet excluding IPv6, in octets.
Next Header	8	Type of header following IPv6, as in RFC-1700.[2]
Hop Limit	8	Time-to-live value.
Source Address	128	Sender's address.
Destination Address	128	Recipient's address.

The IPv6 header does not have a header checksum field. Although this seemed important in the days of IPv4, error checking in the link layer protocols has made this field obsolete. The removal of header checksum leaves the possibility that the IPv6 header could get corrupted between the network interface and the network protocol. The probability for this to happen is minimal and computers are far more reliable nowadays than they were in 1981, when IPv4 was designed.

IPv6 also has built-in provisions for Quality of Service (i.e., Flow Label, Traffic Class), although these will not be discussed in this text. QoS is a major improvement over IPv4, and IPv6 should be able to prove its worth as a voice over IP and video over IP delivery vehicle. The major advantage of IPv6 QoS is its

architectural independence of the underlying link layer. Not surprisingly, online gamers have also shown some interest in IPv6.

Getting Started — Addressing

The first thing you need is an address, and IPv6 has 2^{128} of those, compared to 2^{32} of IPv4. In this section we will deal with the basics of IPv6 addresses and address formats. The logic of IP addressing has changed quite a bit from the days of Classes A, B, and C. As with IPv4, each address type is defined by its prefix and RFC-2373[3] lists the following prefixes:

Allocation	Prefix (binary)	Fraction of Address Space
Reserved	0000 0000	1/256
Unassigned	0000 0001	1/256
Reserved for NSAP Allocation	0000 001	1/128
Reserved for IPX Allocation	0000 010	1/128
Unassigned	0000 011	1/128
Unassigned	0000 1	1/32
Unassigned	0001	1/16
Aggregatable Global Unicast Addresses	001	1/8
Unassigned	010	1/8
Unassigned	011	1/8
Unassigned	100	1/8
Unassigned	101	1/8
Unassigned	110	1/8
Unassigned	1110	1/16
Unassigned	1111 0	1/32
Unassigned	1111 10	1/64
Unassigned	1111 110	1/128
Unassigned	1111 1110 0	1/512
Link-local Unicast Addresses	1111 1110 10	1/1024
Site-local Unicast Addresses	1111 1110 11	1/1024

Table 1. IPv6 Address Assignments and Prefix

Since working with binary prefixes is usually confusing and sometimes error-prone, you may want to consult the IPv6 Address Oracle[4] at the Advanced Network Management Laboratory at Indiana University. While I was trying to learn the address structure, I was almost continuously visiting the site.

Note that the dotted-decimal notation of IPv4 addresses has been replaced with hexadecimal. Practically speaking, an IPv4 address is 32 bits, i.e., 4 octets long, while an IPv6 address is 128 bits, i.e., 16 octets long. Hexadecimal representation is more economical for large addresses, and one can use "::" as an abbreviation for a string of zeros. However, this can be used only once in an address. In other words, one can write an IPv6 address as

```
fe80::280:c7ff:0:434
```

or equally

```
fe80:0:0:0:280:c7ff:0:434d
```

but fe80::280:c7ff::434d would be ambiguous and thus illegal.

It is possible to use the IPv4 dotted-decimal notation under certain circumstances. If there is an embedded IPv4 address inside of the IPv6 address, the last four octets are represented using the dotted-decimal notation:

```
::ffff:192.168.1.1
```

Currently IPv6 defines the following address types:

- Unicast. A unicast address identifies a single interface. There are several unicast address types such as unspecified address, loopback address, global aggregateable unicast address and local-use addresses. There are other types which are not in the scope of this discussion.
- Anycast is a special case of unicast. An anycast address identifies a set of interfaces, typically belonging to different nodes. A packet sent to an anycast address will be delivered to the nearest interface. Anycast is not in the scope of this discussion.
- Multicast. There is no broadcast in IPv6 since multicast can be used for the same purpose. A multicast address is identified by the prefix ff/8.

Unicast Addresses

A unicast address consists of the subnet prefix and the interface identifier (Figure 2). Although the interface identifier depends on the type of the unicast address and the underlying media (e.g., Ethernet LAN, point-to-point circuit or a tunnel endpoint), it must be 64 octets for all addresses in Table 1.

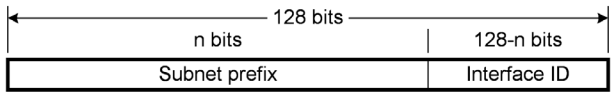

Figure 2. Unicast Address

In the following text we will discuss loopback addresses, local-use addresses and aggregateable global unicast addresses. Other address types such as IPv6 addresses with embedded IPv4 addresses, NSAP addresses, and IPX addresses are covered in RFC-2373 and Huitema.[5]

Interface Identifier

Finding the interface identifier for LAN-connected nodes is relatively straightforward, since the 48-bit MAC address is globally unique. The procedure to expand the MAC address to the 64-bit interface identifier is based on EUI-64.[6] This is accom-

plished by inserting two octets with hexadecimal values of 0xff and 0xfe in the middle of the address and setting the universal/local bit to 1. This is displayed in Figure 3:

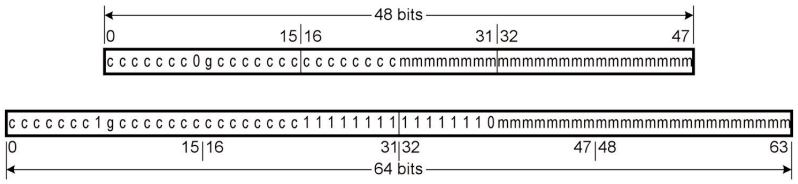

Figure 3. Expansion of a 48-bit MAC Address into an EUI-64 Interface Identifier

The g bit denotes the local/global bit and it is set to zero for 48-bit MAC-based identifiers. The c and m bits are for the company ID and manufacturer-selected extension identifier, respectively. Hence, we will get the following kind of expansions:

0:3:ba:6:14:66 → 203:baff:fe06:1466
0:80:c7:54:43:4d → 280:c7ff:fe54:434d

IPv6 does not have the Address Resolution Protocol[7] to convert IPv6 addresses to MAC addresses. Instead, this function has been transferred to the ICMPv6 Neighbor Discovery protocol, which provides generic mechanisms to discover the MAC addresses of the other hosts. Neighbor Solicitation is either multicast or unicast, and Neighbor Advertisement (the reply) is always unicast.

One may well use the same interface identifier on multiple interfaces as long as the subnet prefix is different. RFC-2373 states the following:

> Note that the use of the same interface identifier on multiple interfaces of a single node does not affect the interface identifier's global uniqueness or each IPv6 address's global uniqueness created using that interface identifier.

For example, Sun hardware assigns the same MAC address to all Ethernet interfaces. This is perfectly legal as long as the interface identifier is unique to each link (e.g., Ethernet segment) and the complete IPv6 address (64 bits of the subnet prefix and 64 bits of the interface identifier) is globally unique.

UNSPECIFIED ADDRESS

The first unicast address type you will encounter is the unspecified address:

0:0:0:0:0:0:0:0

or, in the compressed form:

::

The unspecified address is only used when an interface does not have a unicast address assigned to it yet. It can never be assigned to a node or used as a destination address.

LOOPBACK ADDRESS

The loopback address is the equivalent of 127.0.0.1 and cannot be used on a physical interface. The IPv6 loopback address consists of 127 binary zeros ending with binary one:

0:0:0:0:0:0:0:1

or in the compressed form:

::1

As with IPv4, the loopback address is always associated within a single node. You cannot use it as the source or the destination addresses for packets you are sending over a link to another node.

LINK-LOCAL ADDRESS

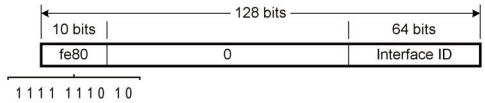

Figure 4. Link-local Address

You can use this address type (Figure 4) on a subnet (e.g., Ethernet segment) the host is connected to. The link-local address is configured automatically for each IPv6-capable interface. Since there is only the prefix, fe80::/10, but no subnet identifier, these addresses cannot be routed.

The link-local address space is roughly equivalent to the IPv4 169.254.0.0/16 address space[8] that DHCP auto-configured hosts are expected to use if no DHCP server is present. It is pointless to attempt to route link-local addresses, since there will be other link-local addresses on the destination network and the destination host will have no idea which is the correct host.

On a Solaris system, your interface list will look like the following (after the interfaces have been configured for IPv6 but the local router has not handed out the subnet prefix yet. . . or there may not be a local router at all):

```
# ifconfig -a6
lo0: flags=2000849<UP,LOOPBACK,RUNNING,
                MULTICAST,IPv6> mtu 8252 index 1
      inet6 ::1/128
eri0: flags=2000841<UP,RUNNING,MULTICAST,IPv6>
                mtu 1500 index 2
      ether 0:3:ba:e:6a:6a
      inet6 fe80::203:baff:fe0e:6a6a/10
```

The bottom line is that link-local addresses are instant: You don't have to ask for an IPv6 address to be able to communicate with the other IPv6 hosts on the same segment. You only need to hook up to the network and the other hosts are immediately reachable, without any waiting periods for a DHCP server that isn't there. Finally, since MAC addresses are globally unique, there can be up to 2^{64} nodes on a single IPv6 LAN segment.

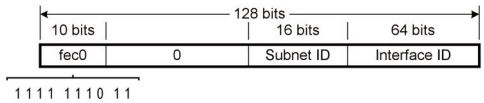

Figure 5. Site-local Address

SITE-LOCAL ADDRESS

The site-local unicast address (Figure 5) is the IPv6 equivalent of the 10.0.0.0/8-network.[9] Site-local address is for organizations that want to set up private routed IPv6 internetworks without Internet connectivity.

The site-local address prefix is fec0::/10. There are only 16 bits for the subnet identifier, which is equal to the two middle octets of the 10.0.0.0/8 network, i.e., 2^{16} subnets. Since the subnet and the interface identifier are separate entities, it is not possible to increase the number of subnets at the expense of hosts on each subnet. The interface identifier is a 64-bit fixed-length field that cannot be shortened. On the other hand, with IPv6 one does not have to put any all-zeros and all-ones addresses aside for network numbers and broadcast addresses, respectively, since each subnet is denoted by its own subnet identifier and there is no broadcast in IPv6.

Today, it seems that a high fraction of enterprise networks are using RFC-1918[9] address space and resort to Network Address Translation (NAT) to communicate with the rest of the world. To make things worse, many of them confuse NAT with network security even though the two main goals of NAT are:

- To extend the registered official address space one has received from the ISP, e.g., five official addresses on the outside and a mixture of private networks on the inside.
- To get at least some independence from one's Internet Service Provider. It is difficult enough to get official IPv4 network numbers, but renumbering all the IPv4 networks and hosts is a nightmare if the ISP changes.

It remains to be seen whether the IPv6 address space is large enough to convince enterprise network administrators to put site-local addresses aside and use registered (i.e., aggregateable global unicast) IPv6 addresses instead. Moreover, IPv6 networks are much easier to renumber since all nodes get the network prefix(es) automatically from the nearest router; there can be multiple prefixes on an interface simultaneously, and if things start to go wrong, one can always get to a host via a link-local address. If these arguments are not enough, there will be a market for IPv6 network address translation.

As your IPv6 interface starts up, it will receive its network prefix from its default gateway:

```
# ifconfig -a6
lo0: flags=2000849<UP,LOOPBACK,RUNNING,
                   MULTICAST,IPv6> mtu 8252 index 1
     inet6 ::1/128
eri0: flags=2000841<UP,RUNNING,MULTICAST,IPv6>
                   mtu 1500 index 2
     ether 0:3:ba:e:6a:6a
     inet6 fe80::203:baff:fe0e:6a6a/10
eri0:1: flags=2080841<UP,RUNNING,MULTICAST,
                   ADDRCONF,IPv6> mtu 1500 index 2
     inet6 fec0::a801:203:baff:fe0e:6a6a/64
```

Depending on your IPv6 implementation, additional IPv6 addresses may show up as subinterfaces (Solaris) or aliases (KAME[10] stack on BSD). This is merely a matter of representation, since the two forms share the same functionality.

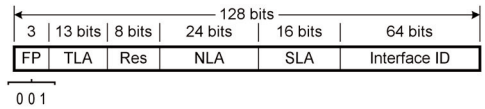

Figure 6. Aggregatable Global Unicast Address

AGGREGATABLE GLOBAL UNICAST ADDRESS

You need the aggregateable global unicast address (Figure 6) to run IPv6 on the Internet, that is, if your ISP is connected to 6bone (*http://www.6bone.net/*) and they have registered a prefix for themselves. The address type is specified in RFC-2374.[11]

The Format Prefix (FP) is always binary 001 for this type of an address. Since the length of FP is three bits and the Top-Level Aggregation Identifier (TLA) is 13 bits, the TLA is actually 16 bits, with the FP restricting the values TLA may have. Hence, the possible values for the TLAs are 2000::/16 through 3fff::/16.

The Top-Level Aggregation Identifier is the highest level in the routing hierarchy. A large ISP could have a TLA, but currently there is no TLA for a single organization. Instead, three types of TLAs have been allocated:

- Regional Internet Registry allocated sub-TLAs (prefix 2001::/16) a.k.a. Production sub-TLAs.
- 6to4 TLAs (prefix 2002::/16) to route IPv6 traffic over the IPv4 Internet.
- 6bone pseudo-TLAs (pTLAs, prefix 3ffe::/16).

The Reserved field is for future use and is always zero. However, in the following section we will discuss the previously men-

tioned TLA types more closely, and all of them use the address space put aside in the Reserved field.

The Next-Level Aggregation Identifier is used by organizations that have received a TLA to create an addressing hierarchy. Instead of using the space of 24 bits as a flat index, RFC-2374 recommends putting aside a number of bits as a selector and reserving the remaining space for each site the organization has to support (Figure 7).

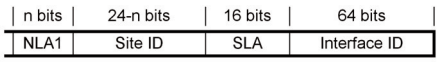

Figure 7. NLA and Site Identifiers

The organization may also choose to use the site identifier space to create an NLA hierarchy and manage routing tables more efficiently (Figure 8). This is not mandated in RFC-2374, but the recipients of TLAs are advised to set up their addressing in a hierarchical manner.

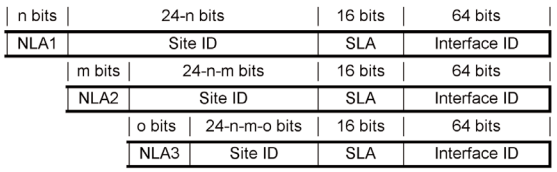

Figure 8. NLA Hierarchy

The bottom line is that 24 bits of NLA is a lot of real estate for anybody. The designers of IPv4 did not want to use their 24 bits of network space as a flat index but created the structure of Class A, B, and C networks – and added Classless Inter-Domain Routing later – to be able to control address allocation. Similarly, it does make sense to set up an NLA hierarchy since otherwise one might end up with 2^{24} entries in the routing table.

The Site-Level Aggregation Identifier is the registered equivalent of the subnet identifier in the site-local address. It can accommodate a total of 2^{16}, or 65,536, networks with up to 2^{64} nodes on each. However, it is the responsibility of each administering organization to maintain its address space reasonably well, and the current state-of-the-art is to set up an addressing hierarchy (Figure 9).

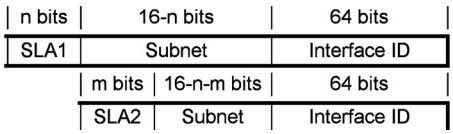

Figure 9. SLA Hierarchy

There is an interesting consequence of using 16 bits both for the SLA and the subnet identifier. Even if the organization decides to start with site-local addresses and upgrade to aggregateable global unicast addresses later, there is an upgrade path. Both site-local addresses and the various types of aggregateable global unicast addresses have 16 bits assigned for the local network identifier. Hence, one will not have to change the local addressing structure to get or change an ISP, since this information remains untouched in the process.

While your host auto-configures its IPv6 interfaces, you will see new entries in the interface list. Note how it is perfectly legal to use the same network value (0xa801) both in the subnet identifier and SLA for the interfaces eri0:1 and eri0:2, respectively. Interface renumbering has never been this easy.

```
# ifconfig -a6
lo0: flags=2000849<UP,LOOPBACK,RUNNING,
                MULTICAST,IPv6> mtu 8252 index 1
     inet6 ::1/128
eri0: flags=2000841<UP,RUNNING,MULTICAST,IPv6>
                mtu 1500 index 2
     ether 0:3:ba:e:6a:6a
     inet6 fe80::203:baff:fe0e:6a6a/10
eri0:1: flags=2080841<UP,RUNNING,MULTICAST,
                ADDRCONF,IPv6> mtu 1500 index 2
     inet6 fec0::a801:203:baff:fe0e:6a6a/64
eri0:2: flags=2080841<UP,RUNNING,MULTICAST,
                ADDRCONF,IPv6> mtu 1500 index 2
     inet6 2001:11f8:5ef9:a801:203:baff:fe0e:6a6a/64
```

OTHER AGGREGATABLE GLOBAL UNICAST ADDRESSES

In the following sections we will discuss other aggregateable global unicast address types. It would have been too easy to have only one address structure and, therefore, IETF has invented three of them, depending on the address prefix one is using.

TLA 1

Production sub-TLAs (prefix 2001::/16) are meant to be used on native IPv6 infrastructure. This is also known as the TLA 1, and the initial prefix assignments can be found in RFC 2928.[12] To make things more complicated, the TLA 1 addresses have a structure that differs from the one discussed before. (Figure 10)

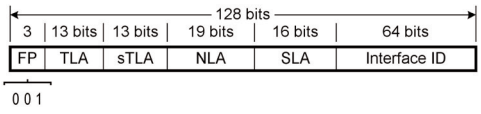

Figure 10. TLA 1 Address

All sub-TLA addresses share the same TLA prefix, 2001::/16, but the sTLA portion of the address is allocated by the Regional Internet Registries, i.e., APNIC, ARIN, and RIPE. As of September 11, 2002, the RIRs had allocated 219 sTLA prefixes.

The length of the sTLA is 13 bits, i.e., the total prefix length FP + TLA + sTLA should be 29 bits. However, the Global IP Allocation List[13] at RIPE shows prefix lengths of 32 and 35 bits:

WIDE-JP-19990813	2001:0200::/35
EU-UUNET-19990810	2001:0600::/35
UK-JANET-19991019	2001:0630::/32

Obviously, this extends the sub-TLA to the Next-Level Aggregation Identifier and leaves 16 bits or 13 bits (/32 and /35 prefixes, respectively) for the NLA. On the other hand, even with 13 bits for the NLA and 16 bits for the Site-Level Aggregation (SLA) Identifier one can design quite a respectable network.

6to4 TLA

6to4 TLAs (prefix 2002::/16) are usedto route IPv6 traffic over the IPv4 Internet and even over IPv4 dialups, if necessary. This TLA and the address format are discussed in Carpenter et al.[14] and Carpenter and Moore.[15] The address format is shown in Figure 11.

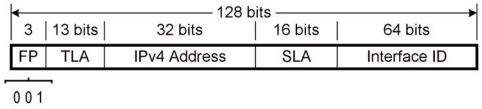

Figure 11. 6to4 Address

6to4 is an alternative vehicle if you want to run IPv6 but don't have access to a 6bone gateway. In this case, your IPv6 address is constructed from your registered IPv4 address. Of course, if you are dialing in to an ISP or your cable modem/DSL provider assigns your address via DHCP, your IPv4 address won't be static and you cannot run any persistent services.

6bone TLA

6bone (prefix 3ffe::/16) is a global IPv6 testbed on the Internet. Originally 6bone links were IPv6-over-IPv4 tunnels, but the infrastructure is gradually moving toward native IPv6 links and routers. On the practical side of things, one is most likely to get a 6bone address if one wants to run IPv6 outside a single LAN or an organization. Since the ISPs and other organizations that have registered a TLA 1 address may not be able to deliver an IPv6 service to end customers for a long time, your most viable route to the IPv6 world is to use 6bone addresses and IPv6-over-IPv4 encapsulation.

The 6bone pTLA address is shown in Figure 12:

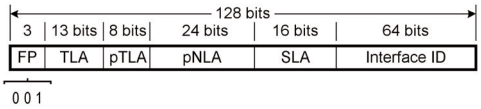

Figure 12. 6bone pTLA Address

As of September 3, 2002, there were 132 6bone prefixes allocated. Unlike RIR-allocated sub-TLAs, there were /24, /28, and /32 prefixes:

ROOT66/US-CA	3ffe:0000::/24
TRUMPET/AU	3ffe:8000::/28
TELEPAC/PT	3ffe:4000::/32

If you don't have any 6bone gateway you can access, probably the easiest way to get connected is to use the TSP client provided by Freenet6 (*http://www.freenet6.net/*).

Multicast Addresses

While multicast came to IPv4 only later, in the form of the D address class,[16] it has been a critical part of IPv6 from the very beginning. The major advantage that multicast has over broadcast is better control of propagation, since an application will not have to reach every station on the network but only those nodes that actually need the data. The format of the multicast address is in Figure 13:

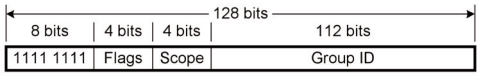

Figure 13. Multicast Address

The prefix for multicast is all-ones (i.e., binary 1111 1111 or ff::/8). The next four bits are the flags (Figure 14), but only the last bit is in use. The T bit must be 0 for permanently assigned multicast addresses (visit *http://www.iana.org/* for more information) and 1 for non-permanent (i.e., transient) addresses. The first three bits are reserved for future use and must be set to zero.

Figure 14. Multicast Flags

While IPv4 multicast has few propagation-controlling mechanisms other than time to live, IPv6 multicast has a clearly defined scope:

Value	Scope
0	Reserved
1	Node-local scope
2	Link-local scope
3	—
4	—
5	Site-local scope
6	—
7	—
8	Organization-local scope
9	—
a	—
b	—
c	—
d	—
e	Global scope
f	Reserved

Table 2. Multicast Scope Values

For example, the assigned Group Identifier for Routing Information Protocol Next Generation (RIPng) routing protocol is 9 (hex) and thus all RIPng updates are sent using the multicast address ff02::9, i.e., the scope is for the local link only. It is a bit difficult to imagine any practical use for route updates beyond the local link, but an application such as Network Time Protocol could well have several scope values:

ff01::101 means all NTP servers on the same node as the sender.
ff02::101 means all NTP servers on the same link (e.g., Ethernet LAN) as the sender.
ff05::101 means all NTP servers in the same site as the sender.
ff08::101 means all NTP servers in the same organization as the sender.
ff0e::101 means all NTP servers in the Internet.

Note that transient multicast addresses are relevant only within the given scope. For example, the transient, site-local address ff15::101 is a completely different entity from the transient, link-local address ff12::101 or the permanent address ff05::101. Transient addresses are disposed of at the end of their use.

Since multicast has a critical role in IPv6 and all IPv6 hosts and nodes must be multicast-capable, it is likely that the emergence of IPv6 will also enable large-scale deployment of multicast gaming and multicast audio and video over the Internet.

Auto-configuration

Address auto-configuration is one of the many areas where IPv6 makes sense to an end user. IPv6 defines two modes of auto-configuration: stateless and stateful. All IPv6 hosts are capable of stateless auto-configuration,[17] in which they generate their link-local address from the MAC addresses, possibly using some external source (the nearest router) to generate the site-local and aggregateable global unicast address. At the end of the auto-configuration process there is duplicate detection to verify uniqueness of the addresses and to prevent address collisions with the other hosts on the same link.

The most common example of stateful auto-configuration is a DHCP server. While stateful auto-configuration is perfect for setting up the addresses and the default route, it does not cover things like DNS resolution, path to the boot image, timeservers, and other information one might need for host configuration.

In practice, one may use both stateless and stateful auto-configuration to set up a host. Stateless auto-configuration is used for the initial configuration, and the rest of the parameters are obtained through a stateful process. Consequently, the auto-configuration server does not necessarily have to share the same link with the host, although this could be desirable for administrative reasons. In other words, if things start going wrong the administrator does not have to troubleshoot both the network and the server. Moreover, in large organizations these are two different groups that may not even want to talk to each other, and having a critical organization-wide resource behind multiple routers could become fertile ground for finger-pointing and other corporate trouble.

Solaris Host

The information is this section is based on the Solaris documentation[18] and Zilbauer.[19] You can set up Solaris 8 or Solaris 9 in just a few steps:

1. Find out the device name for your primary Ethernet interface. There are a few ways to do this but let us use ls:

```
# ls -l /etc/hostname*
-rw-r--r--   1 root   root      23 Jun 15 14:36 /etc/hostname.eri0
```

2. Create an empty file /etc/hostname6.interface and reboot:

```
# touch /etc/hostname6.eri0
# ls -l /etc/hostname*
-rw-r--r--   1 root   root      23 Jun 15 14:36 /etc/hostname.eri0
-rw-r--r--   1 root   other      0 Sep 12 13:37 /etc/hostname6.eri0
# reboot
```

   Rebooting is necessary due to the way Solaris initializes interfaces for IPv4 and IPv6.

3. After the system has come back up you will see IPv6 active on the loopback and the Ethernet interfaces:

```
# ifconfig -a6
lo0: flags=2000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv6> mtu 8252 index 1
    inet6 ::1/128
eri0: flags=2000841<UP,RUNNING,MULTICAST,IPv6> mtu 1500 index 2
    ether 0:3:ba:e:6a:6a
    inet6 fe80::203:baff:fe0e:6a6a/10
```

4. Edit /etc/inet/ipnodes[20] to add the addresses to the host table. For some reason Sun didn't want to continue to use /etc/hosts but created a completely new hosts file instead. Interestingly, the syntax of the file has not changed from /etc/hosts. This is a perfectly valid reason to start using DNS with IPv6 as soon as possible.

```
# vi /etc/inet/ipnodes
"/etc/inet/ipnodes" [Read only] 17 lines, 500 characters
:a
fe80::203:baff:fe0e:6a6a mysun.v6.mydomain.org mysun.v6 ll-mysun
.
:wq!
"/etc/inet/ipnodes" 18 lines, 522 characters
```

   Note how we used the .v6 subdomain for the IPv6 addresses. Bucklin and Sekiya[21] make an excellent argument for separating IPv6 addresses from the IPv4 ones. Most importantly, there are quite a few resolvers in use that do not understand the AAAA record and may fail when they get such a record back. On the other hand, since it is uncertain how IPv6 will be rolled out to organizations, keeping the IPv6 and IPv4 namespaces separate at this stage seems a practical solution.

   Naming conventions for IPv6 hosts can be a bit tricky. While IPv4 usually defines only one IP address for an interface, it is common to have at least two IPv6 addresses for an interface: one link-local and one routable. Hence, to distinguish between these two we will prefix the hostname with ll- (e.g., ll-mysun) for the link-local address while the fully qualified domain name points to the routable address. The interface name is another possible prefix.

5. We also have to edit /etc/nsswitch.conf to enable host lookups from DNS as well as /etc/inet/ipnodes:

```
# ex /etc/nsswitch.conf
"/etc/nsswitch.conf" 26 lines, 690 characters
:/^ipnodes
ipnodes:        files
:s/files/files dns/
ipnodes:        files dns
:wq!
"/etc/nsswitch.conf" 26 lines, 694 characters
```

6. As we discussed in the section on Link-local Address, you can immediately communicate with other IPv6 hosts on the local LAN. As always, pinging them is a good way to start, and if ping works, Telnet will probably too:

```
# ping -A inet6 fe80::2e0:98ff:fe83:48d0
fe80::2e0:98ff:fe83:48d0 is alive
```

Now you are all set. Auto-configuration is about the only reasonable way to set up an IPv6 host, since it guarantees consistent setup across all your hosts on the network. Of course, without a router there will not be anyone to hand out a site-local prefix or an aggregateable global unicast prefix, and you will have to use link-local addresses only.

FREEBSD HOST

FreeBSD and other BSD variants running the KAME stack are easy to configure for IPv6. The GENERIC kernel in FreeBSD-4.6 has IPv6 enabled by default, and you only have to add the following line to your /etc/rc.conf and reboot:

```
ipv6_enable="YES"
```

Your Ethernet interface is probably set up for IPv6 anyway, but this entry will guarantee that the host will also request the prefix from the nearest router and sets up its routing table accordingly.

1. Check that there is ipv6_enable entry in /etc/rc.conf and /etc/defaults/rc.conf:

```
# grep ipv6_enable /etc/rc.conf
# grep ipv6_enable /etc/defaults/rc.conf
ipv6_enable="NO"       # Set to YES to set up for IPv6.
#
```

   In this case the default setting is "NO" and we have to override it in /etc/rc.conf:

2. Add two entries, ipv6_enable and ipv6_network_interfaces, to the end of /etc/rc.conf:

```
# ex /etc/rc.conf
"/etc/rc.conf " 22 lines, 625 characters
:$
:a

# IPv6 configuration settings
ipv6_enable="YES"
ipv6_network_interfaces="auto"
.
:wq
"/etc/rc.conf " 24 lines, 668 characters
```

3. Reboot.

```
# shutdown -r now
```

   When the host boots up it will obtain the prefixes and routing information from the nearest router, if any. If no router is live, it will use its link-local address until a router becomes available.

4. Log in and run ifconfig(8) after the system has rebooted:

```
# ifconfig -a
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
    inet6 ::1 prefixlen 128
    inet6 fe80::1%lo0 prefixlen 64 scopeid 0x2
    inet 127.0.0.1 netmask 0xff000000
ed1: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
    inet6 fe80::2e0:98ff:fe83:48d0%ed1 prefixlen 64 scopeid 0x3
    inet6 fec0::a801:2e0:98ff:fe83:48d0 prefixlen 64 autoconf
    inet 192.168.35.51 netmask 0xffffff00 broadcast 192.168.35.255
    ether 00:e0:98:83:48:d0
    media: Ethernet 10baseT/UTP
    status: active
```

Note that the KAME stack has a funny habit of appending the interface name at the end of the link-local address. Hence, to ping the Solaris host we just configured we have to issue the following command (unlike Solaris, FreeBSD has ping and ping6 for IPv4 and IPv6, respectively):

```
# ping6 fe80::203:baff:fe0e:6a6a%ed1
PING6(56=40+8+8 bytes) fe80::2e0:98ff:fe83:48d0%ed1 --> fe80::203:baff:fe0e:6a6a%ed1
16 bytes from fe80::203:baff:fe0e:6a6a%ed1, icmp_seq=0 hlim=64 time=0.12 ms
16 bytes from fe80::203:baff:fe0e:6a6a%ed1, icmp_seq=1 hlim=64 time=0.121 ms
16 bytes from fe80::203:baff:fe0e:6a6a%ed1, icmp_seq=2 hlim=64 time=0.134 ms
^C
--- fe80::203:baff:fe0e:6a6a%ed1 ping6 statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/std-dev = 0.120/0.125/0.134/0.006 ms
```

The link-local address notation adopted by KAME does have its advantages when you have a multi-homed host, especially if you have a Sun box running some variant of BSD. Because all Sun Ethernet interfaces share the same MAC address, the link-local address becomes ambiguous – one does not know which interface to use to reach the destination. For example, the following link-local address could refer to any interface on a multi-homed Sun box:

```
fe80::a:a20:22ff:fe6d:6e
```

In contrast, the KAME notation explicitly points to the right interface:

```
fe80::a:a20:22ff:fe6d:6e%le0
fe80::a:a20:22ff:fe6d:6e%le1
```

Of course, the discussion on interface identifiers in the link-local address has meaning only on the host one has the KAME stack on. In other words, do not add the interface extension to the IPv6 address if you are pinging a FreeBSD host from Solaris:

```
# ping -A inet6 fe80::2e0:98ff:fe83:48d0%ed1          ← This will not work
ping: unknown host fe80::2e0:98ff:fe83:48d0%ed1

# ping -A inet6 fe80::2e0:98ff:fe83:48d0              ← This works!
fe80::2e0:98ff:fe83:48d0 is alive
```

SOLARIS ROUTER

If you want to connect your IPv6 LAN to the rest of the world you will need a router. Cisco, Nortel, and Juniper, to name a few vendors, all have IPv6 support in at least some of their products. However, we will choose the poor man's alternative and use a dual-homed Solaris box instead.

The basic configuration of a Solaris router does not differ from setting up a Solaris node. Since we have two Ethernet interfaces, we have to create a hostname6.*interface* file for them both.

1. Find out your interface names and create empty /etc/hostname6.interface for all physical devices. If both interfaces are on the motherboard (as is the case with Netra X1) or you have a quad Ethernet card, figuring out the second interface is simple.

   ```
   # ls -l /etc/hostname*
   -rw-r--r--   1 root   root      23 Jun 15 14:36 /etc/hostname.dmfe0
   ```

   Since Netra X1 has two Ethernet interfaces and the primary is dmfe0, we can make an educated guess that the secondary interface is dmfe1. Now we only have to create the hostname6 files.

   ```
   # touch /etc/hostname6.dmfe0 /etc/hostname6.dmfe1
   # ls -l /etc/hostname*
   -rw-r--r--   1 root   root      23 Jun 15 14:36 /etc/hostname.dmfe0
   -rw-r--r--   1 root   other     0 Sep 12 13:37 /etc/hostname6.dmfe0
   -rw-r--r--   1 root   othe      0 Sep 12 13:37 /etc/hostname6.dmfe1
   ```

   Normally we would reboot the box right after creating hostname6.*interface*, but this box is going to be a router. Hence, change the default directory to /etc/inet and create ndpd.conf to set up prefixes for Neighbor Discovery.

2. Edit /etc/inet/ndpd.conf to set up prefixes for each IPv6 interface. To keep things simple we are going to use site-local addresses in our example. The following sample configuration has been adapted from Solaris documentation:

   ```
   # cd /etc/inet
   # ex ndpd.conf
   "ndpd.conf" [New file]
   :a
   ifdefault AdvReachableTime 30000 AdvReTransTimer 2000
   prefixdefault AdvValidLifetime 240m AdvPreferredLifetime 120m

   # 0xa801 = 168.1
   if dmfe0 AdvSendAdvertisements 1
   ```

```
prefix fec0:0:0:a801::/64 dmfe0

# 0xa802 = 168.2
if dmfe1 AdvSendAdvertisements 1
prefix fec0:0:0:a802::/64 dmfe1
.
:wq
"ndpd.conf" 8 lines, 248 characters
# reboot
```

Now we are all set for reboot. When the system boots up it will create the IPv6 interfaces and auto-configure them. Consequently, the in.ndpd auto-configuration daemon will start up and assign the prefixes for the interfaces. It will also hand out the prefixes for all the stations on the local Ethernet. At the end of the process, the RIPng routing daemon will start up to advertise and listen for route updates on both sides of the router.

3. Log in and check the IPv6 interfaces list after the host has booted up. You should see the link-local interfaces the host has configured for itself and the site-local prefix it has received from the router:

```
# ifconfig -a6
lo0: flags=2000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv6> mtu 8252 index 1
    inet6 ::1/128
dmfe0: flags=2100841<UP,RUNNING,MULTICAST,ROUTER,IPv6> mtu 1500 index 2
    ether 0:3:ba:6:14:66
    inet6 fe80::203:baff:fe06:1466/10
dmfe0:1: flags=2180841<UP,RUNNING,MULTICAST,ADDRCONF,ROUTER,IPv6> mtu 1500 index 2
    inet6 fec0::a801:203:baff:fe06:1466/64
dmfe1: flags=2100841<UP,RUNNING,MULTICAST,ROUTER,IPv6> mtu 1500 index 3
    ether 0:3:ba:6:14:66
    inet6 fe80::203:baff:fe06:1466/10
dmfe1:1: flags=2180841<UP,RUNNING,MULTICAST,ADDRCONF,ROUTER,IPv6> mtu 1500 index 3
      inet6 fec0::a802:203:baff:fe06:1466/64
```

4. Before we continue any further, it is a good practice to record the IP addresses in /etc/inet/ipnodes and enable IPv6 DNS lookups in /etc/nsswitch.conf:

```
# vi /etc/inet/ipnodes
"/etc/inet/ipnodes" [Read only] 17 lines, 500 characters
:a
fe80::203:baff:fe06:1466        sun-gw.v6.mydomain.org sun-gw.v6 ll-sun-gw
fec0::a801:203:baff:fe06:1466   dmfe0-sun-gw.v6.mydomain.org dmfe0-sun-gw
fec0::a802:203:baff:fe06:1466   dmfe1-sun-gw.v6.mydomain.org dmfe1-sun-gw
.
:wq!
"/etc/inet/ipnodes" 18 lines, 522 characters
```

It is a bit difficult to find reasonable IPv6 naming conventions on Sun hardware, since Sun insists on using the same MAC address on every interface. One way around this is to use the ll- prefix for all the link-local addresses and the interface name for all routable addresses. It is not perfect but, after all, /etc/inet/ipnodes is like /etc/hosts: you use the file to get the most critical interface names (i.e., yours) and obtain everything else from the DNS. In other words, this minimal /etc/inet/ipnodes would be as follows:

```
::1                             localhost.mydomain.org localhost
fe80::203:baff:fe06:1466        sun-gw.v6.mydomain.org sun-gw.v6 ll-sun-gw
```

For maximum flexibility you may not even want to have anything other than the link-local address in /etc/inet/ipnodes , since the link-local address will always be associated with the device. Otherwise, all routable addresses, be those site-local or aggregateable global unicast ones, become elusive. If the host knows only its link-local address and gets everything else from the DNS, there is one issue less to worry about when you eventually have to renumber the network.

Finally, the -gw suffix to the hostname is in accordance with the ipnodes(4) manual page, which, in turn, refers to RFC-952.[22]

5. We also have to edit /etc/nsswitch.conf to enable host lookups from DNS as well as /etc/inet/ipnodes:

```
# ex /etc/nsswitch.conf
"/etc/nsswitch.conf" 26 lines, 690 characters
:/^ipnodes
ipnodes:    files
:s/files/files dns/
ipnodes:    files dns
:wq!
"/etc/nsswitch.conf" 26 lines, 694 characters
```

This is everything you have to do to get started. The existing hosts on both sides will learn of the gateway when it boots up and will adjust the interface and routing tables accordingly. For example, there is a new IPv6 address for the Solaris workstation we configured earlier:

```
# ifconfig -a6
lo0: flags=2000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv6> mtu 8252 index 1
        inet6 ::1/128
eri0: flags=2000841<UP,RUNNING,MULTICAST,IPv6> mtu 1500 index 2
        ether 0:3:ba:e:6a:6a
        inet6 fe80::203:baff:fe0e:6a6a/10
eri0:1: flags=2080841<UP,RUNNING,MULTICAST,ADDRCONF,IPv6> mtu 1500 index 2
        inet6 fec0::a801:203:baff:fe0e:6a6a/64
```

Similarly, the routing table has changed to indicate a new default gateway:

```
Routing Table: IPv6
  Destination/Mask            Gateway                   Flags     Ref     Use     If
--------------------------- --------------------------- -------   -----   ------  -------
fec0:0:0:a801::/64          fec0::a801:203:baff:fe0e:6a6a U         1       2       eri0:1
fe80::/10                   fe80::203:baff:fe0e:6a6a    U         1       1       eri0
ff00::/8                    fe80::203:baff:fe0e:6a6a    U         1       0       eri0
default                     fe80::203:baff:fe06:1466    UG        1       0       eri0
::1                         ::1                         UH        1       0       lo0
```

And this is how it looks on FreeBSD:

```
Routing Tables

Internet6:
Destination                       Gateway                       Flags      Netif  Expire
::/96                             ::1                           UGRSc      lo0    =>
default                           fe80::203:baff:fe06:1466%ed1  UGc        ed1
::1                               ::1                           UH         lo0
::ffff:0.0.0.0/96                 ::1                           UGRSc      lo0
fe80::/10                         ::1                           UGRSc      lo0
fe80::%lo0/64                     fe80::1%lo0                   Uc         lo0
fe80::1%lo0                       link#2                        UHL        lo0
fe80::%ed1/64                     link#3                        UC         ed1
fe80::203:baff:fe06:1466%ed1      00:03:ba:06:14:66             UHLW       ed1
fe80::2e0:98ff:fe83:48d0%ed1      00:e0:98:83:48:d0             UHL        lo0
fec0:0:0:a823::/64                link#3                        UC         ed1
fec0::a801:203:baff:fe06:1466     00:03:ba:06:14:66             UHLW       ed1
fec0::a801:2e0:98ff:fe83:48d0     00:e0:98:83:48:d0             UHL        lo0
ff01::/32                         ::1                           U          lo0
ff02::/16                         ::1                           UGRS       lo0
ff02::%lo0/32                     ::1                           UC         lo0
ff02::%ed1/32                     link#3                        UC         ed1
```

If sometime in the future you decide to open up your network to the world and get connected to 6bone, you only have to advertise the new 6bone TLA or production sub-TLA prefix and the hosts will pick it up instantly.

MANUAL CONFIGURATION

It is also possible to configure the IPv6 address manually on Solaris. This makes little sense on a local LAN, but it is quite necessary, for example, when you have to define the endpoints of an IPv6 over IPv4 tunnel. However, on a local LAN this would only contribute to the administrator's headache, since you would have to manually change this configuration every time you renumber the local LAN. If you used auto-configuration instead, the nearest router would take care of handing out the prefixes.

To summarize, there are situations when you have to configure the IP address manually, but the local LAN would not be involved in any of those. However, for the sake of simplicity the following example will show manual configuration of the primary Ethernet interface:

1. Check the interface name for your primary Ethernet interface:

```
# ls -l /etc/hostname*
-rw-r--r-- 1 root    root   23 Jun 15 14:36 /etc/
                                          hostname.eri0
```

2. Enter the prefix parameters into /etc/hostname6.eri0:

```
# cat > /etc/hostname.eri0
addif fec0::2001:203:baff:fe0e:6a6a up
Ctrl-D
#
```

3. Reboot.

HOW DOES IT WORK?

Since RFC-2462 specifies the process for stateless address auto-configuration, we will only examine the process as it appears on the network. There are some variations to the process, depending on your particular stack implementation, but Figure 15 displays the operation at a generic level:

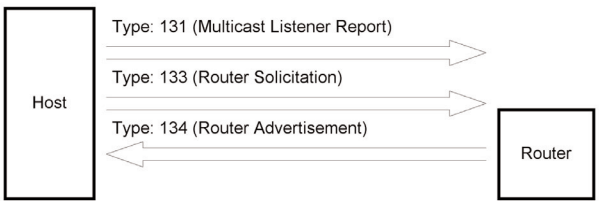

Figure 15. Stateless Address Auto-configuration

The first step in auto-configuration is for the host to generate the link-local address for itself. If the host is connected to a LAN and therefore has a MAC address, this is a fairly straightforward procedure:

0:3:ba:e:6a:6a → fe80::203:baff:fe0e:6a6a

Although MAC addresses are supposed to be globally unique, this is not always the case. Some vendors have used unregistered or duplicate MAC addresses in their low-end Ethernet interfaces and there is a possibility that one of these devices is on the same LAN segment with the auto-configuring host.

The duplicate detection algorithm is relatively straightforward. After the host has configured a link-local address for itself, it will join the All-Nodes multicast group by sending a Multicast Listener Report and announcing the multicast address it is listening to:

Figure 16. Multicast Listener Report

If there is another host listening at the same address, it will notify the host and give its link-local address. If there is a collision, the host may modify its address and try again, or stop the process altogether and alert the operator to intervene. Duplicate address detection is not perfect; it cannot detect offline hosts or the reply may be lost due to a collision. Duplicate detection still is a cheap sanity check to prevent the most obvious address problems.

After the host has verified that it has a more or less unique link-local address, it continues the auto-configuration process by sending a Router Solicitation to the All-Routers multicast address:

Figure 17. Router Solicitation

The router on the local segment will reply with a Router Advertisement sent to the All-Nodes multicast address (Figure 18). The Router Advertisement will contain the prefixes for the network and the lifetime of the prefix. Note that it makes perfect sense to use the All-Nodes address for the Router Advertisement. There could be multiple hosts in the midst of auto-configuration, in which case it is cheaper to use multicast as opposed to replying with a unicast to each and every host that sent a Router Solicitation.

Figure 18. Router Advertisement

Since high availability seems to be the most repeated mantra in quite a few TCP/IP networks, there is a tendency to have a router pair on critical LAN (i.e., Ethernet) segments. Since the routers are supposed to use HSRP or VRRP to monitor each other's status, only the master router can reply to the Router Advertisement. Depending on the configuration, the master may use either its own interface address or the high availability virtual address.

A host should not know anything about routing. The idea of Router Solicitation and Router Advertisement was already in ICMPv4 but it did not gain much popularity. IPv6 takes the concept further, to the extent that one cannot have a routable IPv6 address if there is no router to hand out the prefix. Yet the hosts can communicate with each other on the local segment, using the link-local addresses.

Vendors, such as Sun Microsystems, may decide to add their own solicitation messages to auto-configuration (Figure 19). Sun Solaris registers itself to the multicast group ff02:202 a.k.a. Sun RPC PMAPPROC_CALLIT. This group appears to be for the communication of Sun RPC portmapper to locate some particular RPC service.

Figure 19. Sun RPC Portmapper

Stateless address auto-configuration is a great concept, but like all concepts, it can be subverted by stupid designs. For example, Sun Netra X1 has two Ethernet interfaces on the motherboard. Some designers got the bright idea of using the two interfaces on different subnets and setting up route metrics to use either side as the default gateway. IPv6 auto-configuration will not permit this, since the router sending the Router Advertisement will also become the default gateway, and having two default gateways will cause asymmetric routing. One would have to bypass auto-configuration with fixed prefixes and run RIPng on the host without packet forwarding to enable this kind of setup, thus losing the automations (and benefits of IPv6) described above.

IPv6 and DNS

The status of the DNS in the IPv6 world is still in flux. BIND 8 implemented the RFC-1886[23]-specified IPv6 DNS extensions such as the AAAA resource record and the ip6.int domain for reverse lookups. There is no true IPv6 name resolution over IPv6 with BIND 8, unless the server and the resolver have been patched for IPv6. Note that Solaris 9 and FreeBSD-4.x include BIND 8 in the distribution.

To confuse things further, RFC-2874[24] introduced a list of new DNS extensions for address aggregation and network renumbering and changed the outlook of IPv6 name resolution completely. This was not entirely a bad thing, since old A and PTR records were specified in the days of the Class A, B, and C networks. Although there was name delegation for forward maps, reverse map delegations for CIDR blocks have been more or less painful. CIDR blocks and network renumbering, the exercise every administrator will go through sooner or later, were not enough of a concern for forward maps.

RFC-1886 basically extended the old DNS concepts to the IPv6 world with little thought put into aggregateable global unicast addresses, TLAs, NLAs, or SLAs. With 16 octets of an address, CIDR blocks and renumbering became major issues, and the A6 and DNAME resource records and the ip6.arpa domain were proposed. The BIND 9 implementation prefers RFC-2874 but provides support for RFC-1886. The following quote is from the BIND 9 Administrator Reference Manual.[25]

> For forward lookups, BIND 9 supports both A6 and AAAA records. The use of AAAA records is deprecated, but it is still useful for hosts to have both AAAA and A6 records to maintain backward compatibility with installations where AAAA records are still used. In fact, the stub resolvers currently shipped with most operating systems support only AAAA lookups, because following A6 chains is much harder than doing A or AAAA lookups.
>
> For IPv6 reverse lookups, BIND 9 supports the new "bit-string" format used in the ip6.arpa domain, as well as the older, deprecated "nibble" format used in the ip6.int domain.

BIND 9 name server can listen to IPv6, enabling the users to use IPv6 for name lookups if they have the new Lightweight Resolver installed. The introduction of a new resolver is due to simultaneous IPv4 and IPv6 lookups and the added complexity of the A6 and DNAME resource records.

To summarize, DNS support for IPv6 will be at an introductory phase until BIND 9 and the Lightweight Resolver are available for all operating systems that have an IPv6 stack. Even then there will be a considerable installed base of BIND 8 sites, which will have a negative effect in the rollout of IPv6.

RFC-1886 Lookups

FORWARD LOOKUPS

From the perspective of the named configuration file, IPv6 forward lookup zones are no different from the IPv4 ones. The configuration file specifies the name of the zone, and the protocol-specific details are kept in the zone file. For example, there isn't anything IPv6-specific in the following zone entry from the master configuration file, except the v6 subdomain:

```
zone "v6.mydomain.org" in {
        type master;
        file "db.v6.mydomain.org";
};
```

The AAAA resource record is the IPv6 equivalent of the IPv4 A record. The syntax of the AAAA resource record is exactly the same as the A record, except that the addresses take longer to type. For example, consider the following entries for the zone v6.mydomain.org:

```
$ORIGIN v6.mydomain.org.
localhost           86400     IN     AAAA    ::1
mysun               86400     IN     AAAA    fec0::a801:203:baff:fe0e:6a6a
dmfe0-sun-gw        86400     IN     AAAA    fec0::a801:203:baff:fe06:1466
dmfe1-sun-gw        86400     IN     AAAA    fec0::a802:203:baff:fe06:1466
```

REVERSE LOOKUPS

The reverse lookups are performed using the nibble format and the ip6.int domain. One octet or byte consists of two nibbles, e.g., decimal 192 equals 0xc0, which has nibbles 0xc and 0x0. The nibble format is the IPv6 equivalent of the traditional IPv4 in-addr.arpa representation, and the logic of the nibble format is exactly the same. Since the IPv6 addresses are hexadecimal, it makes a certain sense to reverse the order of the address and separate each nibble with a dot. This permits reverse zone delegation similar to the one in IPv4.

First we will examine the reverse map of the IPv6 localhost. The normal abbreviation for the localhost address is:

```
::1
```

This is equivalent to the expanded address:

```
0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1
```

Consequently, we could have the following PTR resource record for localhost:

```
$ORIGIN 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.ip6.int.
1    86400   IN     PTR      localhost.v6.mydomain.org.
```

This translates to the following zone entry in the master configuration file:

```
zone "0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.ip6.int" in {
        type master;
        file "db..1";
};
```

The reverse zone entries for the prefixes fec0:0:0:a801::/64 and fec0:0:0:a802::/64 are not much different. Since we have to reverse the order of nibbles, we get the following PTR resource records:

```
$ORIGIN 1.0.8.a.0.0.0.c.e.f.ip6.int.
a.6.a.6.e.0.e.f.f.f.a.b.3.0.2.0          86400      IN          PTR      mysun.v6.mydomain.org.
0.d.8.4.3.8.e.f.f.f.8.9.0.e.2.0          86400      IN          PTR      mybsd.v6.mydomain.org.
6.6.4.1.6.0.e.f.f.f.a.b.3.0.2.0          86400      IN          PTR      dmfe0-sun-gw.v6.mydomain.org.

$ORIGIN 2.0.8.a.0.0.0.c.e.f.ip6.int.
6.6.4.1.6.0.e.f.f.f.a.b.3.0.2.0          86400      IN          PTR      dmfe1-sun-gw.v6.mydomain.org.
```

And these are the respective zone master configuration file entries:

```
zone "1.0.8.a.0.0.0.c.e.f.ip6.int" in {
        type master;
        file "db.fec0..a801";
};
```

```
zone "2.0.8.a.0.0.0.c.e.f.ip6.int" in {
        type master;
        file "db.fec0..a802";
};
```


The most obvious problem of the nibble format becomes very clear very quickly: The address prefixes and the interface identifiers are difficult to carry around, and each resource record takes a lot of proofreading to get it right. Reverse zone delegation may help here to limit the size of a single resource record but, on the other hand, it makes little sense to delegate to the interface identifier (e.g., the 203:baff portion of the aforementioned addresses). However, it does not help to set up a separate root zone for the fec0::/10 prefix, since one has to define fec0:0:0:a801::/64 and fec0:0:0:a802::/64 in the master configuration file and the zone files anyway.

RESOLVER

Note that we did not set up any resolver entry in the discussion on stateless address auto-configuration. This process could be automated with stateful address auto-configuration, but since DHCP6 is still in the works we have to set up our configuration file manually. Since we already made the decision to keep all IPv6 hosts in a separate .v6 subdomain and worry about migration later, we have to have both domain names in a search list. Moreover, since the BIND 8 resolver uses IPv4 only, we have to list our name server(s) as usual:

```
search mydomain.org v6.mydomain.org
nameserver 192.168.1.36
```

Since IPv4 applications are supposed to be in the majority, this setup will first look up for IPv4 names with mydomain.org. Since IPv6 applications do not care about A records, the lookup will proceed using v6.mydomain.org, hoping to find AAAA records. Telnet, FTP, and ping (on Solaris) will use IPv6 if available and revert back to IPv4 by default.

BIND 9 AND RFC-2874

FORWARD LOOKUPS

RFC-2874 introduced the new A6 resource record to eventually replace the AAAA record. It can be used just like the AAAA record, as the following example will show:

```
$ORIGIN v6.mydomain.org.
localhost       86400   IN   A6   0    ::1
mysun           86400   IN   A6   0    fec0::a801:203:baff:fe0e:6a6a
mybsd           86400   IN   A6   0    fec0::a801:2e0:98ff:fe83:48d0
dmfe0-sun-gw    86400   IN   A6   0    fec0::a801:203:baff:fe06:1466
dmfe1-sun-gw    86400   IN   A6   0    fec0::a802:203:baff:fe06:1466
```

These entries are no different from the aforementioned AAAA records. Site-local addresses gain little from the A6 records, since the network space is only 16 bits wide and you can have only so much address aggregation in two octets. On the other hand, changing the prefix for all the zones at a large site takes a considerable amount of work. To avoid this we can rewrite the zone by using all zeroes for the prefix:

```
$ORIGIN v6.mydomain.org.
mysun           86400   IN   A6   48   0:0:0:a801:203:baff:fe0e:6a6a prefix.v6.mydomain.org.
mybsd           86400   IN   A6   48   0:0:0:a801:2e0:98ff:fe83:48d0 prefix.v6.mydomain.org.
dmfe0-sun-gw    86400   IN   A6   48   0:0:0:a801:203:baff:fe06:1466 prefix.v6.mydomain.org.
dmfe1-sun-gw    86400   IN   A6   48   0:0:0:a802:203:baff:fe06:1466 prefix.v6.mydomain.org.
```

We can then specify the prefix with a separate resource record:

```
prefix          86400   IN   A6   0    fec0:0:0::
```

First, the lookup for mysun.v6.mydomain.org will get the A6 record 0:0:0:a801:203:baff:fe0e:6a6a. Since this record is incomplete, the query will follow the pointer to prefix.v6.mydomain.org and will receive the prefix fec0:0:0::. Then resolver on the querying node

will construct the IPv6 address from the local part and the prefix and return fec0::a801:203:baff:fe0e:6a6a as the address. This process is known as the A6 chain. This is how it works with dig:

```
# dig @::1 mysun.v6.mydomain.org. A6
; <<>> DiG 9.2.1 <<>> @::1 mysun.ipv6.mydomain.org. A6
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 13152
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;mysun.ipv6.mydomain.org.          IN    A6

;; ANSWER SECTION:
mysun.ipv6.mydomain.org. 86400  IN    A6    64 ::203:baff:fe0e:6a6a prefix.ipv6.mydomain.org.

;; AUTHORITY SECTION:
ipv6.mydomain.org. 86400          IN    NS    localhost.ipv6.mydomain.org.

;; ADDITIONAL SECTION:
prefix.ipv6.mydomain.org. 86400   IN    A6    0 fec0:0:0:a801::

;; Query time: 1 msec
;; SERVER: ::1#53(::1)
;; WHEN: Thu Sep 19 20:37:03 2002
;; MSG SIZE  rcvd: 146
```

There is no predefined limit to the number of hops in the A6 chain. However, since it is up to the querying application – dig in the example above – to construct the A6 chain, it makes sense to keep the chain as short as possible. If you use other tools, such as nslookup or host, you have to follow the chains manually.

REVERSE LOOKUPS

Nibble format may be rather error-prone since if something is difficult to read, it is also probably difficult to get right. The other improvement of RFC-2874 is the bitstring format for reverse zones. Simply put, the reverse zone example above looks like the following when using bitstrings:

```
$ORIGIN \[x00000000000000000000000000000000/124].ip6.arpa.
\[x1/4]                         86400   IN    PTR     localhost.v6.mydomain.org.

$ORIGIN \[xfec000a801/64].ip6.arpa.
\[x0203bafffe0e6a6a/64]    86400   IN    PTR     mysun.v6.mydomain.org.
\[x02e098fffe8348d0/64]    86400   IN    PTR     mybsd.v6.mydomain.org.
\[x0203bafffe061466/64]    86400   IN    PTR     dmfe0-sun-gw.v6.mydomain.org.

$ORIGIN \[xfec000a801/64].ip6.arpa.
\[x0203bafffe061466/64]    86400   IN    PTR     dmfe1-sun-gw.v6.mydomain.org.
```

The corresponding master configuration file entries are as follows:

```
zone "\[x00000000000000000000000000000000/124].ip6.arpa" in {
      type master;
      file "db..1";
};

zone "\[xfec000a801/64].ip6.arpa" in {
      type master;
      file "db.fec0..a801";
};
```

```
zone "\[xfec000a801/64].ip6.arpa" in {
        type master;
        file "db.fec0..a802";
};
```

At least bitstrings produce a more readable reverse map representation than nibble format.

The last IPv6 DNS extension of RFC-2874 is DNAME. It is only normal for an IPv6 host to have at least two addresses, link-local address and site-local address, and possibly an aggregateable global unicast address as well, yet all these addresses share the same interface identifier. Although it is normal routine to maintain these resource records in one forward zone, the reverse maps will certainly go to three different zones. For example, mysun.v6.mydomain.org may have the addresses found in Table 3:

ADDRESS	DESCRIPTION
fe80:: 203:baff:fe0e:6a6a	Link-local Address
fec0::a801:203:baff:fe0e:6a6a	Site-local Address
2001:11f8:5ef9:a801:203:baff:fe0e:6a6a	Aggregatable Global Unicast Address

Table 3. Possible Addresses for a Single Host

The reverse zone record for the link-local address would be as follows:

```
$ORIGIN \[xfe80/16].ip6.arpa.
\[x000000000000/48]     86400   IN    DNAME          v6-rev.mydomain.org.
```

Similarly, the site-local address would have the following reverse zone record:

```
$ORIGIN \[xfec0/16].ip6.arpa.
\[x00000000a801/48]     86400   IN    DNAME          v6-rev.mydomain.org.
```

The aggregateable global unicast address would follow with a similar entry:

```
$ORIGIN \[x200111f85ef9/48].ip6.arpa.
\[xa801/16]             86400   IN    DNAME          v6-rev.mydomain.org.
```

Finally, the host entry would specify the interface identifier:

```
$ORIGIN v6-rev.mydomain.org.
\[x0203bafffe0e6a6a/64] 86400   IN    PTR            mysun.v6.mydomain.org.
```

DNAME can be used very effectively in reverse maps, since the interface identifier alone contains twice as much information as an IPv4 address. Moreover, renumbering a network may not happen by turning a switch: It requires a lot of careful planning and the transition period can be several months. A reverse map is usually the first thing about the DNS that will be ignored, and, hopefully, DNAME is powerful enough to help the DNS administrators with this area.

Finally, there are some interoperability considerations. Since BIND 8 and especially BIND 8–based resolver libraries are going to be around for a very long time, it is possible to configure BIND 9 to return AAAA records and PTR records in nibble format, even if the zone files use the A6 records and bitstrings. This feature is off by default and can be activated on a per-client basis:

```
options {
        allow-v6-synthesis {
            192.168.1.30;
        };
};
```

RESOLVER

The new BIND 9 Lightweight Resolver consists of two components: the stub library accessed by the applications and the lwresd resolver daemon responsible for processing the forward and reverse lookups (Figure 20). This new setup was mandated by the A6 and DNAME chains, which could have any number of links. The chains are resolved by the daemon, which returns the completed IPv6 address to the resolver library.

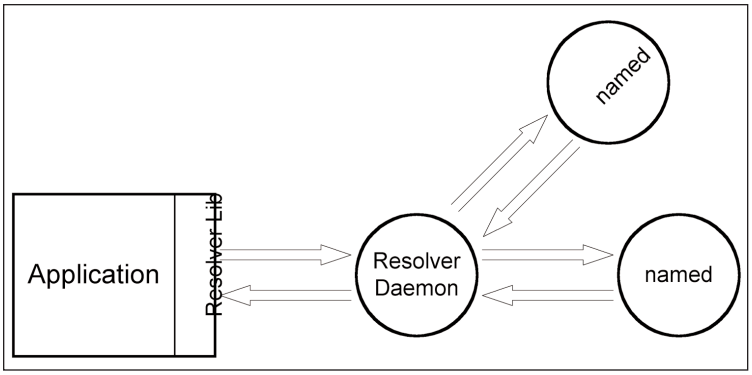

Figure 20. BIND 9 Name Resolution

The lwresd process is essentially a cache-only nameserver for localhost (127.0.0.1, ::1). The communication between the stub library and lwresd is UDP-based at port 921.

IP Security

IP Security, aka IPSec, was originally envisioned as the security architecture for IPv6, but it was retrofitted onto IPv4, mostly for Virtual Private Network tunnels, long before IPv6 had a chance to take off. Consequently, IPSec is almost always associated with VPNs, although it may prove its usefulness in host-to-host communication as well. Good news is that IPSec is a mandatory component of IPv6. The bad news is that we are not there yet.

The general architectural placement of the IPSec layer is between the transport and network layers. The concept of inserting a security protocol between transport and network protocols is actually quite old, since it was one of the ideas proposed in the OSI Security Framework.[26]

The general idea of the IPSec protocol is quite simple, since an IPSec packet consists of two elements: the Authentication Header (AH) and the Encapsulating Security Payload (Figure 21). Since the IPSec protocol takes the transport protocol data unit, authenticates and/or encrypts the data, and passes the completed IPSec packet to the network layer to be transmitted and routed, it can offer very limited protection for the IPv6 protocol. On the other hand, the transport protocol packet (e.g., TCP) enjoys the full protection of the IPSec layer.

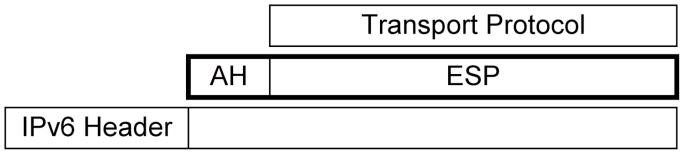

Figure 21. IPSec Packet

AH provides data integrity, data origin authentication, and optional replay protection. It is an authentication service that is used to verify the source of the data and that the data has not been modified while in transit, but it does not protect against eavesdropping. The possible authentication algorithms used by AH are HMAC-MD5 and HMAC-SHA-1.

ESP, on the other hand, offers data confidentiality through encryption. It is possible to combine AH and ESP to have data integrity, data origin authentication, and confidentiality without reapplying AH over an already authenticated packet. The possible encryption protocols are DES, 3DES, Blowfish, and AES. The key lengths are shown below in Table 4.

Since the IPSec architecture is transparent to the application, by definition it cannot be used to authenticate users. The end-user applications such as Telnet, FTP, or X11 must rely on their own methods to verify the authenticity of the users. In other words, if someone is able to establish an IPSec session to the Telnet port and the only authentication mechanism is the standard username and password, the system can be broken into over IPSec. Moreover, if the ESP protects the communication, the network intrusion detection system will never know that something went wrong.

IPSec, and ESP in particular, defeats firewalls too. A firewall cannot see inside the ESP payload and make any kind of filtering decision based on port numbers. In essence, the firewall will only know the end-points (i.e., IPv6 addresses) and has to rely on the host to make the correct decision whether to permit or deny communication for this particular application.

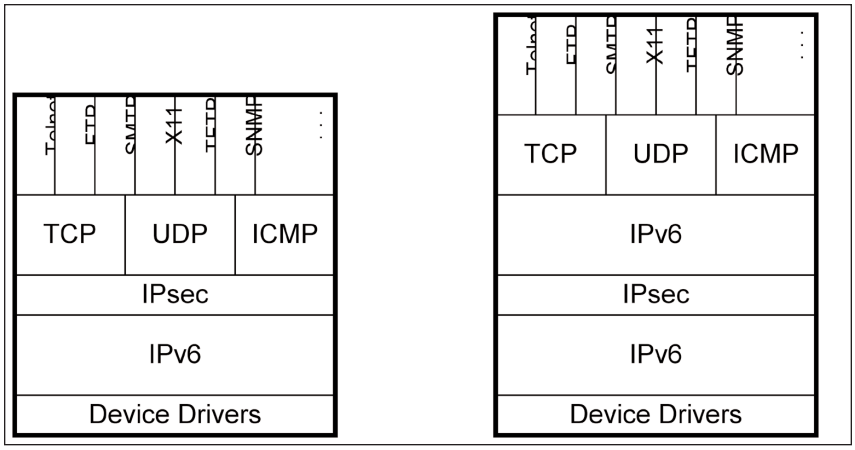

Figure 22. IPSec Transport Mode and Tunnel Mode

The two modes of IPSec are transport mode and tunnel mode (Figure 22). All VPNs use the tunnel mode, while host-to-host communication should use the transport mode.

The transport mode is meant to protect end-to-end communication between two hosts using IPSec. Since the network analyzers such as snoop and tcpdump attach between the device driver and the network layer, IPSec will protect the data in transit from these as well. Authentication and decapsulation of data takes place after the datagram has been successfully received and processed by the network protocol.

As mentioned earlier, VPNs have been the primary users of IPSec, to the extent that IPSec is often associated with the tunnel mode. Boosted by the VPN market, the tunnel mode has turned into another link layer technology, albeit cheaper than frame relay, ATM, or (fractional) T1. The main security issue with a VPN is to determine whether the other network really is trustworthy.

There have been occasions in which an application needed data integrity and confidentiality. A good example is a query application to a database that contains sensitive financial or other data. Since it may not have been possible to protect the data in transit with SSL/TLS, and the hacks to use SSH as a secure proxy, users were somewhat forced to set up a VPN from the server network to their own group. This kind of setup may raise an eyebrow or two, but such setups are often implemented, since getting the job done is usually more important for a corporate environment than waiting for the one true solution. That one true solution could well be the IPSec transport mode, since it can be configured to protect end-to-end communication to the database application and pass everything else between the hosts in clear. To summarize, the transport mode can put an end to one-application VPNs.

Set It Up

In the following section we will discuss how to set up KAME IPSec in transport mode to protect traffic between two systems. Since IPSec provides authentication, data integrity, and confidentiality services, we will discuss trivial host authentication for an inbound connection and then move to a more complex setup using both AH and ESP.

By default IPSec is not compiled into the FreeBSD-4.x kernel. Before proceeding with this discussion you may want to check that the following options have been defined in the kernel configuration file:

```
options   IPSEC          #IP security
options   IPSEC_ESP      #IP security (crypto; define w/
                           IPSEC)
options   IPSEC_DEBUG    #debug for IP security
```

You may also want to issue the setkey -D command to find out whether setkey(8) can talk to the IPSec kernel. If you get any kind of error message, you probably don't have IPSec in the kernel. Follow the instructions in the FreeBSD Handbook to compile a new kernel.

Perhaps the trickiest part in IPSec is defining the security associations and the policy controlling the associations. For example, SSH and applications using SSL/TLS are on top of the transport layer, i.e., the transport protocol has already set up a two-way communication channel between the endpoints. The security context is implicit, since it is defined by the underlying TCP connection from Host A to Host B: We know that there is a TCP connection from A to B and we only have to provide a secure channel from A to B to keep the data safe. In contrast, since the IPSec layer is below the transport protocol, all security contexts must be explicitly defined: It is not enough to set up a context from A to B, but one has to define the context from B to A as well.

The various address types of IPv6 add another dimension to the IPSec setup. If a host has a site-local address and an aggregateable global unicast address in addition to the link-local address, all these address types must be specified in the IPSec policy. It would be somewhat silly to have a policy for the link-local address but have no policy whatsoever for the routable addresses. Consequently, an IPSec policy that applies to the routable addresses but omits the link-local one leaves a back door to the host.

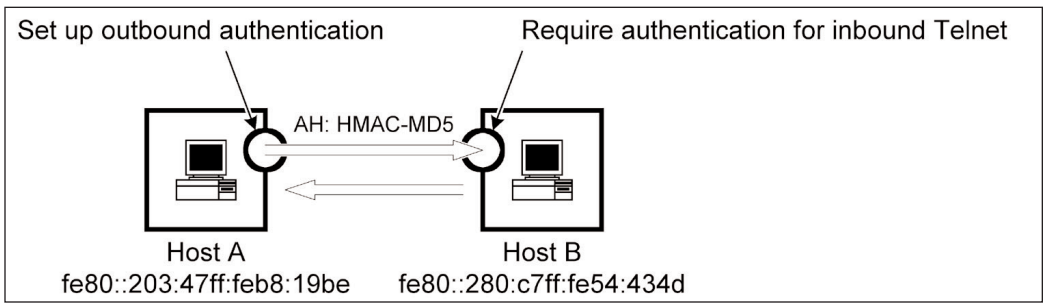

Figure 23. AH required by Host B

Host Authentication

The first example is a simple authentication between two FreeBSD hosts using HMAC-MD5. The link-local address of Host A is fe80::203:47ff:feb8:19be and Host B is fe80::280:c7ff:fe54:434d. For the sake of simplicity we will exclusively use link-local addresses. Any traffic to the Telnet port on Host B must be authenticated at the IPSec layer for further communication to take place. However, no IPSec will apply to the reply packets from B to A. This setup is in Figure 23:

One might find this type of setup suitable to replace, or to use together with, TCP Wrappers. Since TCP Wrappers make the access control decision based on the IP address, one could further strengthen the access control setup by demanding authentication via IPSec before the packet is even passed to TCP Wrappers.

Note that different IPSec algorithms specify different key lengths and that you must have the right key length for the algorithm to accept the key. Table 4 displays the various KAME IPSec authentication and encryption algorithms, and the respective key lengths:

	Key Length		
Algorithm	Bits	Bytes	Description
hmac-md5	128	16	ah: RFC-2403
	128	16	ah-old: RFC-2085
hmac-sha1	160	20	ah: RFC-2404
	160	20	ah-old: 128-bit ICV (no document)
keyed-md5		128	16	ah: rfc1828
		128	16	ah-old: 96-bit ICV (no document)
keyed-sha1		160	20	ah: 96-bit ICV (no document)
		160	20	ah-old: 128-bit ICV (no document)
null		0-2048	0-256	For debugging purposes
hmac-sha2-256		256	32	ah: 96-bit ICV (no document)
		256	32	ah-old: 128-bit ICV (no document)
hmac-sha2-384		384	48	ah: 96-bit ICV (no document
		384	48	ah-old: 128-bit ICV (no document)
hmac-sha2-512		512	64	ah: 96-bit ICV (no document)
		512	64	ah-old: 128-bit ICV (no document)
des-cbc		64	8	esp: RFC-1829
		64	8	esp-old: RFC-2405
3des-cbc		192	24	RFC-2451
simple		0-2048	0-256	RFC-2410
blowfish-cbc		40-448	5-56	RFC-2451
cast128-cbc		40-128	5-16	RFC-2451
des-deriv		64	8	ipsec-ciph-des-derived-01
3des-deriv		192	24	No document
rijndael-cbc		128/192/256	16/24/32	draft-ietf-ipsec-ciph-aes-cbc-00

Table 4. KAME IPSec Authentication Algorithms[27]

1. First we have to define the security associations on A and B in /etc/ipsec.conf. The following example will be used on Host A:

```
# cd /etc
# ex ipsec.conf
"ipsec.conf" [New File]
:a
```

```
add fe80::203:47ff:feb8:19be%fxp0 fe80::280:c7ff:fe54:434d%fxp0 ah 0x30e8 -m transport
                                -A hmac-md5 "mysecretissecret" ;

spdadd fe80::203:47ff:feb8:19be%fxp0 fe80::280:c7ff:fe54:434d%fxp0 [23] tcp -P out ipsec ah/transport//require ;

.
:wq
"ipsec.conf" [New File] 2 lines, 218 characters written
```

This entry is similar on A and B, with the exception of the interface suffix (i.e., fxp0 on Host A and xe0 on Host B). Note that each statement must fit on one line. Line feeds and line escapes (\) are not permitted. The hanging indent in the example above is only for typographical reasons.

The syntax for add and spdadd can be found from the setkey(8) manual page. Note that the Security Parameter Index (0x3048) for add should be a random number, which has to be greater than or equal to 256. This number will be passed between A and B to determine the correct security association between the systems. And since the AH algorithm is HMAC-MD5, the shared key must be 16 bytes long.

The spdadd entry will define the security policy for the security association. The security association is between the hosts A and B, and the association will become active when A opens a TCP connection to the port 23 (Telnet) on the Host B. All other communication (e.g., FTP, TFTP, or ping) will take place outside IPSec.

2. Load the key and the policy into the operating system kernel with the setkey(8):

```
# setkey -f /etc/ipsec.conf
```

3. Set up Host B similarly to require AH authentication for inbound traffic:

```
# cd /etc
# ex ipsec.conf
"ipsec.conf" [New File]
:a
add fe80::203:47ff:feb8:19be%fxp0 fe80::280:c7ff:fe54:434d%fxp0 ah 0x30e8 -m transport
                              -A hmac-md5 "mysecretissecret" ;

spdadd fe80::203:47ff:feb8:19be%fxp0 fe80::280:c7ff:fe54:434d%fxp0 [23] tcp -P out ipsec ah/transport//require ;

.
:wq
"ipsec.conf" [New File] 2 lines, 218 characters written
```

Of course, instead of specifying Host A for spdadd one could use the IPv6 wildcard address (::) to require AH for all connection attempts to the Telnet port.

4. Load the configuration into the kernel on Host B:

```
# setkey -f /etc/ipsec.conf
```

5. Once the IPSec configuration is active, you can view the security association database with setkey -D:

```
# setkey -D
fe80::203:47ff:feb8:19be%xe0 fe80::280:c7ff:fe54:434d%xe0
    ah mode=any spi=12520(0x000030e8) reqid=0(0x00000000)
    A: hmac-md5  6d797365 63726574 6d797365 63726574
    seq=0x0000000e replay=0 flags=0x00000040 state=mature
    created: Sep 22 14:42:32 2002    current: Sep 22 14:49:27 2002
    diff: 415(s         hard: 0(s)          soft: 0(s)
    last: Sep 22 14:42:44 2002      hard: 0(s)   soft: 0(s)
    current: 1184(bytes)    hard: 0(bytes)     soft: 0(bytes)
    allocated: 1        hard: 0      soft: 0
    sadb_seq=2 pid=640 refcnt=1
```

On the other hand, the -DP command will show the effective IPSec policies. There should be only one entry since we didn't have anything else in /etc/ipsec.conf:

```
# setkey -DP
fe80::203:47ff:feb8:19be%xe0 [any] fe80::280:c7ff:fe54:434d%xe0[23] tcp
    in ipsec
    ah/transport//require
    spid=45 seq=1 pid=641
    refcnt=1
```

6. You can test the setup by opening a Telnet session from Host A to B:

```
# telnet -K fe80::280:c7ff:fe54:434d%fxp0
Trying fe80::280:c7ff:fe54:434d%fxp0...
Connected to fe80::280:c7ff:fe54:434d%fxp0
Escape character is '^]'.

FreeBSD/i386 (hostb) (ttyp2)

login:
```

We can monitor the progress of the connection with tcpdump. The first two packets exchanged at the beginning of the session are as follows:

```
fe80::203:47ff:feb8:19be > fe80::280:c7ff:fe54:434d: AH(spi=0x000030e8,seq=0xf): 1052 > 23: S
        3462415989:3462415989(0) win 16384 <mss 1440,nop,wscale 0,nop,nop,timestamp 527422 0>
fe80::280:c7ff:fe54:434d.23 > fe80::203:47ff:feb8:19be.1052: S 1868912997:1868912997(0) ack 3462415990
        win 57344 <mss 1440,nop,wscale 0,nop,nop,timestamp 752453 527422>
```

Obviously, traffic from Host A to Host B contains the AH header with the Security Parameter Index of 0x30e8, and the acknowledgments from B to A are normal TCP packets. If you unloaded the policy on B or changed the key on either host, it would be impossible for Host B to connect:

```
# telnet -K fe80::280:c7ff:fe54:434d%fxp0
Trying fe80::280:c7ff:fe54:434d%fxp0...
```

Although having host-based authentication before the connection is really opened sounds like a good idea, there are at least two security issues with this setup. First, the Security Parameter Index is static. Our packet capture with tcpdump(8) shows clearly how the SPI is exchanged between the hosts in the clear. The ideal SPI is a large random number that changes for each security association. Similarly, the shared secret that is used over the connection is static, and it is stored into /etc/ipsec.conf and the security association database in cleartext.

AUTHENTICATE AND ENCRYPT DATA BETWEEN TWO HOSTS

In the second example we will secure all TCP communication between Host B running FreeBSD-4.x and Host C running Solaris 9. Before we can proceed, it is a good idea to flush the setup on Host B to avoid surprises:

```
# setkey -FP
# setkey -F
```

In this setup we will encrypt the payload, i.e., TCP packet, using 3DES-CBC in the ESP, and we will enforce data integrity with HMAC-MD5 in the AH. To make this setup more interesting, there will be one pair of shared keys, one for AH and another one for the ESP, for the direction Host C to Host B, and another key pair for traffic from Host B to Host C. This setup is shown in Figure 24 on the next page.

We will not use key management in this example, since in.iked(1m), the Solaris 9 IKE daemon, only works with IPv4.[28] Moreover, the KAME IKE daemon, racoon(8),[29] must be built separately from FreeBSD ports. To summarize, it may take a while before IKE is universally available for the various IPv6 IPSec implementations.

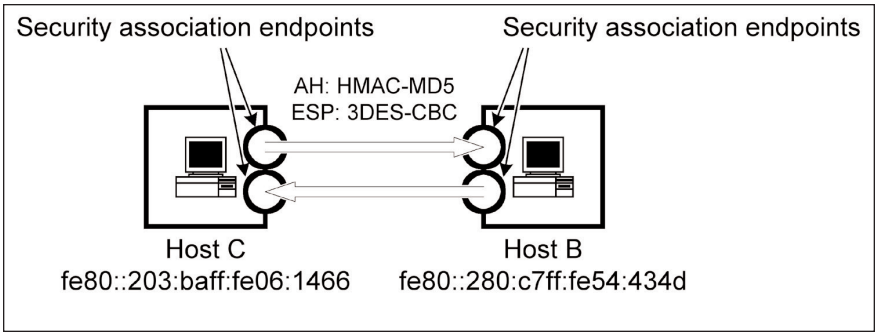

Figure 24. Security Association with AH and ESP

1. First we have to define the security associations and policies on FreeBSD. You may remove the old /etc/ipsec.conf and start working on a new one:

```
# rm ipsec.conf
# ex ipsec.conf
"ipsec.conf" [New File]
:a                                                    lines below broken for display
add fe80::203:baff:fe06:1466%xe0 fe80::280:c7ff:fe54:434d%xe0 esp 0xe63f -E
        3des-cbc "JnRs6Riou3wwlPHv5zdPED5k" -A hmac-md5 "hostahostbsecret" ;
add fe80::280:c7ff:fe54:434d%xe0 fe80::203:baff:fe06:1466%xe0 esp 0xf52a -E
        3des-cbc "G95lvYfQePVkul9Byvibi7fd" -A hmac-md5 "hostbhostasecret" ;

spdadd fe80::203:baff:fe06:1466%xe0 fe80::280:c7ff:fe54:434d%xe0 tcp -P in ipsec esp/transport//require ;
spdadd fe80::280:c7ff:fe54:434d%xe0 fe80::203:baff:fe06:1466%xe0 tcp -P out ipsec esp/transport//require ;
.
:wq
```

   Since the key length for 3DES-CBC is 192 bits, our shared keys must be 24 characters long. One should always use random strings as the keys, but for demonstration purposes we selected simple keys for authentication. In this case the encryption keys are random strings since Solaris ipseckeys(1m) would otherwise complain about weak keys.

2. Load the policy into the kernel:

```
# setkey -f /etc/ipsec.conf
```

3. Print out the entries in the security association database:

```
# setkey -D
fe80::280:c7ff:fe54:434d%xe0 fe80::203:baff:fe06:1466%xe0
    esp mode=any spi=62762(0x0000f52a) reqid=0(0x00000000)
    E: 3des-cbc  4739356c 76596651 6550566b 756c3942 79766962 69376664
    A: hmac-md5  686f7374 62686f73 74617365 63726574
    seq=0x0000003b replay=0 flags=0x00000040 state=mature
    created: Sep 24 15:05:37 2002current: Sep 24 15:17:35 2002
    diff: 718(s)    hard: 0(s)      soft: 0(s)
    last: Sep 24 15:14:15 2002      hard: 0(s)          soft: 0(s)
    current: 6484(bytes)      hard: 0(bytes     soft: 0(bytes)
    allocated: 59     hard: 0     soft: 0
    sadb_seq=1 pid=311 refcnt=2
fe80::203:baff:fe06:1466%xe0 fe80::280:c7ff:fe54:434d%xe0
    esp mode=any spi=58943(0x0000e63f) reqid=0(0x00000000)
    E: 3des-cbc  4a6e5273 3652696f 75337777 6c504876 357a6450 4544356b
    A: hmac-md5  686f7374 61686f73 74627365 63726574
```

```
seq=0x00000041 replay=0 flags=0x00000040 state=mature
created: Sep 24 15:05:37 2002    current: Sep 24 15:17:35 2002
diff: 718(s)    hard: 0(s)      soft: 0(s)
last: Sep 24 15:14:15 2002      hard: 0(s)            soft: 0(s)
current: 4801(bytes)      hard: 0(bytes)    soft: 0(bytes)
allocated: 65      hard:    0 soft: 0
sadb_seq=0 pid=311 refcnt=1
```

This command will dump the contents of the security administration database on-screen, including the AH and ESP keys in the hexadecimal format. Since we cannot enter ASCII keys into the Solaris IPSec key database (/etc/inet/secret/ipseckeys), we must copy the entries to the Solaris key file. This raises the age-old question of how to arrange secure key distribution to avoid sending the AH and the ESP keys over an insecure connection. SSH is always one solution, but it only provides a channel: The key distribution process is still largely manual work or requires custom scripts to extract shared secrets from the central host and distribute those to the remote hosts.

4. On Solaris, edit /etc/inet/secret/ipseckeys:

```
# cd /etc/inet/secret
# ex ipseckeys
"ipseckeys" [...]
:a
add esp spi 0x f52a src6 fe80::280:c7ff:fe54:434d dst6 fe80::203:baff:fe06:1466 \
    encralg 3des-cbc encrkey 4739356c765966516550566b756c39427976696269376664 \
    authalg hmac-md5 authkey 686f737462686f737461736563726574
add esp spi 0xe63f src6 fe80::203:baff:fe06:1466 dst6 fe80::280:c7ff:fe54:434d \
    encralg 3des-cbc encrkey 4a6e52733652696f753377776c504876357a64504544356b \
    authalg hmac-md5 authkey 686f737461686f737462736563726574
.
:wq
```

5. Now we have to load the keys into the Solaris IPSec kernel. This is accomplished with the ipseckey(1m) command:

```
# ipseckey -f ipseckeys
```

We can use ipseckey also to verify that the entries were loaded correctly:

```
# ipseckey -n dump
Base message (version 2) type DUMP, SA type ESP.
Message length 200 bytes, seq=1, pid=8407.
SA: SADB_ASSOC spi=0xe63f, replay=0, state=MATURE
SA: Authentication algorithm = HMAC-MD5
SA: Encryption algorithm = 3DES-CBC
SA: flags=0x80000000 < X_USED >
SRC: Source address (proto=0)
SRC: AF_INET6: port 0, fe80::203:baff:fe06:1466.
DST: Destination address (proto=0)
DST: AF_INET6: port 0, fe80::280:c7ff:fe54:434d.
AKY: Authentication key.
AKY: 686f737461686f737462736563726574/128
EKY: Encryption key.
EKY: 4a6e52733752686e753276766d514976347a64514545346b/192
LT: Lifetime information
CLT: 3072 bytes protected, 0 allocations used.
CLT: SA added at time Tue Sep 24 14:09:01 2002
CLT: SA first used at time Tue Sep 24 14:14:10 2002
CLT: Time now is Tue Sep 24 14:19:24 2002
```

```
Base message (version 2) type DUMP, SA type ESP.
Message length 200 bytes, seq=1, pid=8407.
SA: SADB_ASSOC spi=0x f52a, replay=0, state=MATURE
SA: Authentication algorithm = HMAC-MD5
SA: Encryption algorithm = 3DES-CBC
SA: flags=0x0 < >
SRC: Source address (proto=0)
SRC: AF_INET6: port 0, fe80::280:c7ff:fe54:434d.
DST: Destination address (proto=0)
DST: AF_INET6: port 0, fe80::203:baff:fe06:1466.
AKY: Authentication key.
AKY: 686f737462686f737461736563726574/128
EKY: Encryption key.
EKY: 4638346d765867516451576b756d38437976686268376764/192
LT: Lifetime information
CLT: 2944 bytes protected, 0 allocations used.
CLT: SA added at time Tue Sep 24 14:09:01 2002
CLT: Time now is Tue Sep 24 14:19:24 2002

Dump succeeded for SA type 0.
```

Now we only need the policy to control the use of IPSec.

6. Edit /etc/inet/ipsecinit.conf:

```
# cd /etc/inet
# ex ipsecinit.conf
:a
{
    laddr fe80::203:baff:fe06:1466
    raddr fe80::280:c7ff:fe54:434d
    ulp tcp
} ipsec {
    encr_algs 3des-cbc
    encr_auth_algs hmac-md5
    sa shared
}
.
:wq
```

In this file we will specify that one has to use 3DES-CBC encryption with HMAC-MD5 authentication for all TCP communication between Hosts B and C. Since the direction has not been specified, the policy will apply to both directions. The syntax of ipsecconf(1m) allows more complex policy setups, but these have already been documented in the manual page.[30]

7. Load the IPSec policy into the kernel:

```
# ipsecconf -a ipsecinit.conf
```

Now we are all set. If there were no typos in the shared keys or the Security Parameter Indices, all TCP traffic between B and C is protected with IPSec. As usual, the easiest way to find out is to launch Telnet and monitor the connection with tcpdump (FreeBSD) or snoop (Solaris). If everything went well, the connection attempt from B to C will look like this:

```
# telnet fe80::280:c7ff:fe54:434d
Trying fe80::280:c7ff:fe54:434d...
Connected to fe80::280:c7ff:fe54:434d.
Escape character is '^]'.

FreeBSD/i386 (hostb) (ttyp2)

login:
```

The output from tcpdump should show something like this for the first two packets:

```
fe80::203:baff:fe06:1466 > fe80::280:c7ff:fe54:434d: ESP(spi=0x0000e63f,seq=0x42)
0x0000     6000     0000     003c     323c     fe80     0000     0000     0000     `....<2<........
0x0010     0203     baff     fe06     1466     fe80     0000     0000     0000     .......f........
0x0020     0280     c7ff     fe54     434d     0000     e63f     0000     0042     .....TCM...?...B
0x0030     4e9c     e2dd     3a98     76d4     5eca     1717     3b1c     9538     N...:.v.^...;..8
0x0040     2b0f     5f74     0056     71f7     34f5     1923     a694     68f6     +._t.Vq.4..#..h.
0x0050     d7f0     1c20     3543     ee5e     8bdc     8fe6     09cb     7ab1      ....5C.^......z.
0x0060     9b7a     546f     5cbc     b702                                        .zTo\...
fe80::280:c7ff:fe54:434d > fe80::203:baff:fe06:1466: ESP(spi=0x0000f52a,seq=0x3c) [flowlabel 0x3baff]
0x0000     6003     baff     003c     3240     fe80     0000     0000     0000     `....<2@........
0x0010     0280     c7ff     fe54     434d     fe80     0000     0000     0000     .....TCM........
0x0020     0203     baff     fe06     1466     0000     f52a     0000     003c     .......f...*...<
0x0030     1db8     733e     f397     3c1f     7fef     083d     39d8     eda6     ..s>..<....=9...
0x0040     1b3d     d8a2     1b3e     c8d5     b149     e432     4e89     c31d      .=...>...I.2N...
0x0050     bdb4     c7c5     101b     bafb     19a5     cf0f     d358     e354     ............X.T
0x0060     1753     0af8                                                        .S..
```

When we used tcpdump earlier to monitor AH packets, it was able to disassemble the AH header, peek into the TCP packet, and show us the sequence numbers and other TCP parameters. However, since ESP completely hides the upper-layer payload, we would not know anything about the upper-layer protocol if we hadn't configured the IPSec policy by ourselves. In other words, we can only see that fe80::203:baff:fe06:1466 sent a packet to fe80::280:c7ff:fe54:434d and the receiving host seemed to acknowledge it. It should be no surprise that a packet like this will make a firewall and the (network) intrusion detection very angry.

IPSec is a very powerful network security mechanism to protect applications from network eavesdropping or interception. This discussion has been in the context of IPv6, but the main user community for IPSec seems to be in IPv4, at least for the time being. Network and enterprise management is likely to be the primary beneficiary of IPSec, since management applications have had little or no security capabilities. The list of dangerously insecure million-dollar enterprise systems seems endless and any responsible enterprise management project should absolutely take a close look at the transport mode and assess whether it could be used to improve the security of the management system.

As with AH earlier, the primary security issues with the setup described above are the static Security Parameter Indices and static shared keys. Both SPI and the key should be random values that can be disposed of at the end of the session. History shows repeatedly it is a bad practice to store cleartext secrets in configuration files, even if the files are only readable by the super-user. One can decrease the risk by issuing unique keys for each security association (e.g., four keys for a bi-directional security association), but clearly this creates a key distribution problem.

Conclusion

IPv6 has come a long way from the drafts to the current implementations, but there is still a lot of work to be done. Although IPv6 has been around in one form or another for a long time, it may take a while before IPv6 will be mainstream. This is underscored by the facts that one must run Solaris 9 to get IPSec to work with IPv6, or that only Windows XP users can run Microsoft's IPv6 stack. One must also have BIND 9 to be able to use the new A6 and DNAME resource records, which all means that it will take years before bleeding-edge technology becomes commodity.

Yet IPv6 is a great protocol. When I set up my first IPv6 nodes I had to ask myself several times why I had waited so long. The addressing plans with link-local, site-local, and aggregateable global unicast addresses and stateless address auto-configuration made a lot more sense than requesting that an IP address become eligible in order to get connected to the network.

On the other hand, a new network protocol isn't any good if the only applications for it are ports of a 20-year-old virtual terminal or a file transfer program. Luckily, the situation is not really that bad for IPv6. Obviously, Telnet and FTP were the first IPv6 applications since, they have been a part of TCP/IP for ages. There are IPv6 ports available for open source applications at *ftp.kame.net* and the FreeBSD Ports Collection, for example, shows an ever-growing list of IPv6-enabled applications.

IPv6 is a successor of IPv4 in the sense that it is not very difficult to get IPv4 applications to speak IPv6. There are socket scrubbers available to identify IPv4-specific code and suggest changes to make them IPv6-ready. Consequently, there are few IPv6-only applications, and the convention seems to be to simply make the applications listen to IPv4 and IPv6 sockets simultaneously. This approach does indeed give the users more flexibility in deciding their migration path.

Even if IPv6 networks are hard to find, the developers could start using IPv6 immediately. Since the startup cost to enable IPv6 is minimal – the code is in the operating system already and enabling it requires two steps – it is actually the new interprocess communication mechanism. You don't even need a router to run IPv6 on the local LAN, since the link-local addresses are available to everyone anyway. IPv6 is not a unique protocol in the sense that many IPv6 features are available for IPv4 as well, starting with IPSec. Yet IPv6 does not have to carry the history of IPv4; the many good ideas and concepts arising from IPv4 have been imported to IPv6, and now it is time to move on.

References

[1] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," RFC-2460, December 1998. Available from *http://www.ietf.org/rfc/rfc2460.txt*.

[2] J. Reynolds and J. Postel, "Assigned Numbers," RFC-1700, October 1994. Available from *http://www.ietf.org/rfc/rfc1700.txt* and *http://www.iana.org/*.

[3] R. Hinden and S. Deering, "IP Version 6 Addressing Architecture," RFC-2373, July 1998. Available from *http://www.ietf.org/rfc/rfc2373.txt*.

[4] IPv6 Address Oracle, Indiana University, Advanced Network Management Laboratory. Available from *http://steinbeck.ucs.indiana.edu:47401/*.

[5] C. Huitema, *IPv6: The New Internet Protocol*, 2d ed., Prentice-Hall, 1998.

[6] IEEE, "Guidelines for 64-bit Global Identifier (EUI-64™) Registration Authority," May 2001. Available from *http://standards.ieee.org/regauth/oui/tutorials/EUI64.html*.

[7] D. Plummer, "An Ethernet Address Resolution Protocol," RFC-826, November 1982. Available from *http://www.ietf.org/rfc/rfc0826.txt*.

[8] IANA, "Special-Use IPv4 Addresses," IETF Network Working Group, August 2002. Available from *http://www.ietf.org/internet-drafts/draft-iana-special-ipv4-05.txt*.

[9] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear, "Address Allocation for Private Internets," RFC-1918, February 1996. Available from *http://www.ietf.org/rfc/rfc1918.txt*.

[10] J. Itoh, "Overview of KAME project," Work-in-progress report, USENIX 1998 Annual Conference, New Orleans, 1998. Available from *http://www.kame.net/project-overview.html*.

[11] R. Hinden, M. O'Dell, and S. Deering, "An IPv6 Aggregatable Global Unicast Address Format," RFC-2374, July 1998. Available from *http://www.ietf.org/rfc/rfc2373.txt*.

[12] R. Hinden, S. Deering, R. Fink, and T. Hain, "Initial IPv6 Sub-TLA ID Assignments," RFC-2928, September 2000. Available from *http://www.ietf.org/rfc/rfc2928.txt*.

[13] Global IPv6 Allocations Made by the Regional Internet Registries, RIPE. Available from *http://www.ripe.net/ipv6/ipv6allocs.html*.

[14] B. Carpenter, K. Moore, and B. Fink, "Connecting IPv6 Routing Domains over the IPv4 Internet," *Cisco Internet Protocol Journal*, vol. 3, no. 1, March 2000. Available from *http://www.ieng.com/warp/public/759/ipj_3-1/ipj_3-1_routing.html*.

[15] B. Carpenter and K. Moore, "Connection of IPv6 Domains via IPv4 Clouds," RFC-3056, February 2001. Available from *http://www.ietf.org/rfc/rfc3056.txt*.

[16] S. Deering, "Host Extensions for IP Multicasting," RFC-1112, August 1989. Available from *http://www.ietf.org/rfc/rfc1112.txt*.

[17] S. Thomson and T. Narten, "IPv6 Stateless Address Autoconfiguration," RFC-2462, December 1998. Available from *http://www.ietf.org/rfc/rfc2462.txt*.

[18] Sun Microsystems, "Administering IPv6," Solaris System Administration Guide: IP Services, part no. 806-4075–10, pages 303-322, May 2002. Available from *http://docs.sun.com/*.

[19] R. Zilbauer, "Configuring IPv6 on Solaris 8," Native6Group, 2002. Available from *http://www.native6group.com/docs/Configuring%20IPv6%20on%20Solaris8.pdf*.

[20] Sun Microsystems, "ipnodes – Local Database Associating Names of Nodes with IP Addresses," Solaris 9 Reference Manual Collection, Section 4: File Formats, October 1999. Available from *http://docs.sun/com/*.

[21] B. Bucklin and Y. Sekiya, "IPv6 DNS Setup Information," January 31, 2000. Available from *http://www.isi.edu/~bmanning/v6DNS.html*.

[22] K. Harrenstien, M. Stahl, and E. Feinler, "DoD Internet Host Table Specification," RFC-952, October 1985. Available from *http://www.ietf.org/rfc/rfc0952.txt*.

[23] S. Thomson and C. Huitema, "DNS Extensions to Support IP Version 6," RFC-1886, December 1995. Available from *http://www.ietf.org/rfc/rfc1886.txt*.

[24] M. Crawford and C. Huitema, "DNS Extensions to Support IPv6 Address Aggregation and Renumbering," RFC-2874, July 2000. Available from *http://www.ietf.org/rfc/rfc2874.txt*.

[25] Internet Software Consortium, "IPv6 Support in BIND 9," BIND 9 Administrator Reference Manual, pages 36–39, 2001. Available from *http://www.nominum.com/resources/documentation/Bv9ARM.pdf*.

[26] ISO, "OSI Security Model, Part 1: Security Framework," ISO 10181-1.

[27] KAME Project, "setkey – Manually Manipulate the Ipsec SA/SP Database," FreeBSD System Manager's Manual, Chapter 8, FreeBSD, 2000. Available from *http://www.freebsd.org/cgi/man.cgi?query=setkey&apropos=0&sektion=8&manpath=FreeBSD+4.6-RELEASE&format=html*.

[28] Sun Microsystems, "in.iked – Daemon for the Internet Key Exchange (IKE)," Solaris 9 Reference Manual Collection, Section 1m: System Administration Commands, Sun Microsystems, Inc., February 2002. Available from *http://docs.sun/com/*.

[29] KAME Project, "racoon – IKE (ISAKMP/Oakley) key management daemon," available in the KAME distribution from *http://www.kame.net/*.

[30] Sun Microsystems, "ipsecconf – configure system wide IPSec policy," Solaris 9 Reference Manual Collection, Section 1m: System Administration Commands, February 2002. Available from *http://docs.sun/com/*.