# Compaq TCP/IP Services for OpenVMS

# Guide to IPv6

Order Number: AA–RNJ3A–TE

**January 2001**

This manual describes the Compaq TCP/IP Services for OpenVMS IPv6 features and how to install and configure IPv6 on your system. In addition, this manual describes changes in the socket application programming interface (API) and how to port your applications to run in an IPv6 environment.

| | |
|---|---|
| **Revision Information:** | This is a new manual. |
| **Software Version:** | Compaq TCP/IP Services for OpenVMS Version 5.1 |
| **Operating Systems:** | OpenVMS Alpha Versions 7.1, 7.2-1, OpenVMS VAX Versions 7.1, 7.2 |

**Compaq Computer Corporation**
**Houston, Texas**

ZK6645

This document is available on CD-ROM.

This document was prepared using DECdocument, Version 3.3-1b.

# Contents

## 7 Porting Applications



## A Supported IPv6 RFCs

## B IPv6 Extensions to Management Commands and IPv6 Processes

## C Deprecated Library Functions



## Examples



## Figures

## Tables

# Preface

The Compaq TCP/IP Services for OpenVMS product is the Compaq implementation of the TCP/IP networking protocol suite and internet services for OpenVMS Alpha and OpenVMS VAX systems.

TCP/IP Services provides a comprehensive suite of functions and applications that support industry-standard protocols for heterogeneous network communications and resource sharing.

This manual describes IPv6 features included in this version of TCP/IP Services. The manual covers installing and configuring your system for IPv6, changes to the socket API, and how to port your applications to run in an IPv6 environment.

## Intended Audience

This manual is for experienced OpenVMS and UNIX system managers and assumes a working knowledge of OpenVMS system management, TCP/IP networking, and TCP/IP terminology.

The manual is also for programmers who want to rewrite their applications for the IPv6 environment.

## Document Structure

This manual contains the following chapters and appendixes:

| | |
|---|---|
| Chapter 1 | Describes the IPv6 environment, the roles of systems in this environment, the types and function of the different IPv6 addresses, and how to connect to the 6bone network. |
| Chapter 2 | Describes how to configure the IPv6 software. |
| Chapter 3 | Provides guidelines for running BIND in an IPv6 environment. |
| Chapter 4 | Describes the resources for monitoring IPv6 network traffic. |
| Chapter 5 | Describes how to solve IPv6 problems. |
| Chapter 6 | Describes the IPv6 additions to the socket API. |
| Chapter 7 | Describes how to port applications. |
| Appendix A | Describes the supported IPv6 RFCs. |
| Appendix B | Lists commands and processes supported in this version. |
| Appendix C | Describes deprecated functions that have been replaced by new ones. |

## Related Documents

Table 1 lists the documents available with this version of TCP/IP Services.

**Table 1 TCP/IP Services Documentation**

| Manual | Contents |
|---|---|
| *DIGITAL TCP/IP Services for OpenVMS Concepts and Planning* | This manual provides conceptual information about networking and the TCP/IP protocol including a description of the Compaq implementation of the Berkeley Internet Name Domain (BIND) service and the Network File System (NFS). It outlines general planning issues to consider before configuring your system to use the TCP/IP Services software.<br><br>This manual also describes the manuals in the documentation set, provides a glossary of terms and acronyms for the TCP/IP Services software product, and documents how to contact the InterNIC Registration Service to register domains and access Requests for Comments (RFCs). |
| *Compaq TCP/IP Services for OpenVMS Release Notes* | The release notes provide version-specific information that supersedes the information in the documentation set. The features, restrictions, and corrections in this version of the software are described in the release notes. Always read the release notes before installing the software. |
| *Compaq TCP/IP Services for OpenVMS Installation and Configuration* | This manual explains how to install and configure the TCP/IP Services product. |
| *DIGITAL TCP/IP Services for OpenVMS User's Guide* | This manual describes how to use the applications available with TCP/IP Services such as remote file operations, email, TELNET, TN3270, and network printing. This manual explains how to use these services to communicate with systems on private internets or on the worldwide Internet. |
| *Compaq TCP/IP Services for OpenVMS Management* | This manual describes how to configure and manage the TCP/IP Services product.<br><br>Use this manual with the *Compaq TCP/IP Services for OpenVMS Management Command Reference* manual. |
| *Compaq TCP/IP Services for OpenVMS Management Command Reference* | This manual describes the TCP/IP Services management commands.<br><br>Use this manual with the *Compaq TCP/IP Services for OpenVMS Management* manual. |
| *Compaq TCP/IP Services for OpenVMS Management Command Quick Reference Card* | This reference card lists the TCP/IP management commands by component and describes the purpose of each command. |
| *Compaq TCP/IP Services for OpenVMS UNIX Command Reference Card* | This reference card contains information about commonly performed network management tasks and their corresponding TCP/IP management and Compaq *Tru64* UNIX command formats. |
| *DIGITAL TCP/IP Services for OpenVMS ONC RPC Programming* | This manual presents an overview of high-level programming using open network computing remote procedure calls (ONC RPCs). This manual also describes the RPC programming interface and how to use the RPCGEN protocol compiler to create applications. |
| *Compaq TCP/IP Services for OpenVMS Sockets API and System Services Programming* | This manual describes how to use the Sockets API and OpenVMS system services to develop network applications. |

(continued on next page)

**Table 1 (Cont.) TCP/IP Services Documentation**

| Manual | Contents |
|---|---|
| *Compaq TCP/IP Services for OpenVMS SNMP Programming and Reference* | This manual describes the Simple Network Management Protocol (SNMP) and the SNMP application programming interface (eSNMP). It describes the subagents provided with TCP/IP Services, utilities provided for managing subagents, and how to build your own subagents. |
| *Compaq TCP/IP Services for OpenVMS Tuning and Troubleshooting* | This manual provides information about how to isolate the causes of network problems and how to tune the TCP/IP Services software for the best performance. |
| *Compaq TCP/IP Services for OpenVMS Guide to IPv6* | This manual describes the IPv6 environment, the roles of systems in this environment, the types and function of the different IPv6 addresses, and how to configure TCP/IP Services to access the 6bone network. |

For additional information about Compaq *OpenVMS* products and services, access the Compaq website at the following location:

`http://www.openvms.compaq.com/`

For a comprehensive overview of the TCP/IP protocol suite, you might find the book *Internetworking with TCP/IP: Principles, Protocols, and Architecture*, by Douglas Comer, useful.

## Reader's Comments

Compaq welcomes your comments on this manual. Please send comments to either of the following addresses:

| | |
|---|---|
| Internet | **openvmsdoc@compaq.com** |
| Mail | Compaq Computer Corporation<br>OSSG Documentation Group, ZKO3-4/U08<br>110 Spit Brook Rd.<br>Nashua, NH 03062-2698 |

## How to Order Additional Documentation

Visit the following World Wide Web address for information about how to order additional documentation:

`http://www.openvms.compaq.com/`

If you need help deciding which documentation best meets your needs, call 800-282-6672.

## Conventions

The name TCP/IP Services means both:

- Compaq TCP/IP Services for OpenVMS Alpha
- Compaq TCP/IP Services for OpenVMS VAX

The name UNIX refers to the Compaq *Tru64* UNIX operating system.

The following conventions are used in this manual. In addition, please note that all IP addresses are fictitious.

| | |
|---|---|
| Ctrl/*x* | A sequence such as Ctrl/*x* indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button. |
| PF1 *x* | A sequence such as PF1 *x* indicates that you must first press and release the key labeled PF1 and then press and release another key or a pointing device button. |
| Return | In examples, a key name enclosed in a box indicates that you press a key on the keyboard. (In text, a key name is not enclosed in a box.)<br>In the HTML version of this document, this convention appears as brackets, rather than a box. |
| . . . | A horizontal ellipsis in examples indicates one of the following possibilities:<br>• Additional optional arguments in a statement have been omitted.<br>• The preceding item or items can be repeated one or more times.<br>• Additional parameters, values, or other information can be entered. |
| .<br>.<br>. | A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed. |
| ( ) | In command format descriptions, parentheses indicate that you must enclose choices in parentheses if you specify more than one. |
| [ ] | In command format descriptions, brackets indicate optional choices. You can choose one or more items or no items. Do not type the brackets on the command line. However, you must include the brackets in the syntax for OpenVMS directory specifications and for a substring specification in an assignment statement. |
| \| | In command format descriptions, vertical bars separate choices within brackets or braces. Within brackets, the choices are optional; within braces, at least one choice is required. Do not type the vertical bars on the command line. |
| { } | In command format descriptions, braces indicate required choices; you must choose at least one of the items listed. Do not type the braces on the command line. |
| **bold text** | This typeface represents the introduction of a new term. It also represents the name of an argument, an attribute, or a reason. |
| *italic text* | Italic text indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error *number*), in command lines (/PRODUCER=*name*), and in command parameters in text (where *dd* represents the predefined code for the device type). |
| UPPERCASE TEXT | Uppercase text indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege. |

| | |
|---|---|
| `Monospace text` | Monospace type indicates code examples and interactive screen displays.<br><br>This typeface indicates UNIX system output or user input, commands, options, files, directories, utilities, hosts, and users.<br><br>In the C programming language, this typeface identifies the following elements: keywords, the names of independently compiled external functions and files, syntax summaries, and references to variables or identifiers introduced in an example. |
| - | A hyphen at the end of a command format description, command line, or code line indicates that the command or statement continues on the following line. |
| numbers | All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated. |

# 1

# What Is IPv6?

In the early 1990s, members of the Internet community realized that the address space and certain aspects of the current TCP/IP architecture were not capable of sustaining the explosive growth of the Internet. The problems included the exhaustion of the Internet address space, the size of routing tables, and requirements for new technology features.

The Internet Engineering Task Force (IETF) made several efforts to study and improve the use of the 32-bit Internet Protocol (IPv4) addresses. They also tackled the longer-term goal of identifying and replacing protocols and services that would limit growth.

These efforts identified the 32-bit addressing architecture of IPv4 as the principal problem affecting router overhead and network administration. In addition, IPv4 addresses were often unevenly allocated in blocks that were too large or too small; therefore, these addresses were difficult to change within any existing network.

In July 1994, the Internet Protocol Next Generation (IPng) directorate announced Internet Protocol Version 6 (IPv6) as the replacement network layer protocol, and IETF working groups began to build specifications. (See RFC 1752, *The Recommendation for the IP Next Generation Protocol*, for additional information about the IPv6 protocol selection process.)

IPv6 is both a completely new network layer protocol and a major revision of the Internet architecture. As such, it builds upon and incorporates experience gained with IPv4. This chapter describes the following:

- Terminology
- IPv6 addressing
- IPv6 environment
- IPv6 configuration
- Postconfiguration tasks

## 1.1 Terminology

The following terms are used in this chapter:

- **Node**

  Any system that uses the IPv6 protocol to communicate.

- **Router**

  A node that forwards IPv6 packets addressed to other nodes. These systems typically have more than one network interface installed and configured.

- **Host**

  Any system that is not a router.

- **Link**

  A medium or facility over which nodes communicate with each other at the Link layer. Examples include Ethernet, FDDI links, or internet layer tunnels.

- **interface**

  A node's attachment to a link. An interface is usually assigned an IPv6 address or addresses.

## 1.2 Introduction to IPv6 Addresses

The most noticeable feature of IPv6 is the address itself. The address size is increased from 32 bits to 128 bits. The following sections describe the components of the IPV6 address.

### 1.2.1 Address Text Representation

Use the following syntax to represent IPv6 addresses as text strings:

```
x:x:x:x:x:x:x:x
```

The *x* is a hexadecimal value of a 16-bit piece of the address. For example, the following addresses are IPv6 addresses:

```
FEDC:BA98:7654:3210:FEDC:BA98:7654:3210
1070:0:0:0:0:800:200C:417B
```

IPv6 addresses can contain long strings of zero (0) bits. To make it easier to write these addresses, you can use a double colon (::) once in an address to represent one or more 16-bit groups of zeros. For example, you can compress the second IPv6 address example in the following way:

```
1070::800:200C:417B
```

Alternately, you can use the following syntax to represent IPv6 addresses in an environment of both IPv4 and IPv6 nodes:

```
x:x:x:x:x:x:d.d.d.d
```

In this case, *x* is a hexadecimal value of a 16-bit piece of the address (six high-order pieces) and *d* is a decimal value of an 8-bit piece of address (four low-order pieces) in standard, dotted-quad IPv4 form. For example, the following are IPv6 addresses:

```
0:0:0:0:0:0:13.1.68.3
0:0:0:0:0:FFFF:129.144.52.38
```

When compressed, these addresses are as follows:

```
::13.1.68.3
::FFFF:129.144.52.38
```

Like IPv4 address prefixes, IPv6 address prefixes are represented using the Classless Inter-Domain Routing (CIDR) notation. This notation has the following format:

```
ipv6-address/prefix-length
```

For example, you can represent the 60-bit hexadecimal prefix 12AB00000000CD3 in any of the following ways:

```
12AB:0000:0000:CD30:0000:0000:0000:0000/60
12AB::CD30:0:0:0:0/60
12AB:0:0:CD30::/60
```

## 1.2.2 Types of Addresses

There are three types of IPv6 addresses:

- Unicast
- Anycast
- Multicast

**Note**

Unlike IPv4, IPv6 does not define a broadcast address. To get the function of a broadcast address, use a multicast address. (See Section 1.2.2.3.)

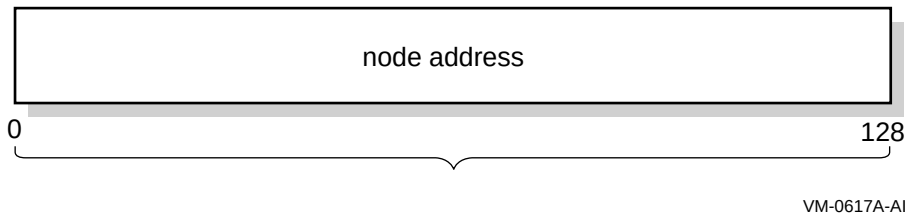
The following sections describe the unicast, anycast, and multicast address types.

### 1.2.2.1 Unicast Addresses

A unicast address is an identifier for an interface. Packets sent to a unicast address are delivered to the node containing the interface that is identified by the address.

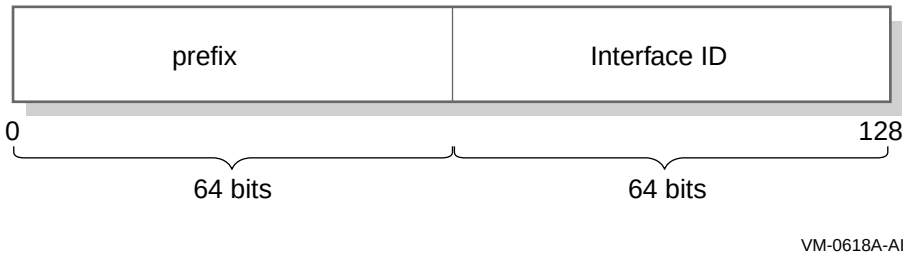
Figure 1–1 shows the format of unicast addresses.

**Figure 1–1 Unicast Addresses**




This address typically consists of a 64-bit prefix followed by a 64-bit interface ID, as shown in Figure 1–2.

**Figure 1–2 64-Bit Prefix Plus 64-Bit Interface ID**




An interface ID identifies an interface on a link. The interface ID is required to be unique on a link, but it may also be unique over a broader scope. In many cases, the interface ID is derived from its Link layer address. The same interface ID can be used on multiple interfaces on a single node.

The following list describes commonly used unicast addresses and their values:

- Unspecified address

  Indicates the absence of an address and is never assigned to an interface. The unspecified address has the following value:

  ```
  0:0:0:0:0:0:0:0 (normal form)
  :: (compressed form)
  ```

- Loopback address

  Used by a node to send IP datagrams to itself and is typically assigned to the loopback interface.

  The IPv6 loopback address has the following value:

  ```
  0:0:0:0:0:0:0:1 (normal form)
  ::1 (compressed form)
  ```

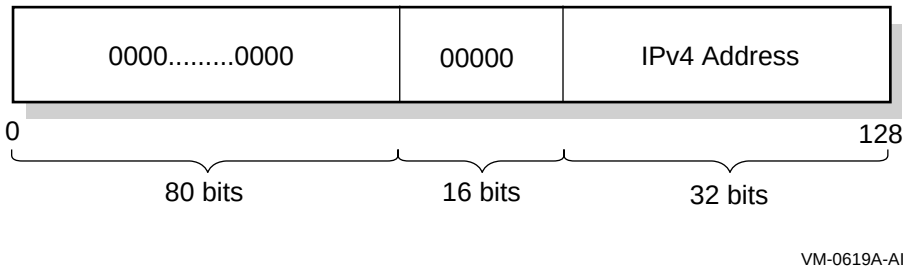
- IPv6 addresses with embedded IPv4 addresses

  Used in mixed IPv4 and IPv6 environments and can be either of the following:

  – IPv4-compatible IPv6 address

    Used by IPv6 nodes to tunnel IPv6 packets across an IPv4 routing infrastructure. The IPv4 address is carried in the low-order 32 bits. Figure 1–3 shows the format of the IPV4-compatible IPV6 address.

**Figure 1–3 IPv4-Compatible IPv6 Address**



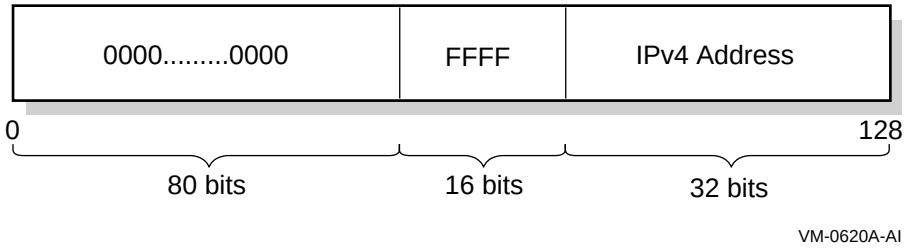

**Note**

Do not use IPv4-compatible IPv6 addresses in DNS or in TCPIP$ETC:IPNODES.

– IPv4-mapped IPv6 address

  Used to represent an IPv4 address and to identify nodes that do not support IPv6. This address is not used in an IPv6 packet. Figure 1–4 shows the format of the IPv4-mapped IPv6 address.

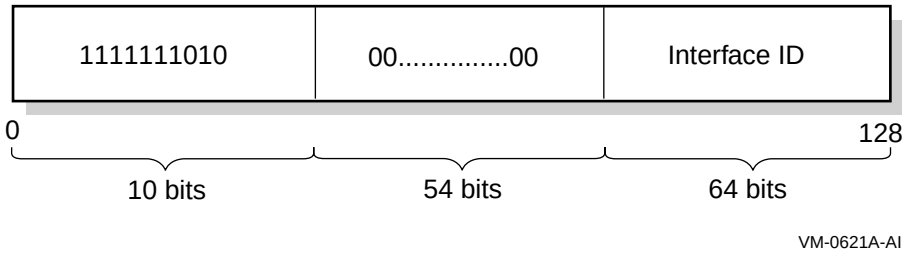
**Figure 1–4 IPv4-Mapped IPv6 Address**




- Local-use IPv6 unicast addresses can be either of the following:
  – Link-local

    Used for addressing on a single link when performing address autoconfiguration or neighbor discovery or when no routers are present. Figure 1–5 shows the format of the link-local address.

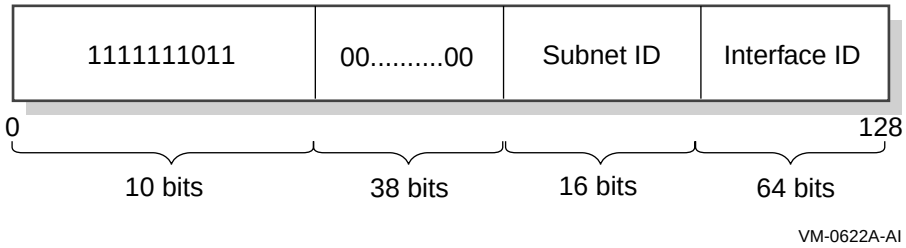
**Figure 1–5 IPv6 Link-Local Unicast Address**




– Site-local

Used for sites or organizations that are not connected to the global Internet. Figure 1–6 shows the format of the site-local address.

**Figure 1–6 IPv6 Site-Local Unicast Address**




If you plan to use site-local addresses, be aware of the following guidelines:

* Do not connect a single node to multiple sites.
* Do not use site-local addresses in the global DNS (the addresses should not be visible outside the site).
* Do not advertise or propagate routes containing site-local prefixes outside the site.

#### 1.2.2.2 Anycast Address

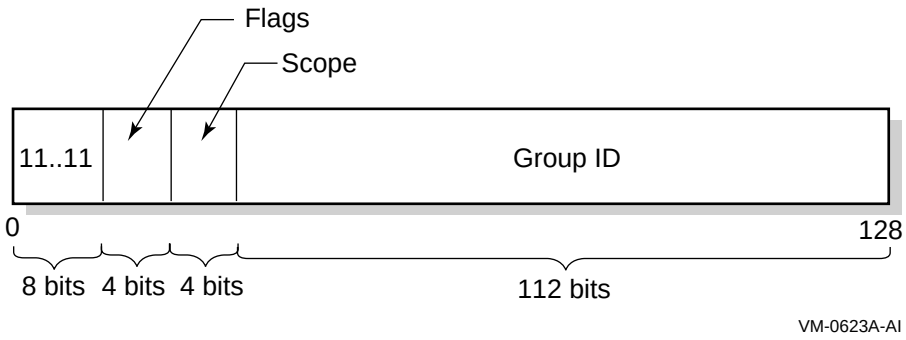
An anycast address is an identifier for a set of interfaces typically belonging to different nodes. Packets sent to an anycast address are delivered to one of the interfaces identified as the "nearest" address, according to the routing protocol's measure of distance.

The format for anycast addresses is identical to the unicast format.

#### 1.2.2.3 Multicast Address

A multicast address is an identifier for a group of nodes. It is similar to an IPv4 multicast address. Figure 1–7 shows the format for multicast addresses.

**Figure 1–7 IPv6 Multicast Address**




In the multicast address format, the fields have the following definitions:

11..11 — Identifies the address as multicast.

Flags — Can be either of the following values:

- 0000, which indicates a permanently assigned (well-known) multicast address,
- 0001, which indicates a nonpermanently assigned (transient) multicast address.

Scope — Indicates the scope of the multicast group. The following table lists the scope values:

| Value (Hex) | Scope |
|---|---|
| 1 | Node-local |
| 2 | Link-local |
| 5 | Site-local |
| 8 | Organization-local |
| E | Global |

Group ID — Identifies the multicast group within the specified scope.

Table 1–1 lists some well-known multicast addresses.

**Table 1–1 Well-Known Multicast Addresses**

| Multicast Address | Meaning |
|---|---|
| FF02::1 | All nodes (link-local) |
| FF02::2 | All routers (link-local) |
| FF02::9 | All RIPng routers (link-local) |

### 1.2.3 Address Prefixes

Each IPv6 address has a unique pattern of leading bits that indicates its address type. These leading bits are called the **format prefix**. Table 1–2 lists some IPv6 address types and their prefixes.

**Table 1–2 IPv6 Address Types and Prefixes**

| Address Type | Prefix |
|---|---|
| Aggregatable global unicast | 2000::/3 |
| Link-local | FE80::/10 |
| Site-local | FEC0::/10 |
| Multicast | FF00::/8 |

### 1.2.4 Address Autoconfiguration

The IPv6 address changes have led to the following definitions for configuring addresses:

- Stateless address autoconfiguration
- Dynamic Host Configuration Protocol Version 6 (DHCPv6), which is stateful address autoconfiguration

In the stateless model, nodes learn address prefixes by listening for Router Advertisement packets. Addresses are formed by combining the prefix with a data link-specific interface token, which is typically derived from the data link address of the interface. This model is favored by administrators who do not need tight control over address configuration. See RFC 2462 for more information.

In DHCPv6, hosts may request addresses, configuration information and services from dedicated configuration servers. This model is favored by administrators who want to delegate addresses based on a client/server model. The DHCPv6 Internet Drafts are currently undergoing revision. See the DHCP charter web page for more information:

`www.ietf.org/html.charters/dhc-charter.html`

**Note**

Version 5.1 of Compaq TCP/IP Services for OpenVMS does not support DHCPv6.

In both cases, the resulting addresses have associated lifetimes, and systems must be able to acquire new addresses and release expired addresses. Combined with the ability to register updated address information with Domain Name System (DNS) servers, these mechanisms provide a path towards network renumbering and provide network administrators with control over the use of network addresses without manual intervention on each host on the network.

### 1.2.5 Address Resolution

The Domain Name System (DNS) provides support for mapping names to IP addresses and mapping IP addresses back to their corresponding names. Because of the increased size of the IPv6 address, the DNS has the following new features:

- AAAA resource record type

  This holds IPv6 addresses, encoded in network byte order. The version of BIND shipped with Compaq TCP/IP Services for OpenVMS supports AAAA records.

- AAAA query

  A query for a specified domain name in the Internet class returns all associated AAAA resource records in the response.

- IP6.INT domain for looking up a name for a specified address (address-to-name mapping)

  An IPv6 address is represented in reverse order as a sequence of 4-bit nibbles separated by dots with the suffix .IP6.INT appended. For example, the IPv6 address `4321:0:1:2:3:4:567:89ab` has the following inverse-lookup domain name:

  ```
  b.a.9.8.7.6.5.0.4.0.0.0.3.0.0.0.2.0.0.0.1.0.0.0.0.0.0.0.1.2.3.4.IP6.INT
  ```

  See Chapter 3 for guidelines on configuring BIND in an IPv6 environment.

## 1.3 Address Assignment

IPv6 addresses are now being deployed by the regional registries. See the IANA web page at the following location for more information:

```
http://www.ipv6.org/iana-ann.html
```

In addition, you can contact your Internet Service Provider (ISP) to obtain an IPv6 address.

Because of the need to test various implementations of the IPv6 RFCs, the IETF has defined a temporary IPv6 address allocation scheme. You can assign the addresses in this scheme to hosts and routers for testing IPv6 on the 6bone (a prototype IPv6 implementation that can be used for testing). See the 6bone home page at the following location for more information about 6bone address allocation and assignment:

```
http://www.6bone.net
```

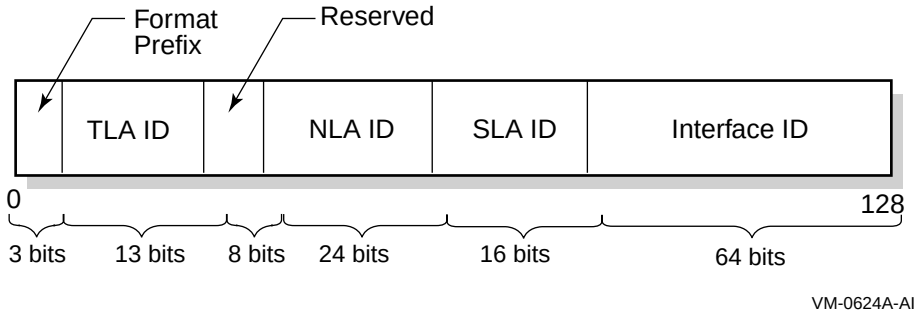
At the present time, the 6bone test addresses are aggregatable global unicast addresses. Contact your 6bone service provider (for example, `gw-6bone@pa.dec.com`) for a 6bone address delegation.

The following sections describe the formats for the aggregatable IPv6 addresses.

### 1.3.1 Aggregatable Global Unicast Addresses

The aggregatable global unicast address format for IPv6 is designed to support current provider-based aggregation and new exchange-based aggregation. Whether a site connects to a provider or to an exchange, the address format enables efficient route aggregation for either type. Figure 1–8 shows the format for an aggregatable global unicast address. (See RFC 2374 for additional information.)

**Figure 1–8 Aggregatable Global Unicast Address Format**



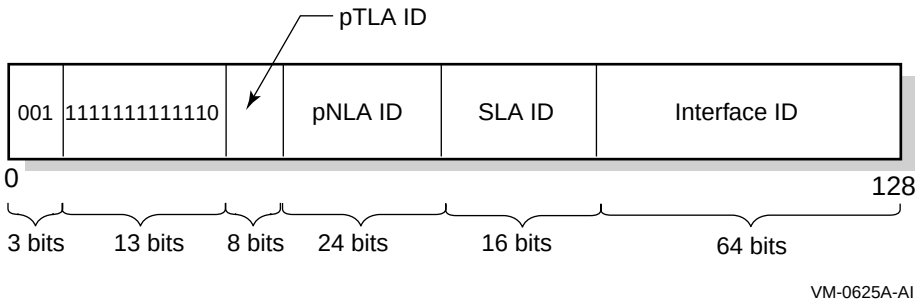

In this address format, the fields have the following definitions:

| | |
|---|---|
| Format Prefix | The format prefix. For aggregatable global unicast addresses, the value for this field is 001. |
| TLA ID | The top-level aggregation identifier. |
| Reserved | Reserved for future use. At present, set to all zeros (0). |
| NLA ID | The next-level aggregation identifier. These are assigned by the TLA ID administrator to create an addressing hierarchy and to identify end-user sites. Each organization assigned a TLA ID is also assigned 24 bits of NLA ID space whose layout and use is the responsibility of the organization. |
| SLA ID | The site-level aggregation identifier. These are used by an individual organization to create its own local addressing hierarchy and to identify subnets. |
| Interface ID | The 64-bit interface identifier of the interface that is connected to the link. |

### 1.3.2 Aggregatable Testing Addresses

Figure 1–9 shows the format for aggregatable global unicast addresses for IPv6 testing. (See RFC 2471 for more information about the proposed testing address allocation plan.)

**Figure 1–9 Aggregatable Testing Address Format**




In this address format, the fields have the following definitions:

| | |
|---|---|
| 001 | The format prefix for aggregatable global unicast addresses. |
| 1111111111110 | The 6bone top-level aggregation (TLA) identifier, 0x1FFE, which is reserved by the Internet Assigned Numbers Naming Authority (IANA) and is used temporarily for IPv6 testing. |
| pTLA ID | The pseudo top-level aggregation identifier. This is assigned by the pTLA ID administrator to define the top level of aggegation (backbone sites) for the 6bone network. |
| pNLA ID | The pseudo next-level aggregation identifier. This is the ID assigned by the pTLA ID administrator to create an addressing hierarchy and to identify end-user sites on the 6bone network. |
| SLA ID | The site-level aggregation identifier. This is the ID assigned by an organization to create its own local addressing hierarchy and to identify subnets. |
| Interface ID | The 64-bit interface identifier of the interface that is connected to the link. |

For the most current information about pTLA and pNLA assignments, see the 6bone home page at the following location:

```
http://www.6bone.net
```

## 1.4 IPv6 Environment

This section shows some example IPv6 configurations. Select a configuration that most closely matches the environment in which you want to configure IPv6 on your system.

Figure 1–10 shows a simple LAN configuration in which host A and host B communicate using IPv6 with no router.

**Figure 1–10 Host-to-Host Configuration with No Router**

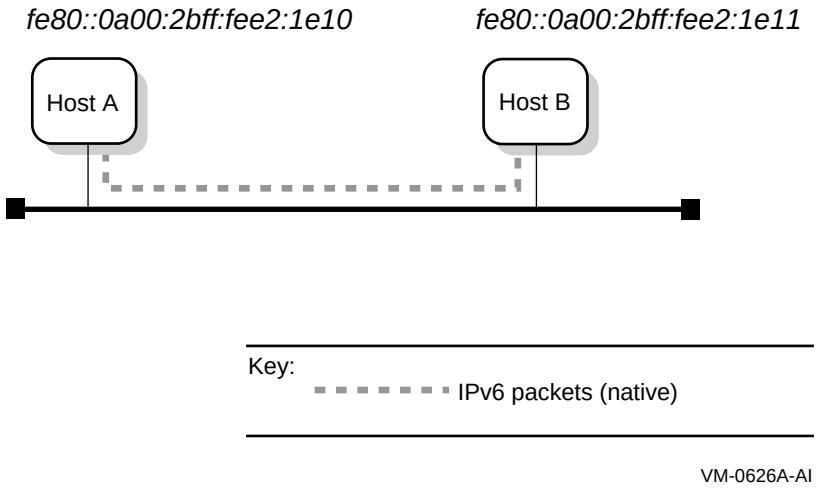



Figure 1–11 shows a simple LAN configuration in which host A, host B, and router A communicate using IPv6. Host A and host B obtain global addresses from router A.

**Figure 1–11 Host-to-Host Configuration with Router**

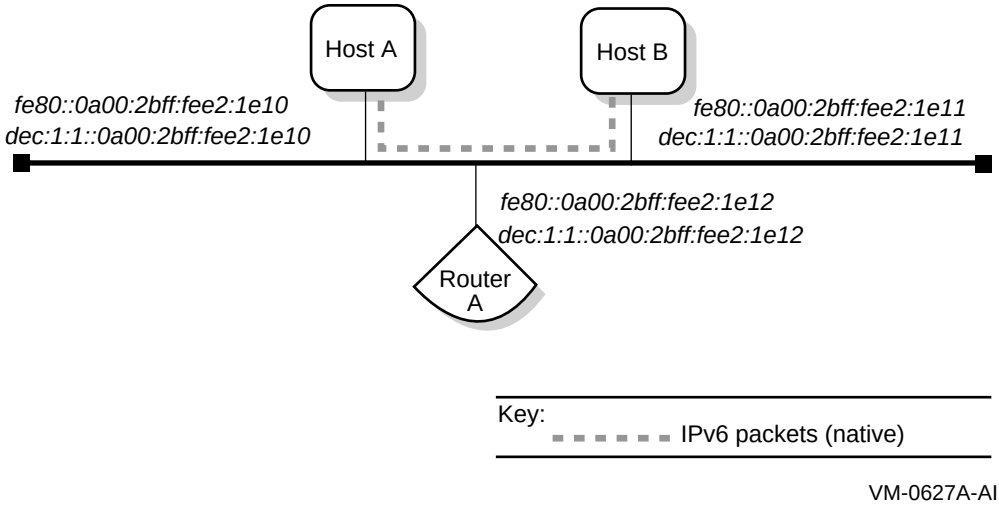



Figure 1–12 shows a configuration in which two IPv6 networks are connected through an IPv6 router (router A).

**Figure 1–12 IPv6 Network to IPv6 Network with Router Configuration**

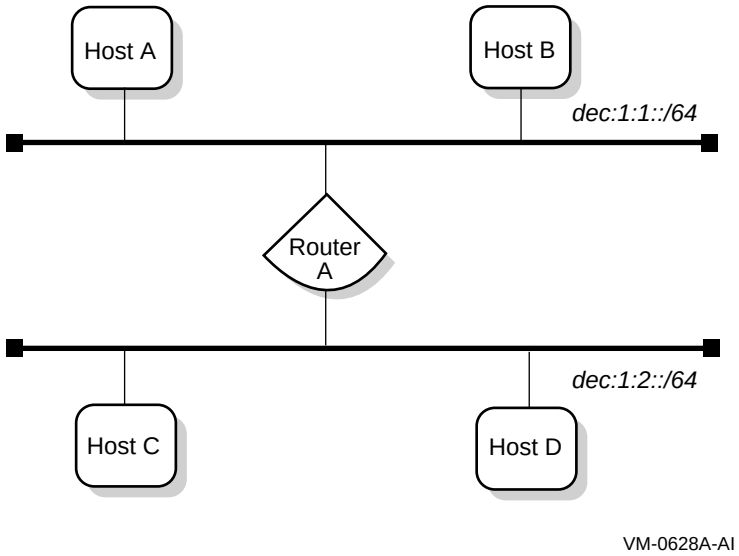



Figure 1–13 shows a configuration in which four IPv6 networks are connected using three routers. The three routers exchange routing information with each other using the RIPng protocol.

**Figure 1–13 Multiple IPv6 Networks and Multiple Routers Configuration**

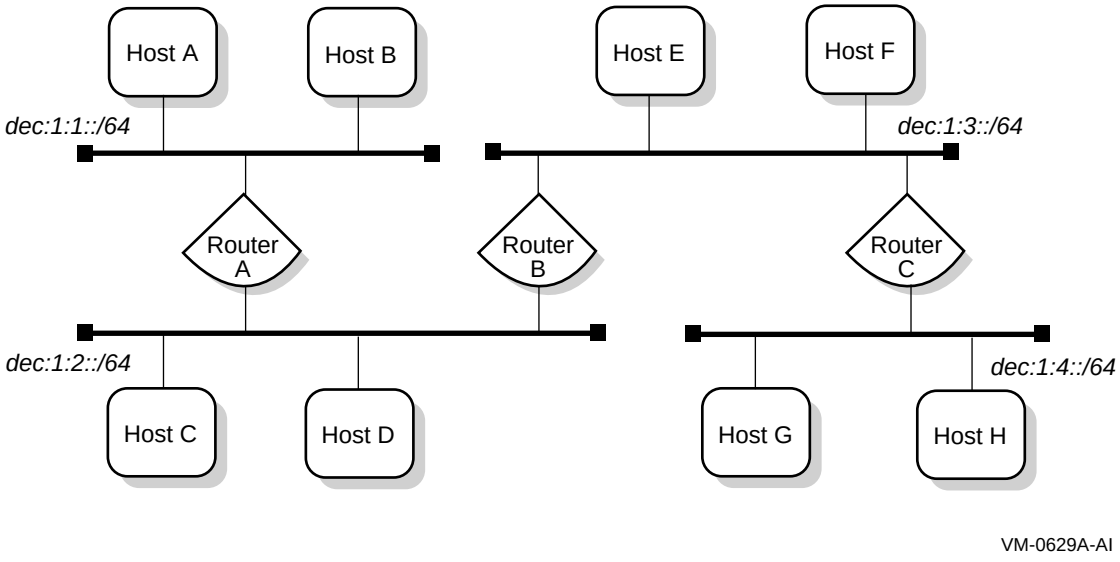



Figure 1–14 shows a configuration in which host A and host B, connected to an IPv4 network, communicate using IPv6 through an IPv4 tunnel.

**Figure 1–14 Host-to-Host Configuration over Tunnel**

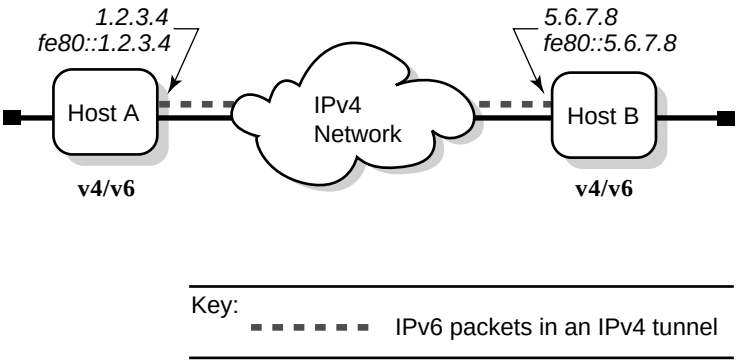



Figure 1–15 shows a configuration in which host X is connected to an IPv4 network. Router A, an IPv6 router, is connected to the same IPv4 network and is also connected to two IPv6 networks. Host X communicates with host B using IPv6 through an IPv4 tunnel between host X and router A.

**Figure 1–15 Host-to-Router Configuration over Tunnel**

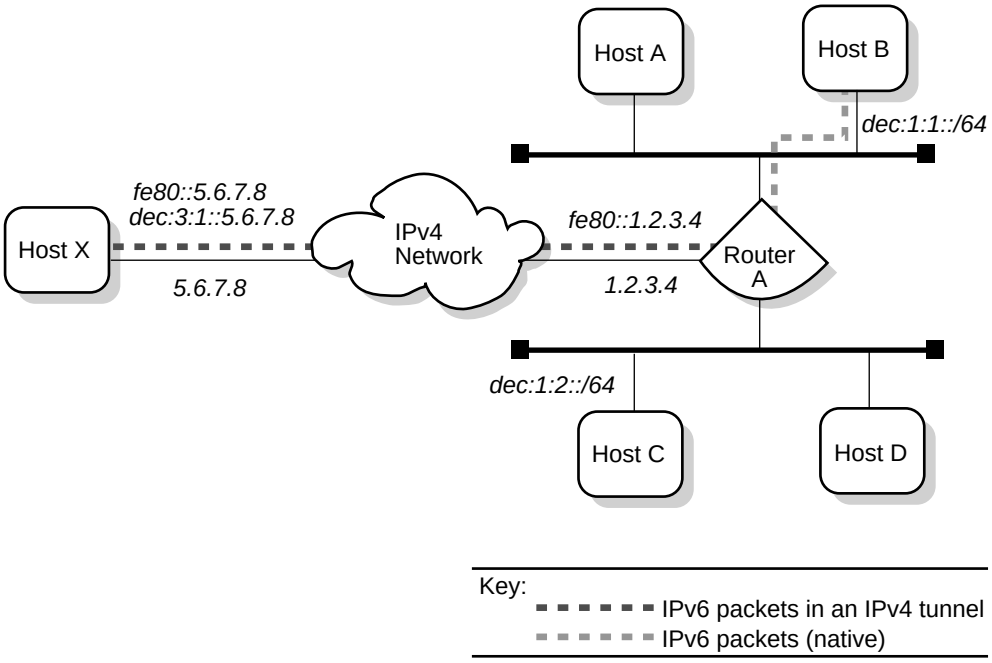
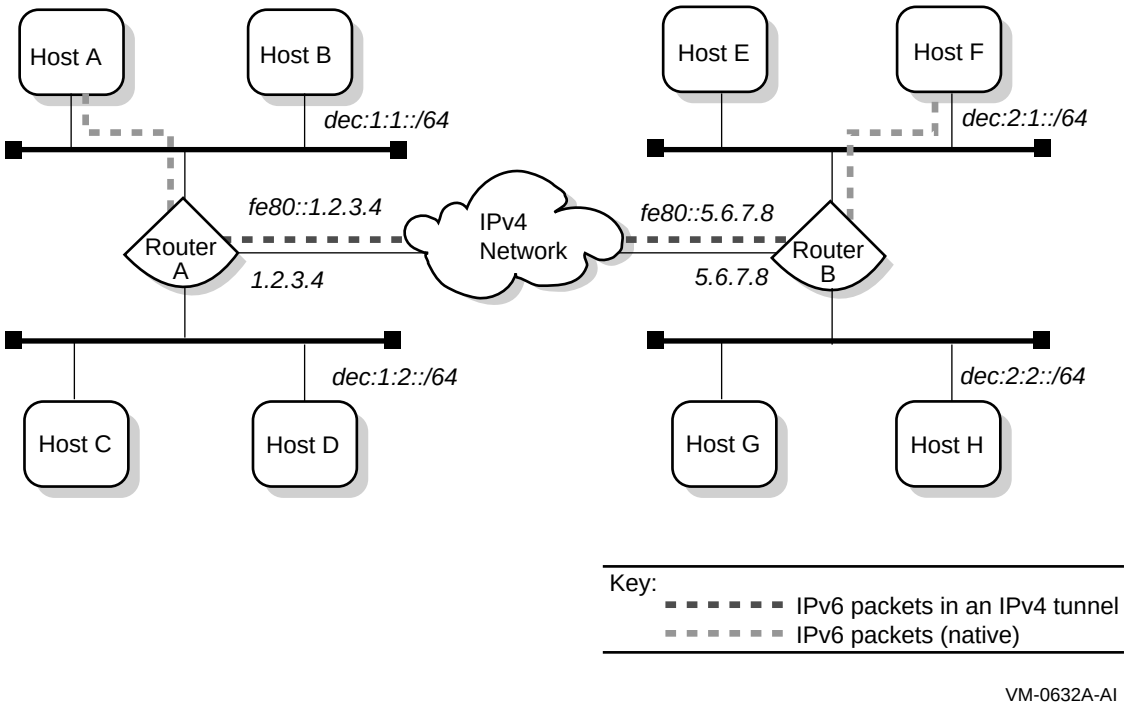



Figure 1–16 shows a configuration in which four IPv6 networks are connected through two routers and an IPv4 network. Host A communicates with host F through an IPv4 tunnel between router A and router B.

**Figure 1–16 IPv6 Network-to-IPv6 Network Configuration over Tunnel**



# 2
# Configuring IPv6

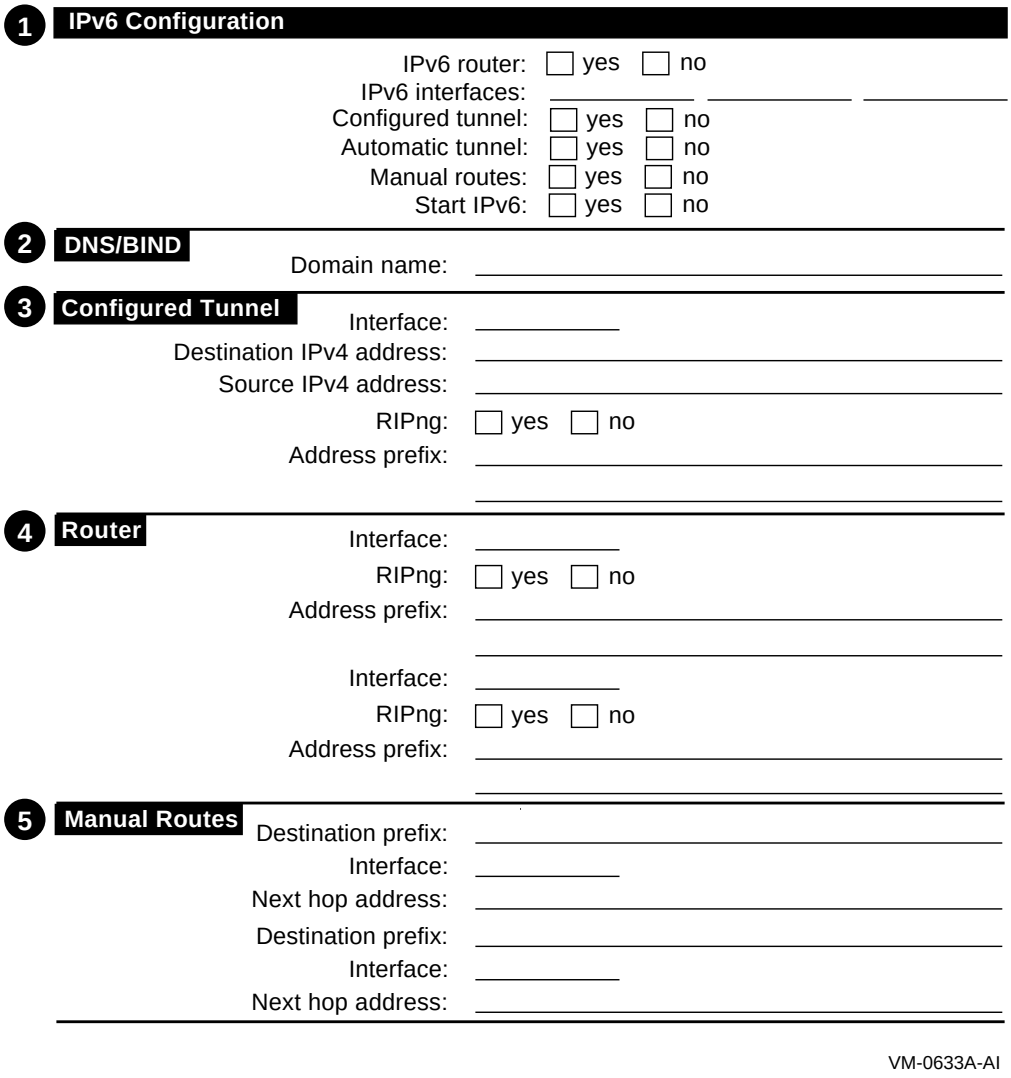
After installimg Compaq TCP/IP Services for OpenVMS Version 5.1, you can configure your system to communicate in an IPv6 network environment by performing the tasks described in the following sections. You can configure your node as either of the following:

- IPv6 host
- IPv6 router

## 2.1 Preparing for Configuration

Before you configure the network software, you must gather information about your system and network environment. The Configuration Worksheet shown in Figure 2–1 can help you assemble this information in an orderly fashion. The following sections describe the information that you need to record on the worksheet.

**Figure 2–1 Configuration Worksheet**




**1** IPv6 Configuration

- IPv6 router

  If you want this system to function as an IPv6 router, check Yes; otherwise, check No. If you check No, the system is configured as an IPv6 host.

  An IPv6 router can advertise address prefixes to all hosts on connected links (for example, a LAN and a configured tunnel) and can forward packets to their destinations. Packets can be forwarded directly on link or over IPv4 tunnels.

- IPv6 interfaces

  Enter the device names of the network interface to the IPv6 network. For example, WE0 and WF0. If you are creating a configured tunnel only on your system, enter None.

- Configured tunnel

  If you want IPv6 to run over a configured IPv4 tunnel, check Yes; otherwise, check No. A configured tunnel has one source and one destination in an IPv4 network. You should use configured tunnels instead of automatic tunnels. You can configure multiple configured tunnels.

- Automatic tunnel

  If you want to configure IPv6 to run over IPv4 automatic tunnels, check Yes; otherwise, check No.

- Manual routes

  If you want to configure manual routes to other systems, check Yes; otherwise, check No.

  On a router, you might want to configure manual routes if one of the following conditions is true:

  – You want a configured tunnel and you are not advertising an address prefix on the tunnel link.

  – You want a configured tunnel and the router at the other end of the tunnel is not running the RIPng protocol.

  – Your system is not running the RIPng protocol.

  On a host, you might want to configure manual routes if you want a configured tunnel to a router and the router is not advertising itself as a default router on the tunnel link.

- Start IPv6

  If you want the IPv6 initialization script executed from the configuration utility, check Yes. If you want the initialization script executed during the next system boot, check No.

**2** DNS/BIND

- Domain name

  The fully qualified domain name for your node. This consists of the host name and the DNS/BIND domain name (for example, `host1.subdomain.example`).

**3** Configured Tunnel

- Interface

  The name of the configured tunnel interface. For example, IT0.

- Destination IPv4 address

  The remote node's IPv4 address (the remote end of the tunnel).

- Source IPv4 address

  Your node's IPv4 address (this end of the tunnel).

- RIPng

  If your system is a router and you want the router to run the RIPng protocol on the tunnel link to exchange IPv6 routing information with a router at the remote end of the tunnel, check Yes; otherwise, check No.

- Address prefix

  If your system is a router and you want to advertise address prefixes to the node at the remote end of the tunnel, enter a 64-bit prefix; otherwise, write Done. If your system is an IPv6 host and the router at the remote end of the tunnel is not advertising an address prefix, enter a 64-bit prefix to be configured on the tunnel interface.

4 Router

- Interface

  The name of the interface on which you want to run the RIPng protocol or advertise an address prefix.

- RIPng

  If you want the router to run the RIPng protocol on the specified interface and to exchange IPv6 routing information with other routers on the LAN, check Yes; otherwise, check No.

- Address prefix

  If you want to advertise address prefixes to all hosts on the link, enter a 64-bit prefix; otherwise, write Done. If you do not specify a 64-bit prefix, the router will not advertise an address prefix. All hosts must obtain their prefix information from another source. Prefixes in IPv6 define a subnet and are typically configured on a router for a specific link by the network administrator. The router advertises this prefix to all nodes connected to that link, along with the length of the prefix, whether the prefix is on link (that is, a neighbor), whether the prefix can also be used for stateless address configuration, and the length of time the prefix is valid.

5 Manual Routes

- Destination prefix

  The address prefix of a remote IPv6 network. The address prefix contains a Classless Inter-Domain Routing (CIDR) style bit length, for example, 5F00::/8. If you want to use the default route, write Default.

- Interface

  The name of the interface through which you are sending traffic to the remote IPv6 network.

- Next hop address

  The IPv6 address of the first router in the path to the destination prefix. Write the link local address of the router. If the connection to the router is over an IPv4 tunnel, write the link local IPv6 address of the remote tunnel endpoint.

When you run the TCPIP$IP6_SETUP configuration utility, it gathers information from the system and prompts you for additional configuration information.

## 2.2 IPv6 System Configuration Examples

This section shows how to use the configuration worksheet to assemble information for selected configurations. Each example shows how individual systems are configured. In some cases, additional options for you to consider are provided.

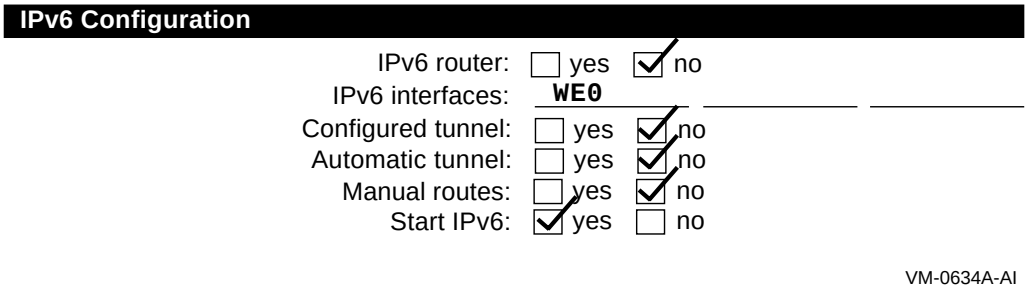
**Note**

OpenVMS interface names must be in uppercase.

### 2.2.1 Simple Host-to-Host Configuration

In a simple host-to-host configuration (shown in Figure 1–10), host A and host B use IPv6 link-local addresses. By default, the TCPIP$IP6_SETUP command configures the hosts automatically with a link-local address for your system. Figure 2–2 shows the completed worksheet for host A.

**Figure 2–2 Simple Host-to-Host Configuration**




After you configure IPv6 on host A, add a link-local address for host B to the TCPIP$ETC:IPNODES.DAT file. (For more information about these files, see Section 3.4.) The configuration process for host B in this configuration is similar to that for host A.

In this configuration, no global address prefix is advertised on the LAN. If you want to advertise a global address prefix, you can either configure one of the hosts as a router by using TCPIP$IP6_SETUP or add an IPv6 router to the LAN configuration. An IPv6 router advertises a global prefix on the link.

You can use the `netstat -in` command to view a local node's link-local and global addresses.

The following `TELNET` command connects host A to host B using host B's link-local address:

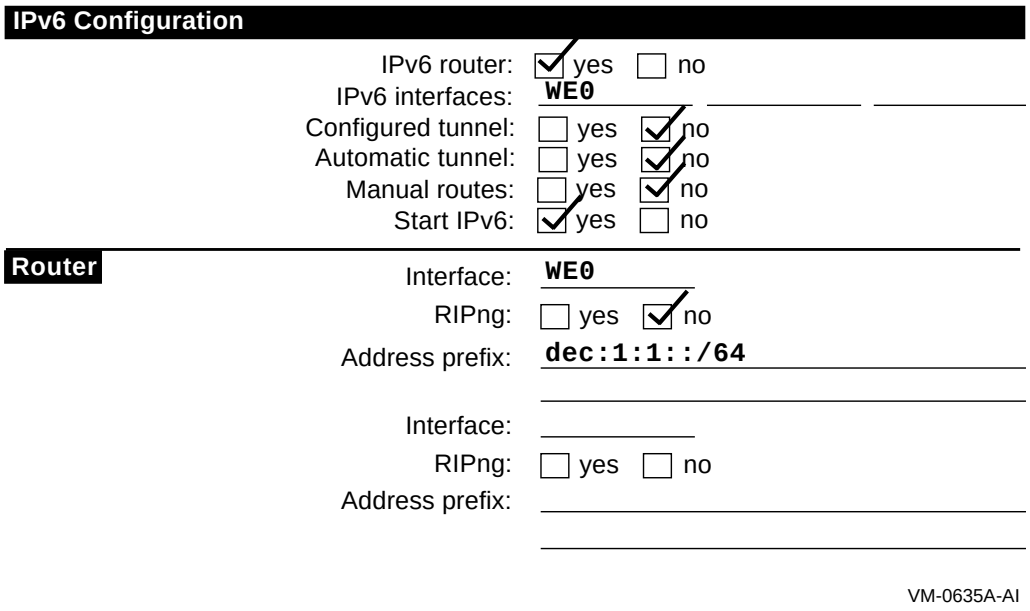
```
$ TELNET fe80::0a00:2bff:fee2:1e11
```

Alternately, you can place the address and node name in the TCPIP$ETC:IPNODES.DAT file. Then use the node name as the argument to the `TELNET` command. (For more information about this file, see Section 3.4.)

### 2.2.2 Host-to-Host with Router Configuration

In a host-to-host with router configuration (shown in Figure 1–11), host A and host B are on a LAN with router A. In this case, router A advertises the global address prefix `dec:1:1::/64` on the LAN. Host A and host B use this address prefix to create global IPv6 addresses. (See Chapter 1 for information about obtaining experimental testing addresses.) Figure 2–3 shows the completed worksheet for router A.

**Figure 2–3 Host-to-Host with Router Configuration**

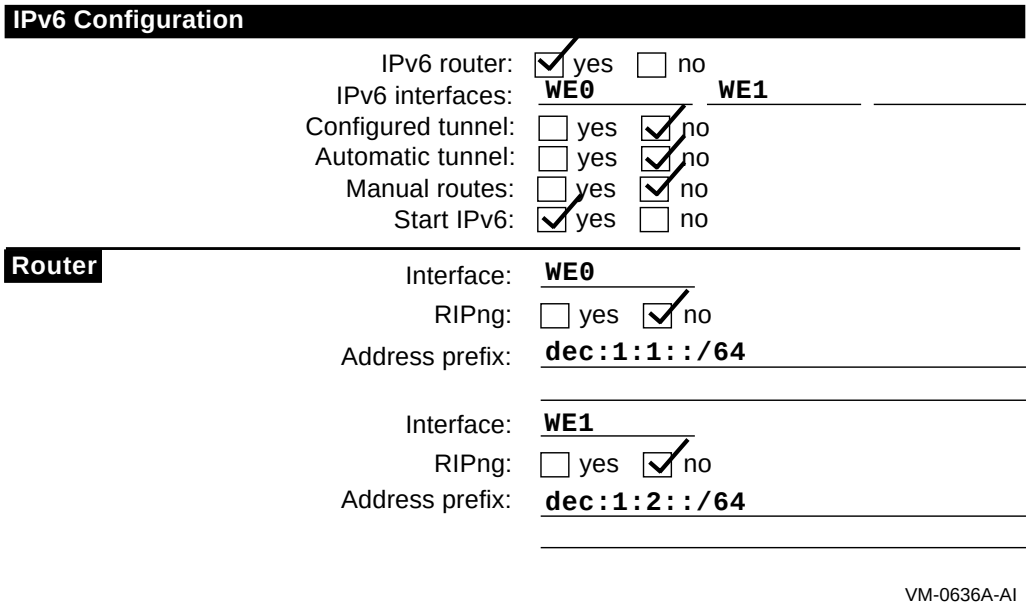



After you configure IPv6 on router A, add the global addresses for the other hosts to the TCPIP$ETC:IPNODES.DAT file. (For more information about this file, see Section 3.4.) Repeat this step on host A and host B. Alternatively, you could establish DNS/BIND in your network using the global addresses.

### 2.2.3 IPv6 Network-to-IPv6 Network with Router Configuration

In an IPv6 network-to-IPv6 network with router configuration (shown in Figure 1–12), two IPv6 networks are connected to each other through router A and its two interfaces. Figure 2–4 shows the completed worksheet for router A.

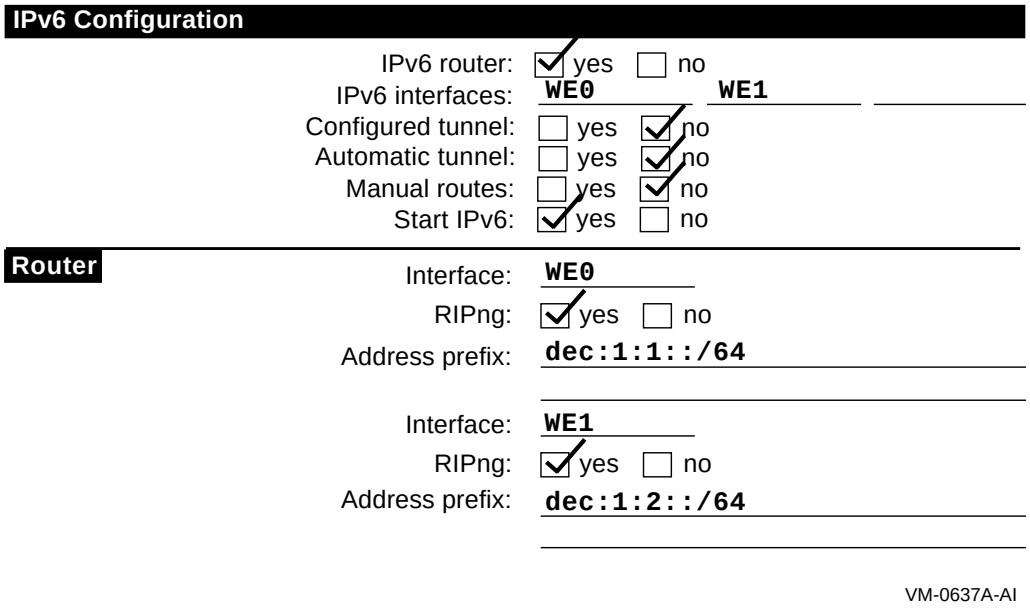
**Figure 2–4 IPv6 Network-to-IPv6 Network with Router Configuration**




### 2.2.4 Multiple IPv6 Networks and Multiple Routers Configuration

In this example configuration (shown in Figure 1–13), four IPv6 networks are connected to each other using three routers. In this configuration, the routers must exchange routing information in order to learn the routes to other subnets in the network. To accomplish this, each router must run the RIPng protocol. Figure 2–5 shows the completed worksheet for router A.

**Figure 2–5 Multiple IPv6 Networks and Multiple Routers Configuration**

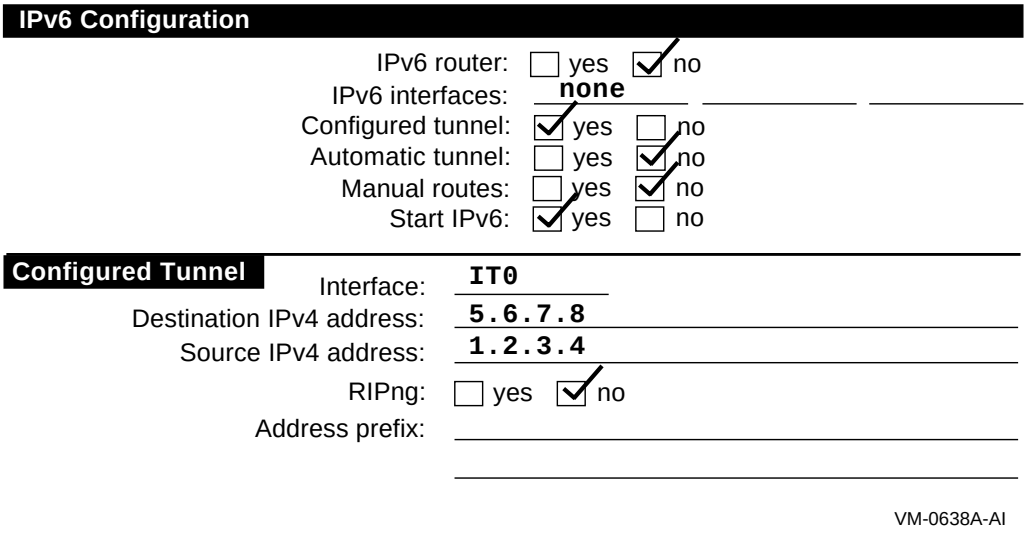



The completed worksheets for router B and C would be similar.

## 2.2.5 Host-to-Host over Tunnel Configuration

In a host-to-host over tunnel configuration (shown in Figure 1–14), two IPv6 systems communicate with each other over a configured tunnel through an IPv4 network and use IPv6 link-local addresses. Figure 2–6 shows the completed worksheet for host A.

**Figure 2–6 Host-to-Host over Tunnel Configuration**




After you configure IPv6 on host A, add the link-local address for host B to the TCPIP$ETC:IPNODES.DAT file. (For more information about this file, see Section 3.4.) The configuration process for host B in this configuration is similar to that for host A.

With this configuration, no global address prefix is advertised on the tunnel. If you want to advertise a global address prefix, you can configure one of the hosts as a router by using TCPIP$IP6_SETUP. An IPv6 router advertises a global prefix on the link.

To view a local node's link-local and global addresses, use the `netstat -in` command.

The following `TELNET` command connects host A to host B:

```
$ telnet fe80::5.6.7.8
```

Alternately, you can place the address and node name in the TCPIP$ETC:IPNODES.DAT file. Then use the Node name as the argument to the `TELNET` command.

### 2.2.6 Host-to-Router over Tunnel Configuration

In a host-to-router over tunnel configuration (shown in Figure 1–15), host X communicates with host B over a configured tunnel through an IPv4 network; both nodes use IPv6 addresses. The tunnel in this case is between host X and router A. Figure 2–7 shows the completed worksheet for host X when router A is advertising itself as the default router for the tunnel link and is advertising a global address prefix on the tunnel link.

**Figure 2–7 Host-to-Router over Tunnel Configuration**

**IPv6 Configuration**

IPv6 router: ☐ yes ☑ no
IPv6 interfaces: **none**
Configured tunnel: ☑ yes ☐ no
Automatic tunnel: ☐ yes ☑ no
Manual routes: ☐ yes ☑ no
Start IPv6: ☑ yes ☐ no

**Configured Tunnel**

Interface: **IT0**
Destination IPv4 address: **5.6.7.8**
Source IPv4 address: **1.2.3.4**
RIPng: ☐ yes ☑ no
Address prefix:

VM-0638A-AI

If router A is not advertising a global address prefix on the tunnel link, the value `dec:3:1::/64` would be in the Address prefix field in the Configured Tunnel section of the host X worksheet. If router A is not advertising itself as the default router for the tunnel link, the information shown in Figure 2–8 would also be on the host X worksheet:

**Figure 2–8 Router Not Advertising a Global Address Prefix**

Manual routes: ☑ yes ☐ no

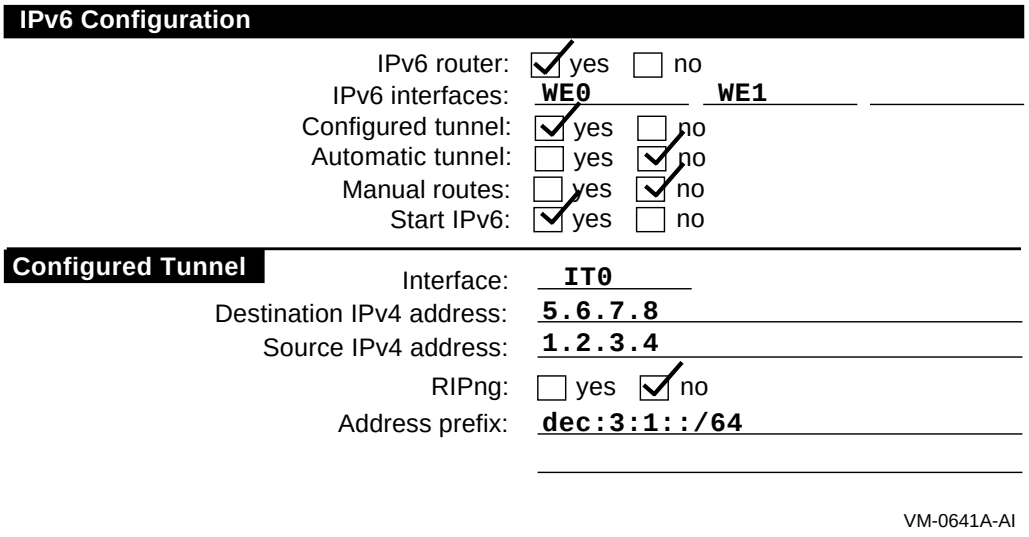
**Manual Routes**

Destination prefix: **default**
Interface: **IT0**
Next hop address: **fe80::1.2.3.4**

VM-0640A-AI

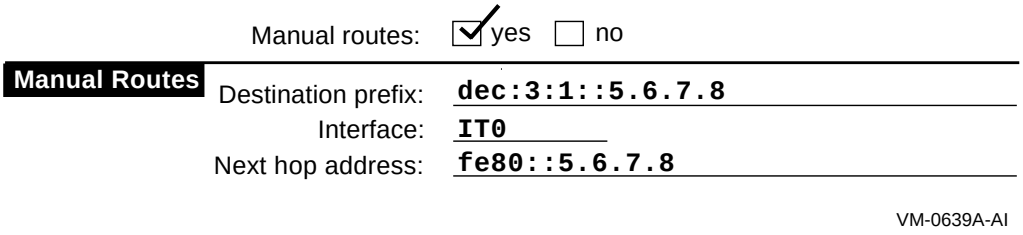
Figure 2–9 shows the completed worksheet for router A when router A is advertising a global address prefix on the tunnel link.

**Figure 2–9 Router Advertising a Global Address Prefix**




If router A is not advertising a global prefix on the tunnel link, the information shown in Figure 2–10 would be on the router A worksheet. Note the manual route to host X. Instead of specifying a destination network prefix, you specify the host route, `dec:3:1::5.6.7.8`, to host X. The next hop is the link-local IPv6 address of host X's tunnel interface, `fe80::5.6.7.8`.

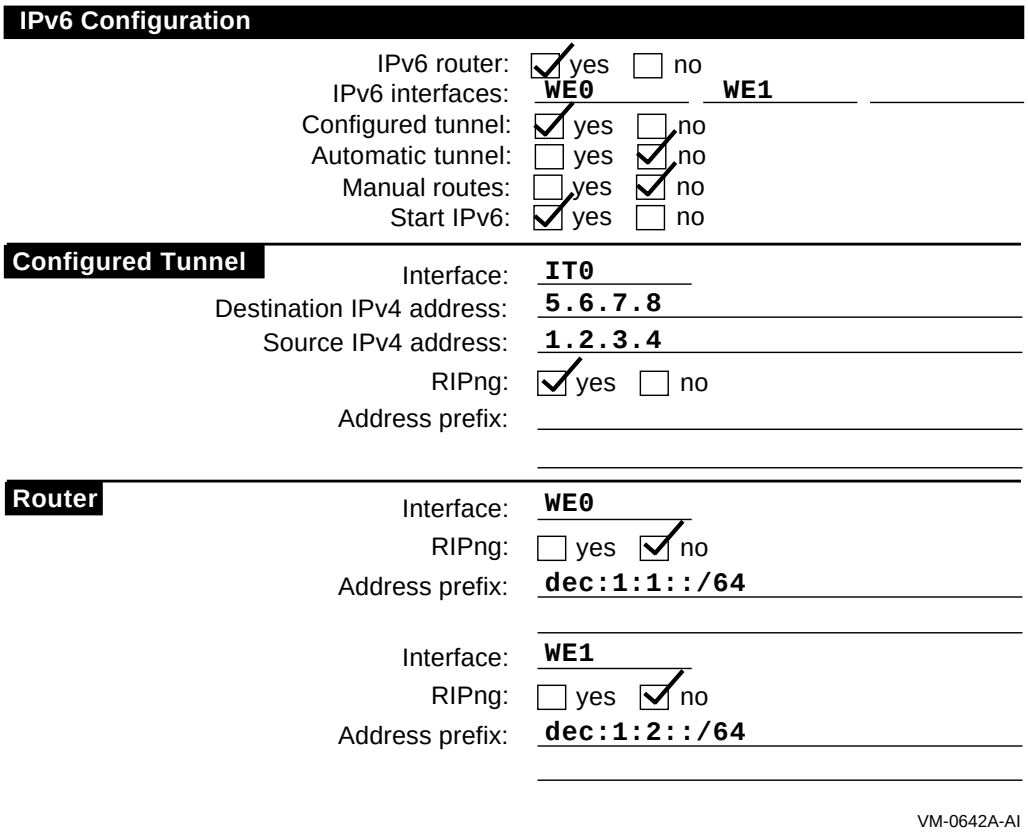
**Figure 2–10 Router A Not Advertising a Global Prefix on the Tunnel Link**




## 2.2.7 IPv6 Network to IPv6 Network over Tunnel Configuration

In an IPv6 to IPv6 network over tunnel configuration (shown in Figure 1–16), host A communicates with host F over a configured tunnel through an IPv4 network. The host configuration is similar to that of host A Section 2.2.1. All hosts automatically use their default router in order to communicate with hosts on other networks. Figure 2–11 shows the worksheet for router A.

**Figure 2–11 IPv6 Network to IPv6 Network over Tunnel Configuration**




You do not have to run RIPng on the WE0 and WE1 interfaces because no routers are attached to the interfaces.

The configuration of router B is similar, except that the source and destination addresses for the configured tunnel would be switched and the address prefixes advertised on WE0 and WE1 would be `dec:2:1::/64` and `dec:2:2::/64`, respectively.

**Note**

If the routers were not configured to use RIPng over the tunnel interface, each router would need to specify a manual route to the other.

## 2.3 Configuring IPv6 Hosts and Routers

This section describes how to configure your system as either an IPv6 host or an IPv6 router.

### 2.3.1 Configuring an IPv6 Host

To configure your system as an IPv6 host, do the following:

1. Log in as SYSTEM. Configure your IPv4 stack through the menu-driven TCPIP$CONFIG configuration procedure. This procedure is described in the *Compaq TCP/IP Services for OpenVMS Installation and Configuration* manual.

   **Note**

   Add the following line to your LOGIN.COM file:

   ```
   $ @SYS$MANAGER:TCPIP$DEFINE_COMMANDS.COM
   ```

   This command procedure defines the UNIX management commands as foreign commands. Rerun your LOGIN.COM to make the definitions effective for the current process.

2. Invoke the TCPIP$IP6_SETUP utility by entering the following command:

   ```
   $ @SYS$MANAGER:TCPIP$IP6_SETUP
   ```

   The utility displays information about the IPv6 network configuration procedure and tells you that you can configure the system as either an IPv6 host or an IPv6 router.

3. Choose to configure the system as an IPv6 host by responding to the following prompt:

   ```
   Configure this system as an IPv6 router? [NO]:
   ```

   Press Return to configure the system as an IPv6 host.

4. Answer the prompts about configuring each interface on your system. The procedure displays the following questions:

   ```
        Do you want to enable IPv6 on this interface?
   Enable IPv6 on interface WF0? [YES]:
   ```

   Press Return if you want to enable IPv6 on this interface; enter N if you do not.

   If your system has multiple interfaces, the procedure repeats this questions for each interface.

5. Indicate whether you want to configure an automatic tunnel by responding to the following prompt:

   ```
   Configure an IPv6 over IPv4 automatic tunnel interface? [NO]:
   ```

   If you want to configure an automatic tunnel, enter Y and press Return; if not, press Return.

6. Indicate whether you want to create a configured tunnel or additional configured tunnels by responding to the following prompt:

   ```
   Create a configured tunnel? [NO]:
   ```

   If you want to create a configured tunnel, enter Y and press Return. You will be prompted for source and destination addresses in steps 7 and 8.

If you do not want to create a configured tunnel or if you have finished adding a series of configured tunnels, press Return. The procedure goes to step 10.

7. If you chose to create a configured tunnel, enter the tunnel's source IPv4 address in response to the following prompt:

   ```
   Source IPv4 address of tunnel IT0?:
   ```

   Enter an IPv4 address in the standard format (*XX.XX.XX.XX*) and press Return.

8. Enter the tunnel's destination IPv4 address in response to the following prompt:

   ```
   Destination IPv4 address of tunnel IT0?:
   ```

   Enter an IPv4 address in the following format (*XX.XX.XX.XX*) and press Return.

9. Indicate whether you want to create another configured tunnel by responding to the following prompt:

   ```
   Create another configured tunnel? [NO]
   ```

   If you want to create another configured tunnel, enter Y and press Return. The procedure takes you back to steps 6 through 8 for each additional configured tunnel you choose to create.

   If you do not want to create another configured tunnel, press Return.

10. The procedure asks whether you want to create a host configuration file based on the choices you have made.

    ```
        Create IPv6 Host configuration file?
    Please enter YES or NO [YES]:
    ```

    If you are not satisfied with the configuration, enter N and press Return. The utility ends immediately without changing any of the current configuration files.

    If you are satisfied with the configuration, enter Y and press Return. The TCPIP$IP6_SETUP command procedure creates a configuration file called SYS$SYSTEM:TCPIP$INET6_CONFIG.DAT.

11. You must now shut down TCP/IP Services for OpenVMS and then restart the network in order to enable IPv6.

### 2.3.2 onfiguring an IPv6 Router

To configure your system as an IPv6 router, do the following:

1. Log in as SYSTEM. Configure your IPv4 stack through the menu-driven TCPIP$CONFIG configuration procedure. This procedure is described in the *Compaq TCP/IP Services for OpenVMS Installation and Configuration* manual.

**Note**

Add the following line to your LOGIN.COM file:

```
$ @SYS$MANAGER:TCPIP$DEFINE_COMMANDS.COM
```

This command procedure defines the UNIX management commands as foreign commands. Rerun your LOGIN.COM to make the definitions effective for the current process.

2. Invoke the TCPIP$IP6_SETUP utility by entering the following command:

   ```
   $ @SYS$MANAGER:TCPIP$IP6_SETUP
   ```

   The utility displays information about the IPv6 network configuration procedure and tells you that you can configure the system as either an IPv6 host or an IPv6 router.

3. Choose to configure the system as an IPv6 router by responding to the following prompt:

   ```
   Configure this system as an IPv6 router? [NO]:
   ```

   If you want to configure the system as an IPv6 router, enter Y and press Return.

4. Answer the prompts about configuring each interface on your system. The procedure displays the following questions:

   ```
        Do you want to enable IPv6 on this interface?
   Enable IPv6 on interface WF0? [YES]:
   ```

   Press Return if you want to enable IPv6 on this interface; enter N if you do not.

5. Answer the prompts about enabling IPv6 routing on each interface on your system. The procedure displays the following questions:

   ```
        Do you want to enable IPv6 routing on this interface?
   Enable IPv6 routing on interface WF0? [YES]:
   ```

   Press Return if you want to enable IPv6 routing on this interface; enter N if you do not.

6. Indicate whether you want the router to run the RIPng protocol on the designated interface by responding to the following prompt:

   ```
   Enable RIPng on interface WF0? [YES]:
   ```

   If you want the router to run the RIPng protocol, press Return; enter N and press Return if you do not.

7. Indicate whether you want the router to advertise an IPv6 address prefix for the LAN on the designated interface, by responding to the following prompt:

   ```
   Address prefix to advertise on interface WF0?:
   ```

If you want the router to advertise an IPv6 address prefix, enter a 64-bit address prefix for the interface and press Return. The procedure repeats the same prompt. You can enter as many additional prefixes as you want for the interface. When you are finished, enter Done and press Return.

If you do not want the router to advertise an IPv6 address prefix on the designated interface, enter Done and press Return.

If there are additional interfaces on your system, the procedure returns to steps 4 through 7 for each interface. Once you have configured all interfaces, the procedure goes to step 8.

8. Indicate whether you want to configure an automatic tunnel by responding to the following prompt:

   ```
   Configure an IPv6 over IPv4 automatic tunnel interface? [NO]:
   ```

   If you want to configure an automatic tunnel, enter Y and press Return; if not, press Return.

9. Indicate whether you want to create a configured tunnel or additional configured tunnels by responding to the following prompt:

   ```
   Create a configured tunnel? [NO]:
   ```

   If you want to create a configured tunnel, enter Y and press Return. You will be prompted for source and destination addresses in steps 10 and 11.

   If you do not want to create a configured tunnel or if you have finished adding a series of configured tunnels, press Return. The procedure goes to step 16.

10. If you chose to create a configured tunnel, enter the tunnel's source IPv4 address in response to the following prompt:

    ```
    Source IPv4 address of tunnel IT0?:
    ```

    Enter an IPv4 address in the standard format (*XX.XX.XX.XX*) and press Return.

11. Enter the tunnel's destination IPv4 address in response to the following prompt:

    ```
    Destination IPv4 address of tunnel IT0?:
    ```

    Enter an IPv4 address in the following format (*XX.XX.XX.XX*) and press Return.

12. Indicate whether you want to enable IPv6 routing on the interface by reponding to the following prompt:

    ```
    Enable IPv6 routing on interface IT0? [YES]:
    ```

    If you want to enable IPv6 routing on the interface, press Return; if not, enter N and press Return.

13. Indicate whether you want to enable RIPng on the interface by responding to the following prompt:

    ```
    Enable RIPng on interface IT0? [YES]:
    ```

    Press Return if you want to enable RIPng protocol on this interface; enter N and press Return if you do not.

14. Indicate whether you want the host to use an IPv6 address prefix on the tunnel interface by responding to the following prompt:

    ```
    Address prefix to advertise on interface IT0?:
    ```

    If you want the host to use an IPv6 address prefix because a router is not advertising a global address prefix, enter the prefix and press Return. Enter as many prefixes as you want. When you are finished entering prefixes for the interface, enter Done and press Return.

    If you do not want the host to use an IPv6 address prefix on the tunnel interface, enter Done and press Return.

15. Indicate whether you want to create another configured tunnel by responding to the following prompt:

    ```
    Create another configured tunnel? [NO]:
    ```

    If you want to create another configured tunnel, enter Y and press Return. The procedure returns to step 9.

    If you do not want to create another configured tunnel, press Return.

16. The TCPIP$IP6_SETUP utility displays the configuration information and asks you to indicate whether you want to update the current startup procedures with the new configuration information.

    ```
        Create IPv6 Router configuration files?
    Please enter YES or NO [YES]:
    ```

    If you are not satisfied with the configuration, enter N and press Return. The utility ends immediately without changing any of the current configuration files.

    If you are satisfied with the configuration, enter Y and press Return. The TCPIP$IP6_SETUP command procedure creates a configuration file called SYS$SYSTEM:TCPIP$INET6_CONFIG.DAT and a router configuration file called SYS$SYSTEM:TCPIP$IP6RTRD.CONF, both with default values.

17. You must now shut down TCP/IP Services for OpenVMS and then restart the network in order to enable IPv6.

## 2.4 Postconfiguration Tasks

After restarting the network with IPv6 enabled, you might want to do the following:

- Connect to the 6bone network
- Initialize a new interface for IPv6
- Create a configured tunnel
- Add addresses to or delete addresses from an interface
- Add or delete a default router
- Manually add a route for an onlink prefix
- Configure a router
- Edit the router configuration file

The following sections describe these tasks.

### 2.4.1 Connecting to the 6bone Network

To connect to the 6bone, choose a 6bone point that is reasonably close to your normal IPv4 paths into the Internet. The 6bone web site at `http://www.6bone.net` contains information on how to join the 6bone and how to find an attachment point. If you want to connect to the 6bone through the Compaq Palo Alto site either before or after you configure IPv6 on your host or router, complete the following steps:

1. Register your IPv4 tunnel by sending your 6bone IPv6 address prefix and the IPv4 address of your router to the following address:

   ```
   gw-6bone@pa.dec.com
   ```

2. Wait for confirmation that support for your tunnel is configured at Compaq.

   Compaq will provide both an IPv6 global address prefix for you to use at your site and the IPv4 address of the Compaq Palo Alto router.

3. Configure your tunnel by running the TCPIP$IP6_SETUP utility.

4. Verify that your tunnel is operational by issuing the `ping` command to one of the following Compaq IPv6 nodes:

   ```
   altavista.ipv6.digital.com
   ftp.ipv6.digital.com
   www.ipv6.digital.com
   ```

   For additional information on connecting to the 6bone, see the 6bone home page at the following location:

   ```
   http://www.6bone.net
   ```

### 2.4.2 Initializing a New Interface for IPv6

In some cases, you might want to either add a new interface card to your system or change an interface card from one type to another. After the new card is installed, you must initialize it for IPv6 operation. To initialize an interface, use the `ifconfig` command with the following syntax:

```
ifconfig device ipv6 up
```

**Note**

OpenVMS interface names must be in uppercase. When you enter them with UNIX management commands at the DCL prompt, you must enclose the name of the interface in double quotation marks.

For LAN interfaces, the `ifconfig` command creates the link-local address (`FE80::`) and starts detection of duplicate addresses.

For example, to initialize Ethernet interface WE0 for use with IPv6, enter the following:

```
$ ifconfig "WE0" ipv6 up
```

To initialize the loopback interface for use with IPv6, enter the following:

```
$ ifconfig "LO0" ipv6 up
```

To initialize the automatic tunnel interface, enter the following:

```
$ ifconfig "TN0" ipv6 up
```

This command designates one of the system's IPv4 addresses for use as the tunnel endpoint.

If you want the designated IPv4 address to be the permanent tunnel endpoint, you must use TCPIP$IP6_SETUP.

#### 2.4.2.1 Setting the IPv6 Interface Identifier

You can set the IPv6 interface ID at the same time you initialize an interface by using the `ifconfig` command with the `ip6interfaceid` parameter. For example, to initialize Ethernet interface WE0 for use with IPv6 and to set its interface ID to the 64-bit value `0x0123456789abcdef`, enter the following:

```
$ ifconfig "WE0" ip6interfaceid ::0123:4567:89ab:cdef ipv6 up
```

Although the interface ID is expressed in standard IPv6 address format, only the low-order 64 bits are used.

#### 2.4.2.2 Removing IPv6 from an Interface

Removing IPv6 from an interface removes the IPv6 configuration associated with the interface, including all IPv6 addresses and IPv6 routes through the interface. To remove IPv6 from an interface, use the `ifconfig` command with the following syntax:

```
ifconfig device -ipv6
```

For example, to remove IPv6 from Ethernet interface WE0, enter the following:

```
$ ifconfig "WE0" -ipv6
```

### 2.4.3 Creating a Configured Tunnel

To create a configured tunnel, use the `iptunnel` command with the following syntax:

```
iptunnel create  []
```

For example, to create a tunnel to remote system 16.20.136.47, enter the following command:

```
$ iptunnel create 16.20.136.47
```

To initialize the tunnel for IPv6 operation, enter the following:

```
$ ifconfig "IT0" ipv6 up
```

**Note**

OpenVMS interface names must be in uppercase. When you enter them with UNIX management commands at the DCL prompt, you must enclose the name of the interface in double quotation marks.

### 2.4.4 Adding an Address to an Interface

To add or assign an IPv6 prefix to an interface and to direct the kernel to automatically append the interface identifier, use the `ifconfig` command with the following syntax:

```
ifconfig  inet6 ip6prefix
```

The following example assigns the address `dec:2::0a00:2bff:fe12:3456` to interface WE0 (the interface ID is `0a00:2bff:fe12:3456`):

```
$ ifconfig "WE0" inet6 ip6prefix dec:2::/64
```

The **ip6prefix** parameter directs the kernel to automatically append the interface identifier to the address prefix.

To add or assign a full IPv6 address to an interface manually, use the `ifconfig` command with the following syntax:

```
ifconfig  inet6
```

The following example assigns the address dec:2::1 to interface WE0:

```
$ ifconfig "WE0" inet6 dec:2::1
```

**Note**

For IPv6 hosts, the TCPIP$ND6HOST process configures interface prefixes automatically, depending on the contents of router advertisements.

For IPv6 routers, the TCPIP$IP6RTRD process configures interface prefixes automatically, depending on the contents of the SYS$SYSTEM:TCPIP$IP6RTRD.CONF file.

### 2.4.5 Deleting an Address from an Interface

To delete an IPv6 address from an interface manually, use the `ifconfig` command with the following syntax:

```
ifconfig  inet6 delete
```

For example:

```
$ ifconfig "WE0" inet6 delete dec:2::1
```

**Note**

OpenVMS interface names must be in uppercase. When you enter them with UNIX management commands at the DCL prompt, you must enclose the name of the interface in double quotation marks.

### 2.4.6 Adding or Deleting a Default Router

To add a default router, use the route utility with the following syntax:

```
route add -inet6 default  -I
```

For example:

```
$ route add -inet6 default fe80::0a00:2bff:fe12:3456 -"I" "WE0"
```

**Note**

UNIX flags and OpenVMS interface names are case sensitive. When entering UNIX management commands at the DCL prompt, you must enclose uppercase UNIX flags and OpenVMS interface names in quotes.

To delete a default router, use the route utility with the following syntax:

```
route delete -inet6 default  -I
```

For example:

```
$ route delete -inet6 default fe80::0a00:2bff:fe12:3456 -"I" "WE0"
```

**Note**

For IPv6 hosts, the TCPIP$ND6HOST process performs the add and delete router operations automatically, depending on the contents of router advertisements.

### 2.4.7 Manually Adding a Route for an On-Link Prefix

After you manually add an address prefix to an interface, you also can add a static route so that traffic to other hosts with the same prefix is sent directly to the destination rather than through a router. For example, if the prefix DEC:5::/64 has been added to the Ethernet interface WE0, which has been initialized with the link-local address fe80::0a00:2bff:fe12:3456, the following command adds a route to neighboring hosts with the same prefix:

```
$ route add -inet6 dec:5::/64 fe80::0a00:2bff:fe12:3456 -interface
```

This command specifies that destinations with prefix dec:5::0/64 are reachable through the interface with address fe80::0a00:2bff:fe12:3456. That is, dec:5::0/64 is an on-link prefix.

**Note**

For IPv6 hosts, the TCPIP$ND6HOST process automatically adds on-link prefixes based on the contents of router advertisements.

### 2.4.8 Configuring a Router

Before configuring a router, you must enable forwarding by setting the **ipv6forwarding** and **ipv6router** attributes of the kernel inet subsystem to 1. You set these attributes by entering the following sysconfig commands:

```
$ sysconfig -r inet ipv6forwarding=1
$ sysconfig -r inet ipv6router=1
```

### 2.4.9 Editing the Router Configuration File

After you configure the system as an IPv6 router, the TCPIP$IP6RTRD process sends out periodic router advertisements for the following reasons:

- To advertise itself as a potential default router for IPv6 traffic. The IPv6 hosts on the link receive these advertisements as part of their Neighbor Discovery processing.
- To advertise an IPv6 address prefix, in which case hosts on the link perform address autoconfiguration.

The SYS$SYSTEM:TCPIP$IP6RTRD.CONF file contains the configuration data needed to send Router Advertisement messages. This file is created when TCPIP$IP6_SETUP is run (if the system is configured as a router). The link interface and advertised prefix are inserted, and other default values are used. You can modify this file as appropriate for your network, for example, when using multiple prefix values. Example 2–1 shows a sample configuration file.

**Example 2–1 Sample TCPIP$IP6RTRD.CONF File**

```
#
# Sample ip6rtrd configuration file
#
interface WE0 {
        MaxRtrAdvInterval 600
        MinRtrAdvInterval 200
        AdvManagedFlag 0
        AdvOtherConfigFlag 0
        AdvLinkMTU 1500
        AdvReachableTime 0
        AdvRetransTimer 0
        AdvMaxHopLimit 64
        AdvDefaultLifetime 1800
        Prefix dec:1::/64 {
                AdvValidLifetime 1200
                AdvPreferredLifetime 600
                AdvOnLinkFlag 1
                AdvAutonomousFlag 1
        }
}
```

See Section B.2.2 for more information about the TCP/IP$IP RTRD.CONF. file.

# 3
# Configuring BIND

The information in this chapter is for experienced DNS/BIND administrators. If you are not a DNS/BIND administrator, give this information to the administrator for your site.

The DNS implementation is based on BIND Version 8.1.2, which provides more extensive configuration options than previous versions (for example, access control lists, categorized logging). As a result, the configuration format has changed. In previous releases, the BIND configuration was stored in UCX$CONFIGURATION.DAT. With TCP/IP Services Version 5.1, the BIND configuration is maintained as an ASCII text file called TCPIP$BIND.CONF.

**Important**

For IPv6 environments, the BIND server supports AAAA lookups over IPv4 (AF_INET) connections only. The resolver and server have not been ported to IPv6, but IPv6 applications can make `getaddrinfo` and `getnameinfo` calls to retrieve the AAAA records.

The BIND resolver and server support dynamic updates to the DNS/BIND database. See Section 3.3 for information about enabling this feature.

## 3.1 IPv6 Server Guidelines

Configuring an IPv6 master server is similar to configuring an IPv4 master server with a few exceptions. The following sections describe the exceptions.

To configure a DNS/BIND server to operate in an IPv6 environment, review the following guidelines:

- Select a node to function as an IPv6 name server.
- Dedicate a zone to IPv6 addresses, or add IPv6 addresses to your enterprise's current zone.
- If you want global IPv6 name services, you must delegate a domain under the `ip6.int` domain for the reverse lookup of IPv6 addresses. Send mail to the following address to request a domain for reverse lookups:

  `bmanning@isi.edu`

  See RFC 1886 for more information.

- If the system is configured as a DNS/BIND server, change the resolver configuration to point to the local node for name lookups, as follows:

  1. Run the TCP/IP Services configuration procedure:

     ```
     >
     $ @SYS$STARTUP:TCPIP$CONFIG
     ```

  2. Select Core Environment.
  3. Select Resolver.
  4. Enter the BIND server of LOCALHOST.

## 3.2 Sample BIND Configuration Files

The SYS$COMMON:[SYSHLP.EXAMPLES.TCPIP.IPV6.BIND] directory contains DNS configuration files that show sample IPv6 information for you to study and adapt to your environment. Of the files in that directory, the following example files contain IPv6 information that show reverse lookup addresses:

**Example 3–1 Sample IPV6.DB File**

```
; ****************************************************************
; *                                                              *
; *    Copyright 2000 Compaq Computer Corporation                *
; *                                                              *
; *   The software contained on this media  is  proprietary  to  *
; *   and  embodies  the  confidential  technology  of  Compaq   *
; *   Computer Corporation.  Possession, use,  duplication  or   *
; *   dissemination of the software and media is authorized only *
; *   pursuant to a valid written license from Compaq Computer   *
; *   Corporation.                                               *
; *                                                              *
; *   RESTRICTED RIGHTS LEGEND   Use, duplication, or disclosure *
; *   by the U.S. Government is subject to restrictions  as  set *
; *   forth in Subparagraph (c)(1)(ii)  of  DFARS  252.227-7013, *
; *   or  in  FAR 52.227-19, as applicable.                      *
; *                                                              *
; ****************************************************************

;
; Example BIND data file for ipv6.my.domain
;
@       IN      SOA     ns.my.domain. postmaster.ns.my.domain. (
                        1       ; Serial
                        3600    ; Refresh
                        300     ; Retry
                        3600000 ; Expire
                        3600 )  ; Minimum
;
; Nameservers (must have IPv4 addresses until BIND gets ported to IPv6)
;
        IN      NS      ns.my.domain.
;
; IPv6 nodes
;
host1 IN AAAA 5F00:0000:0102:0300:0203:0800:2B0A:0B0C
host2 IN AAAA 5F00:0000:0102:0300:0203:0800:2B0D:0E0F
```

**Example 3–2 Sample IPV6.REV File**

```
; ****************************************************************
; *                                                              *
; *    Copyright 2000 Compaq Computer Corporation                *
; *                                                              *
; *   The software contained on this media  is  proprietary  to  *
; *   and  embodies  the  confidential  technology  of  Compaq   *
; *   Computer Corporation.  Possession, use,  duplication  or   *
; *   dissemination of the software and media is authorized only *
; *   pursuant to a valid written license from Compaq Computer   *
; *   Corporation.                                               *
; *                                                              *
; *   RESTRICTED RIGHTS LEGEND   Use, duplication, or disclosure *
; *   by the U.S. Government is subject to restrictions  as  set *
; *   forth in Subparagraph (c)(1)(ii)  of  DFARS  252.227-7013, *
; *   or  in  FAR 52.227-19, as applicable.                      *
; *                                                              *
; ****************************************************************

;
; Example BIND data file for 3.0.2.0.0.0.3.0.2.0.1.0.0.0.0.0.0.0.0.f.5.IP6.INT
;
; (corresponds to the 5F00:0000:0102:0300:0203::/80 prefix)
;
@       IN      SOA     ns.my.domain. postmaster.ns.my.domain. (
                        1       ; Serial
                        3600    ; Refresh
                        300     ; Retry
                        3600000 ; Expire
                        3600 )  ; Minimum
;
; Nameservers (must have IPv4 addresses until BIND gets ported to IPv6)
;
        IN      NS      ns.my.domain.
;
; IPv6 nodes
;
c.0.b.0.a.0.b.2.0.0.8.0 IN PTR host1.ipv6.my.domain.
f.0.e.0.d.0.b.2.0.0.8.0 IN PTR host2.ipv6.my.domain.
```

## 3.3 Enabling Dynamic Updates to the DNS Database

To enable dynamic updates for a DNS/BIND server, do the following:

- Edit the TCPIP$BIND.CONF file and add the allow-update substatement to the zone statements for those zones you want to dynamically update and for the reverse lookup zone. For example, the following statements are the result of editing the first two zone statements in Example 3–3 and making the required changes:

**Example 3–3 Sample TCPIP$BIND.CONF_IPV6 File**

```
// ********************************************************************
// *                                                                  *
// *    Copyright 2000 Compaq Computer Corporation                    *
// *                                                                  *
// *   The software contained on this media  is  proprietary  to      *
// *   and  embodies  the  confidential  technology  of  Compaq       *
// *   Computer Corporation.  Possession, use,  duplication  or       *
// *   dissemination of the software and media is authorized only     *
// *   pursuant to a valid written license from Compaq Computer       *
// *   Corporation.                                                   *
// *                                                                  *
// *   RESTRICTED RIGHTS LEGEND   Use, duplication, or disclosure     *
// *   by the U.S. Government is subject to restrictions  as  set     *
// *   forth in Subparagraph (c)(1)(ii)  of  DFARS  252.227-7013,     *
// *   or  in  FAR 52.227-19, as applicable.                          *
// *                                                                  *
// ********************************************************************

//
// Example named.conf file
//

options {
        directory "sys$specific:[tcpip$bind]";
};

zone "ipv6.my.domain" {
 type master;
 file "ipv6.db";
};

zone "3.0.2.0.0.0.3.0.2.0.1.0.0.0.0.0.0.0.0.0.f.5.IP6.INT" {
 type master;
 file "ipv6.rev";
};

zone "0.0.127.in-addr.arpa" {
 type master;
 file "127_0_0.db";
};

zone "LOCALHOST" in {
        type master;
        file "LOCALHOST.DB";
};

zone "." {
        type hint;
        file "root.hint";
};


         zone "ipv6.my.domain" {
                 type master;
                 file "ipv6.db";
                 allow-update { any; };
         };
         zone "3.0.2.0.0.0.3.0.2.0.1.0.0.0.0.0.0.0.0.0.f.5.IP6.INT" {
          type master;
                 file "ipv6.rev";
                 allow-update { any; };
         };
```

- Start the TCP/IP Services product as follows:

```
$ SYS$STARTUP:TCPIP$STARTUP
```

## 3.4 Local Hosts Database TCPIP$ETC:IPNODES.DAT

TCP/IP Services for OpenVMS provides an editable ASCII version of the local hosts database, TCPIP$ETC:IPNODES.DAT, to support local definition of IPv6 addresses.

Configuring the BIND resolver using TCPIP$CONFIG.COM will produce a template file in TCPIP$ETC:IPNODES.DAT.

**Note**

Be aware that TCPIP SET/SHOW HOST commands do not operate on this file and will affect only the traditional (RMS indexed) local hosts database.

The IPNODES file contains information regarding the known IP nodes (both IPv4 and IPv6) on the network.

For each node, a single line should be present with the following information:

```
IP_address canonical_nodename aliases
```

Items are separated by any number of blanks or tab characters, or both. The pound sign (#) indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines that search the file.

Network addresses, both IPv4 and IPv6, are specified in IPv6 notation using the `inet_pton( )` routine from the Internet address manipulation library. Node names can contain any printable character other than a field delimiter, newline, or comment character.

The following routines, `getaddrinfo( )` and `getnameinfo( )` as defined in the Internet draft that supersedes RFC 2553 (Basic Socket Interface Extensions for IPv6), support the use of the TCPIP$ETC:IPNODES.DAT file.

For details about using these routines see, Section 6.5.1.1 and Section 6.5.2.1.

## 3.5 Converting from BIND 4.9*

The TCP/IP Services product provides a rollover utility you can use to convert your UCX BIND configuration to the new BIND 8.1 format. Issue the TCPIP CONVERT/CONFIGURATION BIND command to convert your files to the new format.

See the *Compaq TCP/IP Services for OpenVMS Management* guide for more information about this utility.

# 4
# Monitoring the Network

To monitor your network, use the following UNIX style management tools:

- ping command
- netstat command
- traceroute command
- IPv6 process log files

See Appendix B for more information about both IPv6 extensions to the management utilities and IPv6 processes.

The following sections describe each topic.

## 4.1 Testing Access to Internet Network Hosts with the ping Command

The ping command accepts an IPv4 address, IPv6 address, or node name on the command line. The following sample command specifies an IPv6 address:

```
$ ping -c 2 5F00:2100:108C:4000:8C40:800:2B2D:2B2

PING (5F00:2100:108C:4000:8C40:800:2B2D:2B2): 56 data bytes
64 bytes from 5F00:2100:108C:4000:8C40:800:2B2D:2B2: icmp6_seq=0
     hlim=58 time=17 ms
64 bytes from 5F00:2100:108C:4000:8C40:800:2B2D:2B2: icmp6_seq=1
     hlim=58 time=17 ms
----5F00:2100:108C:4000:8C40:800:2B2D:2B2 PING Statistics----
2 packets transmitted, 2 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 17/17/17 ms
```

The ping command accepts a -V4 or -V6 flag to send an IPv4 ECHO_REQUEST to a node with an IPv4 address, or to send an IPv6 ECHO_REQUEST to a node with an IPv6 address, respectively. If you do not specify either flag, the ping command sends an appropriate ECHO_REQUEST based on the address family being used.

You can also use the -I flag to force the use of a specific interface. For example:

```
$ ping -"I" "WE0" FE80::800:2B2D:2B2
```

## 4.2 Displaying Network Statistics with the netstat Command

You can display network statistics for sockets, interfaces, and routing tables. The netstat command accepts either the -f inet or -f inet6 flag to limit the data displayed to either IPv4 or IPv6, respectively. For example, the netstat -f inet6 -rn command displays only IPv6 routing table entries, whereas the default displays both IPv4 and IPv6 entries.

The `netstat -s` commmand displays statistics for all protocols including IPv6 and ICMPv6.

```
$ netstat -s
```

**Note**

UNIX flags are case sensitive. When using an uppercase flag you must enclose it with quotes to get the expected behavior. OpenVMS interface names are case sensitive. The name of the interface must be enclosed with quotes.

## 4.3 Displaying a Datagram's Route to a Network Host with the traceroute Command

The `traceroute` command used with the `host` argument prints the route that packets take to both IPv4 and IPv6 hosts.

In the following examples, the backslash (\) and the continuation of output onto a second line is for display purposes only. In actual output, the information appears on a single line.

```
$ traceroute -n host1-v6
 traceroute to host1-v6.corp.com (3ffe:1200:4110:3:a00:2bff:feb4:89c5), \
30 hops max, 24 byte packets
1  fe80::a00:2bff:fe2a:1ed3  130.86 ms 119.141 ms  119.14 ms
2  3ffe:1200:4110:1:a00:2bff:fe2d:2b2  126.014 ms  117.308 ms  116.33 ms
3  3ffe:1200:4110:3:a00:2bff:feb4:89c5 122.195 ms  135.882 ms 119.263 ms
```

```
$ traceroute 3ffe:1200:4110:3:a00:2bff:feb4:89c5
traceroute to 3ffe:1200:4110:3:a00:2bff:feb4:89c5 \
(3ffe:1200:4110:3:a00:2bff:feb4:89c5), 30 hops max, 24 byte packets
1  fe80::a00:2bff:fe2a:1ed3 (fe80::a00:2bff:fe2a:1ed3) 123.046 ms \
114.258 ms  117.188 ms
2  host2-v6.corp.com (3ffe:1200:4110:1:a00:2bff:fe2d:2b2)  115.234 ms \
117.188 ms  116.287 ms
3  host1-v6.corp.com (3ffe:1200:4110:3:a00:2bff:feb4:89c5)  120.241 ms \
113.398 ms  120.24 ms
```

When the route has an IPv6 over IPv4 tunnel, `traceroute` views this as a single hop. It prints only the IPv6 addresses of the nodes at each end of a tunnel, and none of the intermediate IPv4 routers between the tunnel source and destination. If a traceroute command over a tunnel interface fails, run the command again and specify the tunnel's IPv4 destination address.

The following command shows a trace across the 6bone network to destination `tw4.es.net`. Note that the intermediate routers appear to drop every other message. The probable reason for this is that the routers rate-limit IPv6 ICMP error messages to one per second. Rate-limiting ICMP error messages is valid behavior.

In the following examples, the backslash (\) and the continuation of output onto a second line is for display purposes only. In actual output, the information appears on a single line.

```
$ traceroute tw4.es.net
traceroute to tw4.es.net (3ffe:780:40:1:a00:2bff:febc:e96c), 30 hops max, 24 byte packets
1  gw1.ipv6.pa-x.dec.com (3ffe:1280:1000:1::f842:1428)  83.985 ms * 83.000 ms
2  3ffe:700:20:1::21 (3ffe:700:20:1::21)  108.399 ms *  112.305 ms
3  3ffe:780:40:1:a00:2bff:febc:e96c(3ffe:780:40:1:a00:2bff:febc:e96c) \
124.023 ms  134.766 ms  116.211 ms
```

The following example shows a trace to destination `yogi-gbl` using 2000-byte messages. It also shows the effect of path MTU discovery on `traceroute` results.

```
$ traceroute yogi-gbl 2000
traceroute to yogi-gbl (fec0:10:60:0:200:f8ff:fe40:d8e6), 30 hops max, 2024 byte packets
1  a30rtr-gbl (fec0:10:30:0:200:f8ff:fe45:cfb2)  5.859 ms  3.906 ms  3.907 ms
2  fec0:10:20:0:a00:2bff:feb0:972d (fec0:10:20:0:a00:2bff:feb0:972d) \
4.882 ms  3.906 ms  3.906 ms
3  * fec0:10:40:1::a0a:283c (fec0:10:40:1::a0a:283c)  6.836 ms  6.836 ms
4  yogi-gbl (fec0:10:60:0:200:f8ff:fe40:d8e6)  8.789 ms  8.789 ms  7.812 ms
```

Hops 1 and 2 occur across Ethernet links that have a link MTU of 1500 bytes. Hop 3 occurs across a configured tunnel with an MTU of 1280 bytes.

The 1500-byte message fragments were transmitted without error until they hit the tunnel. The first fragment across hop 3 triggered a "message too big" error, which in turn caused the sender to record a reduced Path MTU for `yogi-gbl`. The sender sent all subsequent messages with smaller fragments. The `traceroute` display shows that the first probe to the tunnel was dropped but that all others succeeded.

## 4.4 IPv6 Process Log Files

The TCPIP$ND6HOSTD and TCPIP$IP6RTRD processes log informational and severe events in the TCPIP$ND6HOSTD.LOG and TCPIP$IP6RTRD.LOG files, which are located in the SYS$MANAGER directory.

Currently logging is always enabled.

# 5
# Solving IPv6 Problems

This chapter contains a diagnostic map to help you solve problems that might occur when you use the IPv6 network and network services. Use this chapter along with the appropriate Compaq documentation to solve as many problems as possible.

## 5.1 Using the Diagnostic Suggestions

IPv6 network and network service problems can occur for a number of reasons. This chapter should help you isolate the problem.

After you isolate the problem, the section refers you to other sections for instructions on how to use the various problem-solving tools and utilities.

You may experience problems that are not documented in this manual when you use the IPv6 network software with other products. See the getting started documentation for the other products for additional information.

## 5.2 Getting Started

Before you start problem solving, ensure that communications hardware is ready for use. Verify the following:

- The system's physical connections are properly installed. See the documentation for your system and communications hardware device.
- Event logging is enabled to monitor network events. See the system administration manual for information about starting event logging and for descriptions of event messages.

Also check the product release notes for up-to-date information on known problems.

You should be familiar with the following terms:

- On-link node

  An on-link node is attached to the same subnetwork as your system. This subnetwork can be a LAN or an IPv6-over-IPv4 configured tunnel. There are no IPv6 routers between your system and the on-link node.

  For a configured tunnel, the on-link node is the node at the destination end of the tunnel.

- Off-link node

  An off-link node is not attached to the same subnetwork as your system. There is at least one IPv6 router between your system and the off-link node.

## 5.3 Solving IPv6 Network Problems

This section describes the most basic causes of IPv6 network problems. Before investigating further, make sure you perform the following checks:

1. Make sure the system is on and has completed all startup procedures.

   Check the power to your system. See the system management manual for your system's startup procedure and any problem solving information.

2. Verify IPv6 installation.

   To verify that the IPv6 components are installed, enter the following command:

   ```
   $ TCPIP SHO VER/ALL
   ```

   TCP/IP Services Version 5.1 files should be listed. If the components are not listed, install TCP/IP Services for OpenVMS Version 5.1 by using the PCSI command. See the *Compaq TCP/IP Services for OpenVMS Installation and Configuration* manual for information about installing the product.

3. Verify IPv6 configuration.

   To verify that IPv6 is configured, enter the following command:

   ```
   $ DIR SYS$MANAGER:TCPIP$INET6_CONFIG.DAT
   ```

   See Chapter 2 for information about setting up and configuring an IPv6 host or router.

4. Verify that IPv6 is started.

   To verify that IPv6 is started, enter the following commands:

   ```
   $ SHO LOG TCPIP$IPv6_STARTED
   $ ping ::1
   ```

   If the "host is unreachable" message appears, enable IPv6 by entering the following command:

   ```
   $ @SYS$STARTUP:TCPIP$STARTUP
   ```

   This creates the IPv6 interfaces, brings them up, and starts the IPv6 processes.

   See Section 5.4 for a description of IPv6 host problems; see Section 5.5 for a description of IPv6 router problems.

## 5.4 Solving IPv6 Host Problems

This section describes possible problems with IPv6 hosts and procedures for solving them.

### 5.4.1 IPv6 Process Is Not Started

Verify that the TCPIP$ND6HOST process is running by issuing the following command:

```
$ SHO SYS /PROCESS=TCPIP$ND6HOST
```

If the process is not running, enable IPv6 with the following command:

```
$ @SYS$STARTUP:TCPIP$STARTUP.COM
```

This creates the IPv6 interfaces, brings them up, and starts the TCPIP$ND6HOST process.

### 5.4.2 Host Is Unknown

If a remote host is not known, the following message appears:

```
unknown host
```

Perform the following steps:

1. Check whether the user is using a valid host name to reach the remote host.
2. Check whether the remote host is in another name domain and whether the user specified the full domain name.
3. If your site uses the BIND name service for name-to-address translation, make sure the database contains an entry for the remote host.

   If it does not, edit the TCPIP$ETC:TCPIP$IPNODES.DAT file to add the host.
4. If you are using a BIND server to search the BIND database for name-to-addres s translation, make sure the resolver is pointing to a valid BIND server. If your nameserver is on the local host, make sure that the BIND server is running. See the *Compaq TCP/IP Services for OpenVMS Management* guide for additional information about setting up your BIND environment.

### 5.4.3 On-Link Node Is Not Reachable

If an on-link node is not reachable, one of the following messages appears:

```
host is unreachable
network is unreachable
timeout
```

Verify that an on-link node or router (if one exists) is reachable by using the `ping` command. If the command fails or if packets are frequently dropped, perform the following steps:

1. If the node is attached to a LAN, check the data link counters by using the LANCP SHO DEVICE *device* /COUNTERS command. Problems with the counters and their possible causes are as follows:
   - Zero blocks sent or received can indicate a network hardware failure or a wiring problem.
   - High collision rates can indicate an improperly wired network or a node that is sending excessive message traffic.
   - Data overrun and buffer unavailable errors indicate that your system is misconfigured.

2. If there is no problem with the data link counters, check the IPv6 and ICMPv6 counters with the `netstat -p ipv6` and `netstat -p ipv6 -icmp` commands, respectively. Problems with counters and their possible causes are:
   - Packets discarded because of errors, or errors resulting from ICMP errors, indicate that another node is generating invalid messages. Other counters show more specific information.
   - Allocation errors can indicate excessive message traffic, a misconfigured system, or a program that repeatedly allocates memory without freeing it.
3. Using the `ifconfig -a` command, verify that IPv6 network interfaces exist, are up, and have `inet6` addresses. If the interfaces do not have `inet6` addresses, check the startup file TCPIP$INET6_CONFIG.DAT. Run the TCPIP$IP6_SETUP utility to correct any errors.

   If your interface does not have a global or site-local address, contact your network administrator to verify that your local router is advertising a prefix on the link. If there is no local router, you can define a prefix by using the `ifconfig` command.
4. Contact the system administrator for the adjacent on-link node. Verify that the on-link node is up and running, that it is configured correctly for IPv6, and that the address you are using is enabled on the node's interface.
5. If IPv4 is configured on both systems, issue the `ping` command to the on-link node's IPv4 address, If the commands succeeds, verify the IPv6 configuration on both systems. If the command fails, see the appropriate troubleshooting manuals.
6. Issue the `ping` command to other nodes on the link to determine whether the failure is confined to one node or extends to multiple nodes. Partial connectivity might indicate a faulty network device or cable on the link.
7. If the link is a configured tunnel, do the following:
   a. Verify the tunnel source and destination addresses by using the `ifconfig -a` command. Contact the administrator for the tunnel destination node and verify that your source and destination addresses match the destination and source addresses on that node.
   b. Issue the `ping` command to the tunnel destination address. If the command fails, see the *Compaq TCP/IP Services for OpenVMS Management* guide for more information.

### 5.4.4 Off-Link Node Is Not Reachable

If an off-link node is not reachable, one of the following message appears:

```
host is unreachable
network is unreachable
timeout
```

Verify that an off-link node is reachable by issuing the `ping` command.

If there is 100% packet loss, perform the following steps:

1. Verify connectivity between your system and an on-link router by using the `ping` command.

If the command fails or shows frequently dropped packets, follow the steps in Section 5.4.3.

If you do not know the address to a router, issue the following command:

```
$ ping -"I" interface ff02::2
```

2. Verify that the interface over which you are sending messages has a global or site-local unicast address enabled by using the `ifconfig -a` command.

   If it does not, contact the router's administrator to verify that the router is advertising a prefix on the link.

   If the link is a configured tunnel and the router is not advertising an address prefix, manually define one for the tunnel by using the TCPIP$IP6_SETUP utility.

3. Contact the administrator for the remote system to verify that the system is up and running, that it is configured correctly for IPv6, and that the IPv6 address on its interface is the same as the address you are using.

   If the address is different, check your system's TCPIP$ETC:TCPIP$IPNODES.DAT file, or have the administrator for the remote system check the DNS entry.

4. Verify that there is a default route (with `U` and `G` flags set) to a router on the network by issuing the `netstat -rf inet6` command. If there is no default route, contact the router administrator to check whether the router is advertising itself as a default router.

   Also, check other routers to see whether your messages are being directed on the wrong path.

5. Trace the path to the off-link node by using the `traceroute` command.

Frequently dropped packets might indicate either network congestion or an intermittent routing problem. To determine the cause, do the following:

1. Verify connectivity between your system and an on-link router by using the `ping` command.
2. Trace the path to the off-link node by using the `traceroute` command.

### 5.4.5 Your Node Is Unreachable

If someone reports a problem reaching your node from another node, perform the following steps:

1. Verify that their node is reachable by issuing the `ping` command.

   If the command fails, follow the steps in Section 5.4.3 for an on-link node or Section 5.4.4 for an off-link node.

2. If they are using a name from the DNS database, verify that the address for your node in the DNS database matches one of the addresses configured on your system's interfaces.

   Use the `nslookup -type=AAAA node-name` command to retrieve the address from DNS and the `ifconfig -a` command to display addresses for your system.

3. If they are using an address defined in their local host file TCPIP$ETC:TCPIP$IPNODES.DAT, use the `ifconfig -a` command to compare that address with the addresses configured on your system's interfaces.

### 5.4.6 Connection Is Not Accepted

If a remote node is not configured to accept a connection from your application, the following message might appear:

```
connection refused
```

Verify that TCP/IP Services has been correctly configured on the remote node to accept connections.

Contact the administrator for the remote node and ask whether the correct socket-based service definitions are defined in the TCPIP$SERVICES.DAT file. Check whether the service has IPv6 enabled.

### 5.4.7 Connection Terminates

If the connection terminates abnormally or a network application appears to hang, perform the following steps:

1. Verify that there is network connectivity to the remote node by using the `ping` command immediately after the failure.

   If the `ping` command fails or shows a high rate of packet loss, follow the steps in either Section 5.4.3 for on-link nodes, or in Section 5.4.4 for off-link nodes.

2. If your application transfers a large amount of data over the network, verify whether large or fragmented messages are being handled correctly by using the `ping -s 2000 nodename` command.

   If the `ping` command fails, trace the path to the remote node with 1200-byte packets by using the `traceroute nodename 1200` command. All IPv6 links should support message sizes of at least 1280 bytes. This command might show the location of the problem in the network.

3. Run the application with different client and server nodes located on different links in the network.

## 5.5 Solving IPv6 Router Problems

This section describes problems with IPv6 routers.

### 5.5.1 IPv6 Process Is Not Running

Verify that the TCPIP$IP6RTRD process is running by issuing the following command:

```
$ SHO SYS /PROCESS=TCPIP$IP6RTRD
```

If the process is not running, start IPv6 with the following command:

```
$ @SYS$STARTUP:TCPIP$STARTUP.COM
```

This creates the IPv6 interfaces, brings them up, and starts the TCPIP$IP6RTRD process.

### 5.5.2 Host Is Unknown

If a remote host is not known, the following message appears:

```
unknown host
```

Perform the following steps:

1. Check whether the user is using a valid host name to reach the remote host.
2. Check if the remote host is in another name domain and whether the user specified the full domain name.
3. If your site uses the BIND name service for name-to-address translation, make sure the database contains an entry for the remote host.

   If it does not, edit TCPIP$ETC:TCPIP$IPNODES.DAT file to add the host.
4. If you are using a BIND server to search the BIND database for name-to-address translation, make sure the resolver is pointing to a valid BIND server. If your name server is on the local host, make sure that the BIND server is running. See the *Compaq TCP/IP Services for OpenVMS Management* guide for additional information about setting up your BIND environment.

### 5.5.3 On-Link Node Is Unreachable

If an on-link node is not reachable, one of the following messages can appear:

```
host is unreachable
network is unreachable
timeout
```

Verify that an on-link node or router is reachable by using the `ping` command. If the command fails or if packets are frequently dropped, complete the following steps:

1. If the node is attached to a LAN, check the data link counters by using the LANCP SHO DEVICE *device* /COUNTERS command. Problems with the counters and their possible causes are as follows:
   - Zero blocks sent or received can indicate a network hardware failure or a wiring problem.
   - High collision rates can indicate an improperly wired network or a node that is sending excessive message traffic.
   - Data overrun and buffer unavailable errors indicate your system is misconfigured.
2. If the data link counters are okay, check the IPv6 and ICMPv6 counters with the `netstat -p ipv6` and `netstat -p ipv6 -icmp` commands, respectively. Problems with the counters and their possible causes are as follows:
   - Packets discarded because of errors, or errors resulting from ICMP errors, indicate that another node is generating invalid messages. Other counters show more specific information.
   - Allocation errors can indicate excessive message traffic, a misconfigured system, or a program that repeatedly allocates memory without freeing it.

3. Verify that IPv6 network interfaces exist, are up, and have `inet6` addresses by using the `ifconfig -a` command. If they do not have `inet6` addresses, check the configuration file TCPIP$INET6_CONFIG.DAT. Run the TCPIP$IP6_SETUP utility to correct any errors.
4. Contact the system administrator for the adjacent on-link node and verify that the on-link node is up and running, that it is configured correctly for IPv6, and that the address you are using is enabled on the node's interface.
5. If IPv4 is configured on both systems, issue the `ping` command to the on-link node's IPv4 address. If the command succeeds, verify the IPv6 configuration on both systems. If the command fails, see the appropriate troubleshooting manuals.
6. Issue the `ping` command to other nodes on the link to determine whether the failure is confined to one node or whether it extends to multiple nodes. Partial connectivity might indicate a faulty network device or cable on the link.
7. If the link is a configured tunnel, do the following:
   a. Verify the tunnel source and destination addresses by using the `ifconfig -a` command. Contact the administrator for the tunnel destination node and verify that your source and destination addresses match the destination and source addresses on that node.
   b. Issue the `ping` command to the tunnel destination address. If the command fails, see the *Compaq TCP/IP Services for OpenVMS Management* guide for more information.

### 5.5.4 Off-Link Node Is Unreachable

If an off-link node is not reachable, the following messages appear:

```
host is unreachable
network is unreachable
timeout
```

Verify that an off-link node is reachable by issuing the `ping` command.

If there is 100% packet loss, perform the following steps:

1. Verify connectivity between your system and an on-link router by using the `ping` command.

   If the command fails or shows frequently dropped packets, follow the steps in Section 5.5.3.
2. Verify that the interface over which you are sending messages has a global or site-local unicast address enabled by using the `ifconfig -a` command.

   If it does not, check the prefixes defined in the SYS$SYSTEM:TCPIP$IP6RTRD.CONF file. Run the TCPIP$IP6_SETUP utility to correct any errors.
3. Contact the administrator for the remote system to verify that the system is up and running, that it is configured correctly for IPv6, and that the IPv6 address on its interface is the same as the address you are using.

   If the address is different, check your system's TCPIP$ETC:TCPIP$IPNODES.DAT file, or have the remote system administrator check the DNS entry.

4. Verify that there is a default route (with U and G flags set) to a router on the network by issuing the `netstat -rf inet6` command.

   If the route is missing or incorrect, check the routes and the address prefixes in the SYS$SYSTEM:TCPIP$IP6RTRD.CONF file.

   If your site uses RIPng, verify that RIP is enabled in the SYS$SYSTEM:TCPIP$IP6RTRD.CONF file. If it is, contact the administrator of the next router to verify that RIP is enabled.

5. Trace the path to the off-link node by using the `traceroute` command.

Frequently dropped packets indicate either network congestion or an intermittent routing problem.

To determine the cause, do the following:

1. Verify connectivity between your system and an on-link router by using the `ping` command.
2. Trace the path to the off-link node by using the `traceroute` command.

### 5.5.5 On-Link Node Addresses Are Not Configured

IPv6 hosts generate their global and site-local unicast addresses automatically by using address prefixes provided by a router on the link. If an on-link node cannot autoconfigure its addresses, perform the following steps:

1. Verify that the host is reachable from your router by using the `ping` command and specifying the host's link-local address. If the command fails or shows a high rate of packet loss, follow the steps in Section 5.5.3.
2. Edit the SYS$SYSTEM:TCPIP$IP6RTRD.CONF file and verify that the router is configured to advertise the correct prefixes and that the timers are reasonable. See Chapter 2 and Appendix B for more information.

### 5.5.6 Router Does Not Forward Messages

If another network user reports that message transmission appears to be failing at your router, perform the following steps:

1. Obtain the source and destination addresses of the message that your router is not forwarding. Then verify that your router can reach each node by using the `ping` command.

   If the command fails or shows a high rate of packet loss, follow the steps in Section 5.5.3 for on-link nodes, or in Section 5.5.4 for off-link nodes.

2. If your router is running the RIPng protocol, verify that the IPv6 router process is running by issuing the following command:

   ```
   $ SHO SYS /PROCESS=TCPIP$IP6RTRD
   ```

   If the process is running, edit the SYS$SYSTEM:TCPIP$IP6RTRD.CONF file and verify that the RIPng protocol is enabled on each IPv6 link. If it is not, your node may not be propagating routes correctly.

3. Make sure that you are not using manual routes on some interfaces and RIPng routes on other interfaces. Manual routes defined in the TCPIP$ROUTE.DAT file do not get propagated to other routers with RIPng.

### 5.5.7 Your Node Is Unreachable

If someone reports a problem reaching your node from another node, perform the following steps:

1. Verify that their node is reachable by issuing the `ping` command.

   If the command fails, follow the steps in Section 5.5.3 for an on-link node, or Section 5.5.4 for an off-link nodes.

2. If they are using a name from the DNS database, verify that the address for your node in the DNS database matches one of the addresses configured on your system's interfaces.

   Use the `nslookup -type=AAAA node-name` command to retrieve the address from DNS; use the `ifconfig -a` command to display addresses for your system.

3. If they are using an address defined in their local host file, compare that address with the addresses configured on your system's interfaces by using the `ifconfig -a` command.

### 5.5.8 Connection Is Not Accepted

If a remote node is not configured to accept a connection from your application, the following message might appear:

```
connection refused
```

Verify that TCP/IP Services has been correctly configured on the remote node to accept connections.

Contact the administrator for the remote node and ask whether the correct socket-based service definitions are defined in the TCPIP$SERVICES.DAT file. Check whether the service has IPv6 enabled.

### 5.5.9 Connection Terminates

If the connection terminates abnormally or if a network application appears to hang, perform the following steps:

1. Verify that there is network connectivity to the remote node by using the `ping` command immediately after the failure.

   If the `ping` command fails or shows a high rate of packet loss, follow the steps in Section 5.5.3 for an on-link node, or in Section 5.5.4 for an off-link node.

2. If your application transfers a large amount of data over the network, verify that large or fragmented messages are being handled correctly by using the `ping -s 2000 nodename` command.

   If the `ping` command fails, trace the path to the remote node with 1200-byte packets by using the `traceroute nodename 1200` command. All IPv6 links should support message sizes of at least 1280 bytes. This command might show the location of the problem in the network.

3. Run the application with different client and server nodes located on different links in the network.

# 6
# Application Interface to Sockets

The TCP/IP Services for OpenVMS programming interface supports the Berkeley Software Distribution (BSD) socket programming interface. It also supports the basic sockets interface extensions for Internet Protocol Version 6 (IPv6) as defined in RFC 2553. The basic syntax of socket functions remains the same. Existing IPv4 applications will continue to operate as before, and IPv6 applications can interoperate with IPv4 applications.

TCP/IP Services for OpenVMS provides Internet domain support for the address family AF_INET and AF_INET6.

This chapter describes the following aspects of the API:

- Socket interface
- Interface identification
- IPv6 multicast datagrams
- Socket options
- Library functions
- Guidelines for compiling and linking

## 6.1 Socket Interface

The IPv6 socket interface incorporates the following changes:

- New address family: AF_INET6
- New protocol family: PF_INET6
- New address structure: `in6_addr`

  ```
  struct in6_addr {
        uint8_t s6_addr[16]:
    }
  ```

  The address is stored in network byte order as an array of sixteen 8-bit elements.

- Revised socket address structure: `sockaddr_in6`

  If the _SOCKADDR_LEN symbol is defined in an application, the following BSD Version 4.4 structure is used:

```
struct sockaddr_in6 {
        uint8_t sin6_len;
        sa_family_t sin6_family;
        in_port_t sin6_port;
        uint32_t sin6_flowinfo;
        struct in6_addr sin6_addr;
        uint32_t sin6_scope_id;
    };
```

**Note**

BSD Version 4.3 will be supported in a subsequent release.

- New wildcard address, defined in network byte order. The address has the following forms:
  - A global variable, `in6addr_any`, that is an `in6_addr` structure.
  - A symbolic constant, IN6ADDR_ANY_INIT, that can be used to initialize an `in6_addr` structure only when it is declared.
- New loopback address, defined in network byte order. The address has the following forms:
  - A global variable, `in6addr_loopback`, that is an `in6_addr` structure.
  - A symbolic constant, IN6ADDR_LOOPBACK_INIT, that can be used to initialize an `in6_addr` structure only when it is declared.

The basic syntax of socket functions remains the same. Existing IPv4 applications will continue to operate as before. IPv6 applications can interoperate with IPv4 applications.

## 6.2 Interface Identification

To identify the interface on which a datagram is received, on which a datagram is to be sent, and on which a multicast group is joined, the API uses a small, positive integer called an **interface index**. The kernel assigns this integer to an interface when the interface is initialized.

The API defines the following new functions:

| Function | Description |
|---|---|
| `if_nametoindex` | Maps an interface name to its corresponding index. |
| `if_indextoname` | Maps an interface index to its corresponding name. |
| `if_nameindex` | Returns an array of all interface names and indexes. |
| `if_freenameindex` | Frees dynamic memory allocated by `if_nameindex` to the array of interface names and indexes. |

### 6.2.1 if_nametoindex Function

The `if_nametoindex` function has the following syntax:

```
#include <net/if.h>

unsigned int if_nametoindex(
        const char *ifname );
```

If the interface does not exist, the function returns 0 and sets errno to ENXIO. If a system error occurs, the function returns 0 and sets errno to an appropriate value.

### 6.2.2 if_indextoname Function

The if_indextoname function has the following syntax:

```
#include <net/if.h>

char *if_indextoname(
        unsigned int ifindex,
        char *ifname );
```

The **ifname** argument points to a buffer that is IFNAMSIZ bytes in length (IFNAMSIZ is defined in TCPIP$EXAMPLES:IF.H). If an interface name is found, it is returned in the buffer. If no interface name corresponds to the specified index, the function returns NULL and sets errno to ENXIO. If a system error occurs, the function returns NULL and sets errno to an appropriate value.

### 6.2.3 if_nameindex Function

The if_nameindex function has the following syntax:

```
#include <net/if.h>
struct if_nameindex *if_nameindex( void );
```

The following if_nameindex structure must also be defined (by including (TCPIP$EXAMPLES:IF.H) prior to the call to if_nameindex:

```
struct if_nameindex {
   unsigned int   if_index;
   char           *if_name;
};
```

The if_nameindex function dynamically allocates memory for an array of if_nameindex structures, one structure for each interface. A structure with an if_index value of 0 and a NULL if_name value indicates the end of the array. If an error occurs, the function returns a NULL pointer and sets errno to an appropriate value. To free the memory allocated by this function, use the if_freenameindex function.

### 6.2.4 if_freenameindex Function

The if_freenameindex function has the following syntax:

```
#include <net/if.h>

void if_freenameindex(
        struct if_nameindex *ptr );
```

The if_freenameindex function frees dynamic memory that was allocated by the if_nameindex function. The argument to this function is the pointer that was returned by the if_nameindex function.

# 6.3 IPv6 Multicast Datagrams

## 6.3.1 Sending IPv6 Multicast Datagrams

To send IPv6 multicast datagrams, an application indicates the multicast group to send to by specifying an IPv6 multicast address in a `sendto` system call. The system maps the specified IPv6 destination address to the appropriate Ethernet or FDDI multicast address prior to transmitting the datagram.

An application can explicitly control multicast options with arguments to the `setsockopt` system call. The following options can be set by an application using the `setsockopt` system call:

- Hop limit (IPV6_MULTICAST_HOPS)
- Multicast interface (IPV6_MULTICAST_IF)
- Disabling loopback of local delivery (IPV6_MULTICAST_LOOP)

**Note**

The examples here illustrate how to use the `setsockopt` options that apply to IPv6 multicast datagrams only.

The IPV6_MULTICAST_HOPS option to the `setsockopt` system call allows an application to specify a value between 0 and 255 for the hop limit field.

Multicast datagrams with a hop limit value of 0 restrict distribution of the multicast datagram to applications running on the local host. Multicast datagrams with a hop limit value of 1 are forwarded only to hosts on the local link. If a multicast datagram has a hop limit value greater than 1 and a multicast router is attached to the sending host's network, multicast datagrams can be forwarded beyond the local link. Multicast routers forward the datagram to known networks that have hosts belonging to the specified multicast group. The hop limit value is decremented by each multicast router in the path. When the hop limit value is decremented to 0, the datagram is not forwarded further.

The following example shows how to use the IPV6_MULTICAST_HOPS option to the `setsockopt` system call:

```
u_char hops;
hops=2;

if (setsockopt(sock, IPPROTO_IPV6, IPV6_MULTICAST_HOPS, &hops,
                     sizeof(hops)) < 0)
                     perror("setsockopt: IPV6_MULTICAST_HOPS error");
```

A datagram addressed to an IPv6 multicast address is transmitted from the default network interface unless the application specifies that an alternate network interface is associated with the socket. The default interface is determined by the interface associated with the default route in the kernel routing table or by the interface associated with an explicit route, if one exists. Using the IPV6_MULTICAST_IF option to the `setsockopt` system call, an application can specify a network interface other than that specified by the route in the kernel routing table.

The following example shows how to use the IPV6_MULTICAST_IF option to the setsockopt system call to specify an interface other than the default:

```
u_int if_index = 1;
.
.
.
if (setsockopt(sock, IPPROTO_IPV6, IPV6_MULTICAST_IF, &if_index,
                                 sizeof(if_index)) < 0)
     perror ("setsockopt: IPV6_MULTICAST_IF error");
  else
     printf ("new interface set for sending multicast datagrams\n");
```

The **if_index** parameter specifies the interface index of the desired interface, or specifies 0 to select a default interface. You can use the if_nametoindex routine to find the interface index.

If a multicast datagram is sent to a group that has the sending node is a member, a copy of the datagram is, by default, looped back by the IP layer for local delivery. The IPV6_MULTICAST_LOOP option to the setsockopt system call allows an application to disable this loopback delivery.

The following example shows how to use the IPV6_MULTICAST_LOOP option to the setsockopt system call:

```
u_char loop=0;
if (setsockopt( sock, IPPROTO_IPV6, IPV6_MULTICAST_LOOP, &loop,
         sizeof(loop)) < 0)
         perror("setsockopt: IPV6_MULTICAST_LOOP error");
```

If the value of **loop** is 0, loopback is disabled; if the value of **loop** is 1, loopback is enabled. For performance reasons, you should disable the default, unless applications on the same host must receive copies of the datagrams.

### 6.3.2 Receiving IPv6 Multicast Datagrams

Before a node can receive IPv6 multicast datagrams destined for a particular multicast group other than the All Nodes group, an application must direct the node to become a member of that multicast group.

This section describes how an application can direct a node to add itself to and remove itself from a multicast group.

An application can direct the node it is running on to join a multicast group by using the IPV6_JOIN_GROUP option to the setsockopt system call:

```
struct ipv6_mreq imr6;
.
.
.
imr6.ipv6mr_interface = if_index;
if (setsockopt( sock, IPPROTO_IPV6, IPV6_JOIN_GROUP,
     (char *)&imr6, sizeof(imr6)) < 0)
     perror("setsockopt: IPV6_JOIN_GROUP error");
```

The **imr6** parameter has the following structure:

```
structipv6_mreq {
        struct in6_addr ipv6mr_multiaddr; /* IP multicast address of
group */
        unsigned int ipv6mr_interface; /* local interface index*/
        };
```

Each multicast group membership is associated with a particular interface. It is possible to join the same group on multiple interfaces. The *ipv6mr_interface* variable can be specified with a value of 0, which allows an application to choose the default multicast interface. Alternatively, specifying one of the host's local interfaces allows an application to select a particular multicast-capable interface. The maximum number of memberships that can be added on a single socket is subject to the IPV6_MAX_MEMBERSHIPS value, which is defined in the <netinet/in.h> header file.

To drop membership from a particular multicast group, use the IPV6_LEAVE_GROUP option to the setsockopt system call:

```
struct ipv6_mreq imr6;
if (setsockopt( sock, IPPROTO_IPV6, IPV6_LEAVE_GROUP, &imr6,
     sizeof(imr6)) < 0)
     perror("setsockopt: IPV6_LEAVE_GROUP error");
```

The **imr6** parameter contains the same structure values used for adding membership.

If multiple sockets request that a node join a particular multicast group, the node remains a member of that multicast group until the last of those sockets is closed.

To receive multicast datagrams sent to a specific UDP port, the receiving socket must have bound to that port using the bind system call. More than one process can receive UDP datagrams destined for the same port if the bind system call is preceded by a setsockopt system call that specifies the SO_REUSEPORT option.

Delivery of IP multicast datagrams to SOCK_RAW sockets is determined by the protocol type of the destination.

## 6.4 Socket Options

To support IPv6, the setsockopt and getsockopt functions recognize a new IPPROTO_IPV6 level.

In addition, the setsockopt function defines the following new options:

| Option | Function |
|---|---|
| IPV6_UNICAST_HOPS | Sets the hop limit for all subsequent unicast packets sent on a socket.<br>You can also use this option with the getsockopt function to determine the current hop limit for a socket. |
| IPV6_MULTICAST_IF | Sets the interface to use for outgoing multicast packets. |
| IPV6_MULTICAST_HOPS | Sets the hop limit for outgoing multicast packets. |
| IPV6_MULTICAST_LOOP | Controls whether to deliver outgoing multicast packets back to the local application. |
| IPV6_JOIN_GROUP | Joins a multicast group on the specified interface. |
| IPV6_LEAVE_GROUP | Leaves a multicast group on the specified interface. |

See RFC 2553 for more information on these socket options.

## 6.5 Library Functions

The following are the changes to the library functions to accommodate the IPv6 enhancements:

- Node name to address translation
- Address to node name translation
- Address conversion functions
- Address-testing macros

### 6.5.1 Node Name to Address Translation Functions

The following resolver options are available for node name to address translation:

| Option | Function |
|---|---|
| gethostbyname | Existing function that returns IPv4 addresses. |
| getaddrinfo | New protocol-independent function for mapping names to addresses. |
| freeaddrinfo | New function that returns addrinfo structures and dynamic storage to the system. |

The following sections describe these changes in detail.

#### 6.5.1.1 getaddrinfo Function

The getaddrinfo function has the following syntax:

```
#include <netdb.h>

int getaddrinfo(
          const char *nodename,
          const char *servname,
          const struct addrinfo *hints,
          struct addrinfo **res);
```

**Parameters**

- **nodename**

  Points to a network node name, alias, or numeric host address (for example, an IPv4 dotted-decimal address or an IPv6 hexadecimal address). This is a null-terminated string or NULL. NULL means the service location is local to the caller. The **nodename** and **servname** parameters cannot both be NULL.

- **servname**

  Points to a network service name or port number. This is a null-terminated string or NULL; NULL returns network-level addresses for the specified nodename). The **nodename** and **servname** parameters cannot both be NULL.

- **hints**

  Points to an addrinfo structure that contains information about the type of socket the caller supports. The <netdb.h> header file defines the addrinfo structure. This is an optional parameter.

- **res**

  Points to a linked list of one or more addrinfo structures.

**Description**

The getaddrinfo( ) routine takes a service location (**nodename**) or a service name (**servname**), or both, and returns a pointer to a linked list of one or more structures of type addrinfo. Its members specify data obtained from either the local hosts database TCPIP$ETC:IPNODES.DAT file, local TCPIP$HOSTS.DAT file, or one of the files distributed by DNS/BIND.

The <netdb.h> header file defines the addrinfo structure.

If you specify the **hints** parameter, all addrinfo structure members other than the following members must be zero or a NULL pointer:

- ai_flags

  Controls the processing behavior of getaddrinfo. See Table 6–1 for a complete description of the flags.

- ai_family

  Specifies to return addresses for use with a specific protocol family.

  - If you specify a value of AF_UNSPEC, the routine returns addresses for any protocol family that can be used with **nodename** or **servname**.
  - If the value is not AF_UNSPEC and ai_protocol is not zero, the routine returns addresses for use only with the specified protocol family and protocol.
  - If the application handles only IPv4, set this member of the **hints** structure to PF_INET.
  - If ai_family is set to PF_INET6, the function looks only in the TCPIP$ETC:IPNODES.DAT file and the lookup fails in the BIND database.

- ai_socktype

  Specifies a socket type for the given service. If you specify a value of 0, you will accept any socket type. This resolves the service name for all socket types and returns all successful results.

- ai_protocol

  Specifies a network protocol. If you specify a value of 0, you will accept any protocol. If the application handles only TCP, set this member to IPPROTO_ TCP.

If the **hints** parameter is a NULL pointer, this is identical to passing an addrinfo structure that has been initialized to zero, and the the ai_family member set to AF_UNSPEC.

Table 6–1 describes the ai_flags member values.

**Table 6–1 ai_flags Member Values**

| Flag Value | Description | |
|---|---|---|
| AI_V4MAPPED | **If af value is AF_NET:** | **If af value is AF_INET6:** |
| | Ignored. | Searches for AAAA records.<br>The lookup sequence is LOCAL host database, TCPIP$ETC:IPNODES.DAT, BIND database.<br>If AAAA records found, returns IPv6 records.<br>If no AAAA records found, searches for A records.<br>If A records found, returns IPv4-mapped IPv6 addresses.<br>If no A records found, returns a NULL pointer. |
| AI_ALL \| AI_ V4MAPPED | **If af value is AF_NET:** | **If af value is AF_INET6:** |
| | Ignored. | Searches for AAAA records.<br>The lookup sequence is LOCAL host database, TCPIP$ETC:IPNODES.DAT, BIND database.<br>If AAAA records found, returns IPv6 records.<br>If no AAAA records found, searches for A records.<br>If A records found, returns IPv4-mapped IPv6 addresses.<br>If no A records found, returns a NULL pointer. |
| AI_CANONNAME | If the **nodename** parameter is not NULL, the function searches for the specified node's canonical name.<br>Upon successful completion, the ai_canonname member of the first addrinfo structure in the linked list points to a null-terminated string containing the canonical name of the specified node name.<br>If the canonical name is not available, the ai_canonname member refers to the **nodename** parameter or to a string with the same contents.<br>The ai_flags field contents are undefined. | |
| AI_NUMERICHOST | A non-NULL node name string must be a numeric host address string.<br>Resolution of the service name is not performed. | |

(continued on next page)

**Table 6–1 (Cont.) ai_flags Member Values**

| Flag Value | Description |
|---|---|
| AI_NUMERICSERV | A non-NULL service name string must be a numeric port string.<br>Resolution of the service name is not performed. |
| AI_PASSIVE | Returns a socket address structure that your application can use in a call to `bind( )`.<br>If the **nodename** parameter is a NULL pointer, the IP address portion of the socket address structure is set to INADDR_ANY (for an IPv4 address) or IN6ADDR_ANY_INIT (for an IPv6 address).<br>If not set, returns a socket address structure that your application can use to call `connect( )` (for a connection-oriented protocol) or either `connect( )`, `sendto( )`, or `sendmsg( )` (for a connectionless protocol). If the **nodename** parameter is a NULL pointer, the IP address portion of the socket address structure is set to the loopback address. |

You can use the flags in any combination to achieve finer control of the translation process. The AI_ADDRCONFIG flag is typically used in combination with other flags to modify the search based on the source address or addresses configured on the system. The following table describes how the AI_ADDRCONFIG flags works by itself.

| Flag Value | If `af` Value is AF_NET | If `af` Value is AF_INET6 |
|---|---|---|
| AI_ADDRCONFIG | If an IPv4 source address is configured, searches for A records. | If an IPv6 source address is configured, searches for AAAA records. |

Most applications will want to use the combination of the AI_ADDRCONFIG and AI_V4MAPPED flags to control their search. To simplify this for the programmer, the AI_DEFAULT symbol, which is a logical OR of AI_ADDRCONFIG and AI_V4MAPPED, is defined. The following table describes how AI_DEFAULT directs the search.

| Flag Value | If `af` Value is AF_NET | If `af` Value is AF_INET6 |
|---|---|---|
| AI_DEFAULT | Searches for A records only if an IPv4 source address is configured on the system. If found, returns IPv4 addresses. If not, returns a NULL pointer. | Searches for AAAA records only if an IPv6 source address is configured on the system. If found, returns IPv6 addresses. If not and if an IPv4 address is configured on the system, searches for A records. If found, returns IPv4-mapped IPv6 addresses. If not, returns a NULL pointer. |

These flags are defined in `<netdb.h>`.

**addrinfo Structure Processing**

Upon successful return, getaddrinfo returns a pointer to a linked list of one or more addrinfo structures. The application can process each addrinfo structure in the list by following the ai_next pointer until a NULL pointer is encountered. In each returned addrinfo structure, the ai_family, ai_socktype, and ai_protocol members are the corresponding arguments for a call to the socket( ) function. The ai_addr member points to a filled-in socket address structure whose length is specified by the ai_addrlen member.

**Return values**

Upon successful completion, the getaddrinfo( ) function returns a 0 (zero); upon failure, it returns a nonzero value.

## 6.5.2 Address to Node Name Translation Functions

The following functions are available for address to node name translation:

| Option | Function |
|---|---|
| gethostbyaddr | Existing function that returns a node name for an IPv4 address. |
| getnameinfo | New protocol-independent function for mapping addresses to names. |
| freeaddrinfo | New function that returns addrinfo structures and dynamic storage to the system. |

The following sections describe these changes.

### 6.5.2.1 getnameinfo Function

The getnameinfo function has the following syntax:

```
int getnameinfo(
        const struct sockaddr *sa,
        socklen_t salen,
        char *host,
        size_t hostlen,
        char **serv,
        size_t servlen,
        int flags );
```

**Parameters**

- **sa**

  Points either to a sockaddr_in structure (for IPv4) or to a sockaddr_in6 structure (for IPv6) that holds the IP address and port number.

- **salen**

  Specifies the length of either the sockaddr_in structure or the sockaddr_in6 structure.

- **node**

  Points to a buffer in which to receive the null-terminated network node name or alias corresponding to the address contained in the **sa**. A NULL pointer instructs the routine not to return a node name. The **node** parameter and **serv** parameter cannot both be zero.

- **nodelen**

  Specifies the length of the node buffer. A value of zero instructs the routine not to return a node name.

- **serv**

  Points to a buffer in which to receive the null-terminated network service name associated with the port number contained in sa. A NULL pointer instructs the routine not to return a service name. The **node** parameter and **serv** parameter cannot both be zero.

- **servlen**

  Specifies the length of the **serv** buffer. A value of zero instructs the routine not to return a service name.

- **flags**

  Specifies changes to the routine's default actions. By default, the routine searches for the fully qualified domain name of the node in the host's database and returns it. See Table 6–2 for a list of flag bits and their meanings.

**Description**

The `getnameinfo( )` routine looks up an IP address and port number in a `sockaddr` structure specified by **sa** and returns node name and service name text strings in the buffers pointed to by the **node** and **serv** parameters, respectively.

If the node name is not found, the routine returns the numeric form of the node address, regardless of the value of the **flags** parameter. If the service's name is not found, the routine returns the numeric form of the service's address (port number) regardless of the value of the **flags** parameter.

The application must provide buffers large enough to hold the fully qualified domain name and the service name, including the terminating null characters.

**Flag bits**

Table 6–2 describes the flag bits and, if set, their meanings.

**Table 6–2 Flag Bits**

| Flag Value | Description |
|---|---|
| NI_DGRAM | Specifies that the service is a datagram service (SOCK_DGRAM). The default assumes a stream service (SOCK_STREAM). This is required for the few ports (512-514) that have different services for UDP and TCP. |
| NI_NAMEREQD | Returns an error if the host name cannot be located in the host's database. |
| NI_NOFQDN | Searches the host's database and returns the node name portion of the fully qualified domain name for local hosts. |
| NI_NUMERICHOST | Returns the numeric form of the host's address instead of its name. Resolution of the host name is not performed. |

(continued on next page)

**Table 6–2 (Cont.) Flag Bits**

| Flag Value | Description |
|---|---|
| NI_NUMERICSERV | Returns the numeric form (port number) of the service address instead of its name. Resolution of the host name is not performed. |

The two NI_NUMERIC* flags are required to support the `-n` flag that many commands provide. All flags are defined in `<netdb.h>` header file.

**Return Values**

Upon successful completion, the `getnameinfo( )` function returns 0 (zero); upon failure, it returns a nonzero value.

#### 6.5.2.2 freeaddrinfo Function

This new function frees system resources used by an address information structure.

The `freeaddrinfo( )` routine frees one or more `addrinfo` structures and any dynamic storage associated with the structures. The process continues until the routine encounters a NULL `ai_next` pointer.

The `freeaddrinfo` function has the following syntax:

```
#include <netdb.h>

void freeaddrinfo(
          struct addrinfo *ai);
```

The **ai** parameter is a pointer to the `addrinfo` structure to be freed.

The `<netdb.h>` header file defines the `addrinfo` structure.

### 6.5.3 Address Conversion Functions

The following address conversion functions are new. They convert both IPv4 and IPv6 addresses.

| Option | Function |
|---|---|
| `inet_pton` | Converts an address in its standard text presentation form to its numeric binary form, in network byte order. |
| `inet_ntop` | Converts a numeric address to a text string suitable for presentation. |

#### 6.5.3.1 inet_pton Function

This function has the following syntax:

```
int inet_pton(
             int af,
             const char *src,
             void *dst);
```

**Parameters**

- **af**

  Specifies the address family. Valid values are AF_INET for an IPv4 address and AF_INET6 for an IPv6 address.

- **src**

  Points to the address text string to be converted.

- **dst**

  Points to a buffer that is to contain the numeric address.

**Description**

The inet_pton( ) function converts a text string to a numeric value in Internet network byte order.

- If the **af** parameter is AF_INET, the function accepts a string in the standard IPv4 dotted-decimal format:

  ```
  ddd.ddd.ddd.ddd
  ```

  In this format, *ddd* is a one- to three-digit decimal number between 0 and 255.

- If the **af** parameter is AF_INET6, the function accepts a string in the following format:

  ```
  x:x:x:x:x:x:x:x
  ```

  In this format, *x* is the hexadecimal value of a 16-bit piece of the address.

  IPv6 addresses can contain long strings of zero (0) bits. To make it easier to write these addresses, you can use double-colon characters (::) one time in an address to represent 1 or more 16-bit groups of zeros.

- For mixed IPv4 and IPv6 environments, the following format is also accepted:

  ```
  x:x:x:x:x:x:ddd.ddd.ddd.ddd
  ```

  In this format, *x* is the hexadecimal value of a 16-bit piece of the address, and *ddd* is a one- to three-digit decimal value between 0 and 255 that represents the IPv4 address. See RFC 2373 for more information about IPv6 addressing formats.

The calling application is responsible for ensuring that the buffer referred to by the **dst** parameter is large enough to hold the numeric address. AF_INET addresses require 4 bytes and AF_INET6 addresses require 16 bytes.

**Return values**

Upon successful completion, the inet_pton( ) function returns a 1. If the input string is neither a valid IPv4 dotted-decimal string nor a valid IPv6 address string, the function returns a 0. If any other error occurs, the function returns a -1.

**Errors**

If the inet_pton( ) routine call fails, errno is set to the following value:

| | |
|---|---|
| EAFNOSUPPORT | The address family specified in the **af** parameter is unknown. |

#### 6.5.3.2 inet_ntop Function

This function has the following syntax:

```
const char *inet_ntop(
                int af,
                const void *src,
                char *dst,
                size_t size);
```

**Parameters**

- **af**

  Specifies the address family. Valid values are AF_INET for an IPv4 address and AF_INET6 for an IPv6 address.

- **src**

  Points to a buffer that contains the numeric Internet address.

- **dst**

  Points to a buffer that is to contain the text string.

- **size**

  Specifies the size of the buffer pointed to by the **dst** parameter. For IPv4 addresses, the minimum buffer size is 16 octets; for IPv6 addresses, the minimum buffer size is 46 octets. The <netinet/in.h> header file defines the INET_ADDRSTRLEN and INET6_ADDRSTRLEN constants, respectively, for these values.

**Description**

The inet_ntop( ) function converts a numeric Internet address value to a text string.

**Return values**

Upon successful conversion, the inet_ntop( ) function returns a pointer to the buffer containing the text string. If the function fails, it returns a pointer to the buffer containing NULL.

### 6.5.4 Address-Testing Macros

Table 6–3 lists the currently defined address-testing macros and the return value for a valid test. To use these macros, include the following file in your application:

```
#include <net/in.h>
```

**Table 6–3 Summary of Address-Testing Macros**

| Macro | Return |
|---|---|
| IN6_IS_ADDR_UNSPECIFIED | True, if specified type. |
| IN6_IS_ADDR_LOOPBACK | True, if specified type. |
| IN6_IS_ADDR_MULTICAST | True, if specified type. |
| IN6_IS_ADDR_LINKLOCAL | True, if specified type. |
| IN6_IS_ADDR_SITELOCAL | True, if specified type. |

(continued on next page)

**Table 6–3 (Cont.) Summary of Address-Testing Macros**

| Macro | Return |
|---|---|
| IN6_IS_ADDR_V4MAPPED | True, if specified type. |
| IN6_IS_ADDR_V4COMPAT | True, if specified type. |
| IN6_IS_ADDR_MC_NODELOCAL | True, if specified scope. |
| IN6_IS_ADDR_MC_LINKLOCAL | True, if specified scope. |
| IN6_IS_ADDR_MC_SITELOCAL | True, if specified scope. |
| IN6_IS_ADDR_MC_ORGLOCAL | True, if specified scope. |
| IN6_IS_ADDR_MC_GLOBAL | True, if specified scope. |
| IN6_ARE_ADDR_EQUAL | True, if addresses are equal. |

## 6.6 Guidelines for Compiling and Linking IPv6 Applications

To compile an IPv6 application, you need to set up the following environment:

```
$ DEFINE DECC$SYSTEM_INCLUDE TCPIP$EXAMPLES:
$ DEFINE ARPA TCPIP$EXAMPLES:
$ DEFINE NET TCPIP$EXAMPLES:
$ DEFINE NETINET TCPIP$EXAMPLES:
$ DEFINE SYS TCPIP$EXAMPLES:
```

This is a temporary measure and is required until Compaq C has all the IPv6 include files.

The library functions described in this chapter are included in the Compaq TCP/IP Services for OpenVMS Version 5.1 kit.

# 7
# Porting Applications

This chapter describes the changes you must make in your application code to operate in an IPv6 networking environment.

- Name changes
- Structure changes
- Other changes

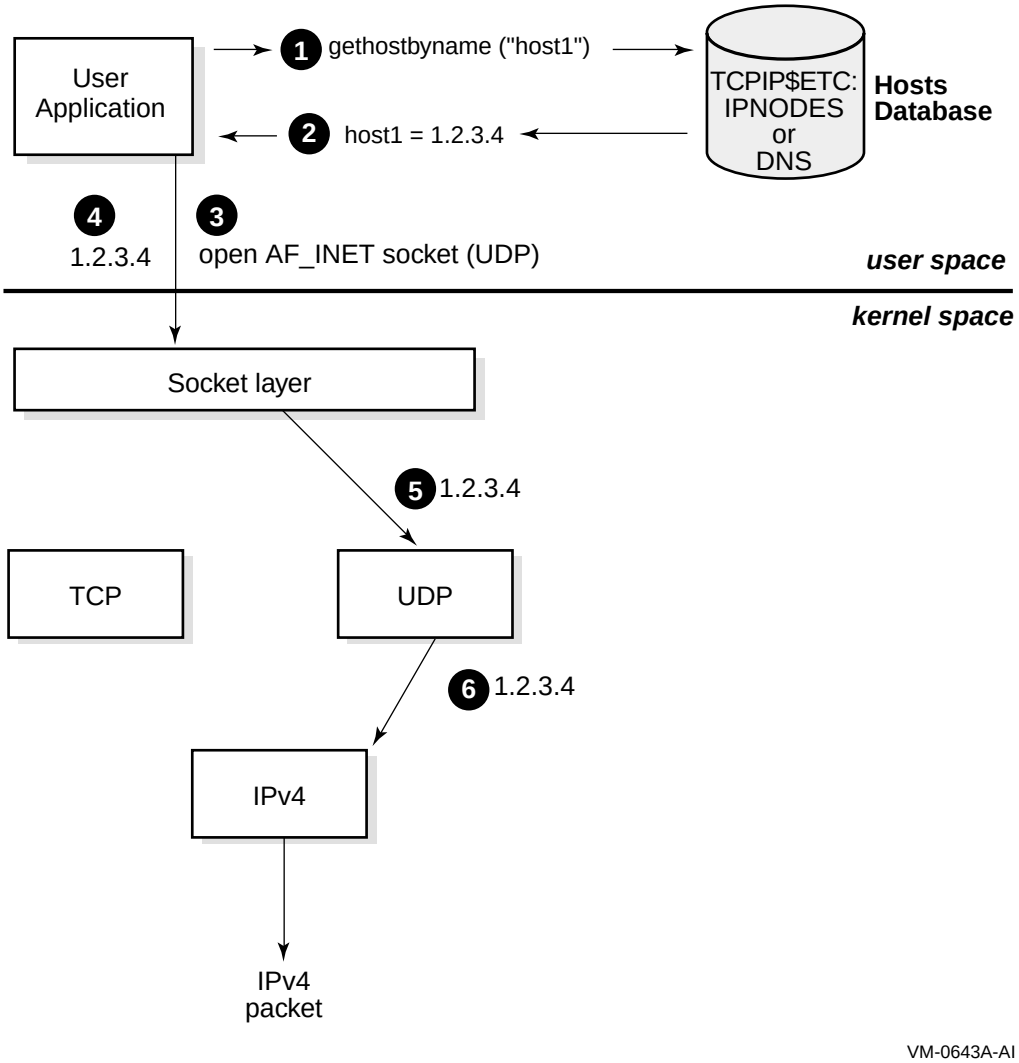
You can also use this information as guidelines for creating new IPv6-ready applications.

See RFC 2553, *Basic Socket Interface Extensions for IPv6*, for complete information on the changes to the BSD socket applications programming interface (API). The basic syntax of socket functions remains the same. Existing IPv4 applications will continue to operate as before, and IPv6 applications can interoperate with IPv4 applications.

## 7.1 Using AF_INET6 Sockets

At present, applications use AF_INET sockets for IPv4 communications. Figure 7–1 shows a sample sequence of events for an application that uses an AF_INET socket to send IPv4 packets.

**Figure 7–1 Using AF_INET Socket for IPv4 Communications**




1 Application calls `gethostbyname` and passes the host name, host1.

2 The search finds host1 in the hosts database and `gethostbyname` returns the IPv4 address 1.2.3.4.

3 The application opens an AF_INET socket.

4 The application sends information to the 1.2.3.4 address.

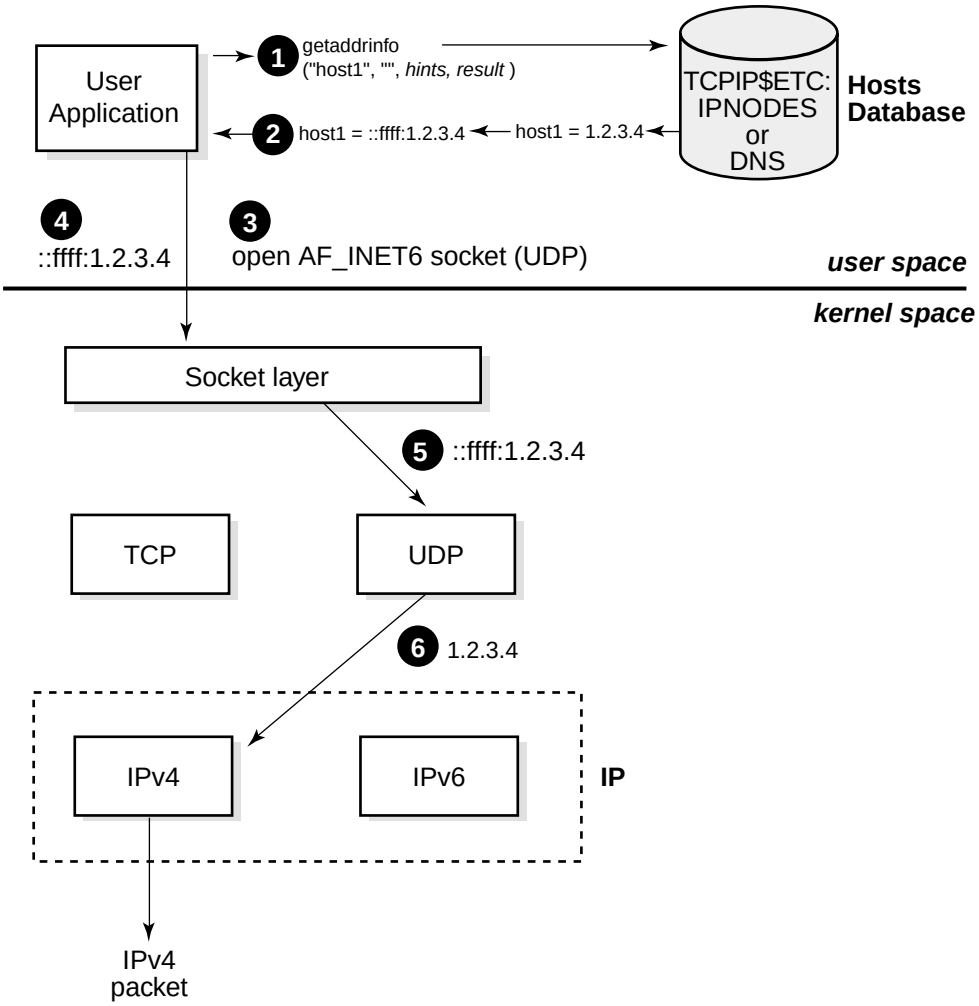
5 The socket layer passes the information and address to the UDP module.

6 The UDP module puts the 1.2.3.4 address into the packet header and passes the information to the IPv4 module for transmission.

Section 7.6.1.1 contains sample program code that demonstrates these steps.

You can use the AF_INET6 socket for both IPv6 and IPv4 communications. For IPv4 communications, create an AF_INET6 socket and pass it a `sockaddr_in6` structure that contains an IPv4-mapped IPv6 address (for example, `::ffff:1.2.3.4`). Figure 7–2 shows the sequence of events for an application that uses an AF_INET6 socket to send IPv4 packets.

**Figure 7–2 Using AF_INET6 Socket to Send IPv4 Communications**




VM-0644A-AI

1 Application calls `getaddrinfo` and passes the host name (host1), the AF_INET6 address family hint, and the AI_DEFAULT flag hint. The flag tells the function that if an IPv4 address is found for host1, return the address as an IPv4-mapped IPv6 address.

2 The search finds an IPv4 address, 1.2.3.4, for host1 in the hosts database and `getaddrinfo` returns the IPv4-mapped IPv6 address `::ffff:1.2.3.4`.

3 The application opens an AF_INET6 socket.

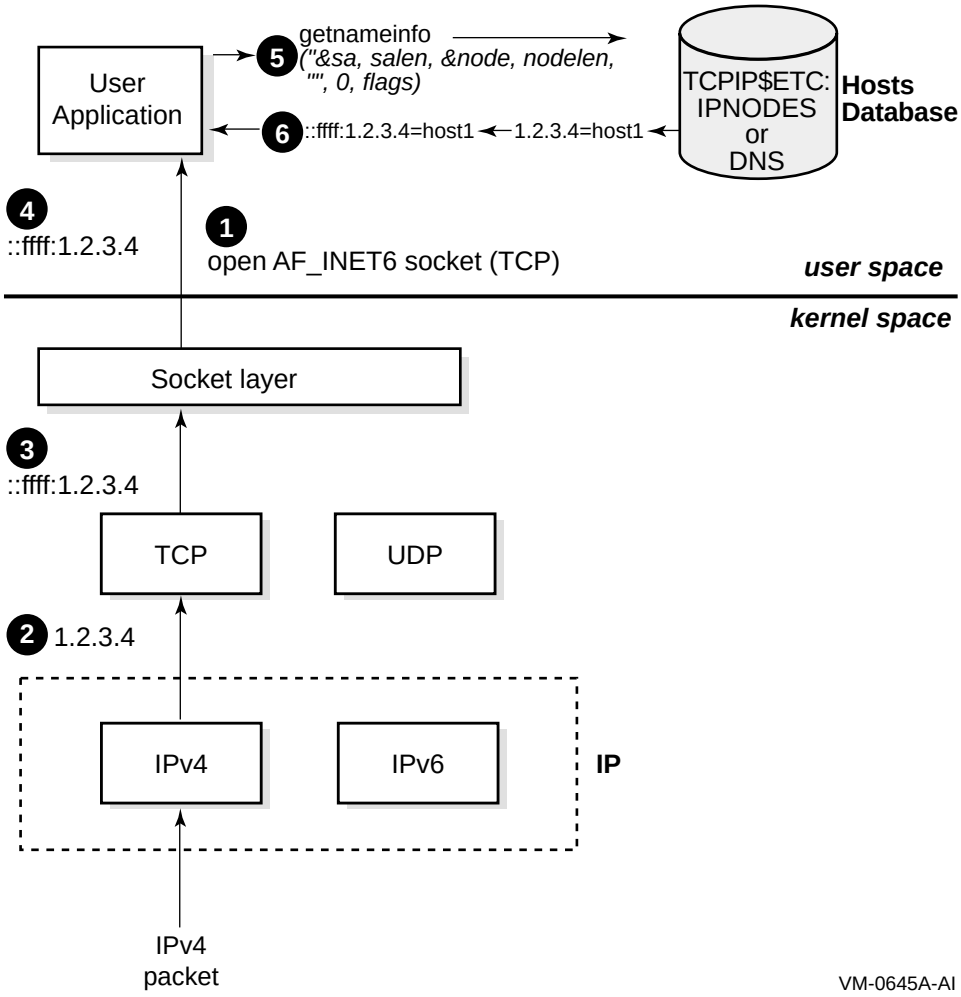
4 The application sends information to the `::ffff:1.2.3.4` address.

5 The socket layer passes the information and address to the UDP module.

6 The UDP module identifies the IPv4-mapped IPv6 address, puts the 1.2.3.4 address into the packet header, and passes the information to the IPv4 module for transmission.

AF_INET6 sockets can receive messages sent to either IPv4 or IPv6 addresses on the system. An AF_INET6 socket uses the IPv4-mapped IPv6 address format to represent IPv4 addresses. Figure 7–3 shows the sequence of events for an application that uses an AF_INET6 socket to receive IPv4 packets.

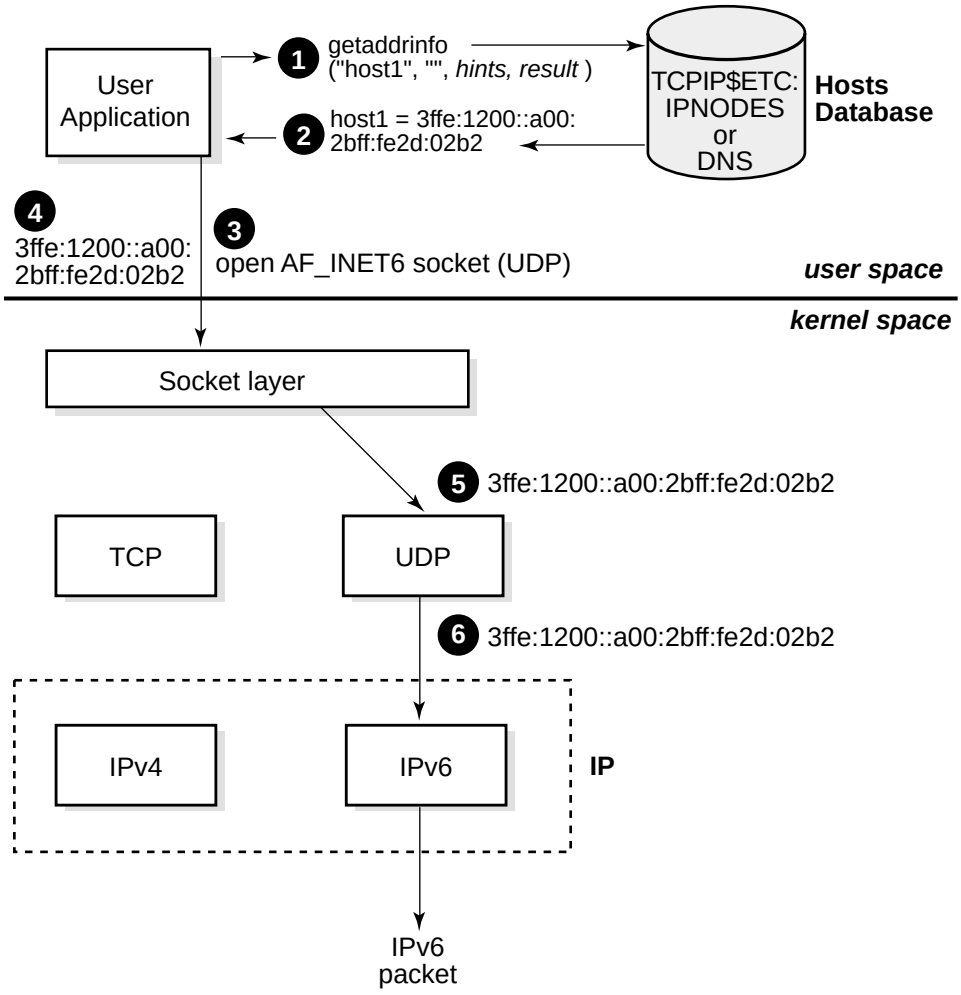
**Figure 7–3 Using AF_INET6 Socket to Receive IPv4 Communications**



1. The application opens an AF_INET6 socket, binds to it, and listens on it.
2. An IPv4 packet arrives and passes through the IPv4 module.
3. The TCP layer strips off the packet header and passes the information and the IPv4-mapped IPv6 address `::ffff:1.2.3.4` to the socket layer.
4. The application calls `accept` and retrieves the information from the socket.
5. The application calls `getnameinfo` and passes the `::ffff:1.2.3.4` address and the NI_NAMEREQD flag. The flag tells the function to return the host name for the address. See Table 6–2 for a description of the flag bits and their meanings.
6. The search finds the host name for the 1.2.3.4 address in the hosts database, and `getnameinfo` returns the host name.

For IPv6 communications, create an AF_INET6 socket and pass it a `sockaddr_in6` structure that contains an IPv6 address (for example, `3ffe:1200::a00:2bff:fe2d:02b2`). Figure 7–4 shows the sequence of events for an application that uses an AF_INET6 socket to send IPv6 packets.

**Figure 7–4 Using AF_INET6 Socket for IPv6 Communications**




1 Application calls `getaddrinfo` and passes the host name (host1), the AF_INET6 address family hint, and the AI_DEFAULT flag hint. The flag tells the function that if an IPv4 address is found for host1, to return it.

2 The search finds an IPv6 address for host1 in the hosts database, and `getaddrinfo` returns the IPv6 address `3ffe:1200::a00:2bff:fe2d:02b2`.

3 The application opens an AF_INET6 socket.

4 The application sends information to the `3ffe:1200::a00:2bff:fe2d:02b2` address.

5 The socket layer passes the information and address to the UDP module.

6 The UDP module identifies the IPv6 address and puts the `3ffe:1200::a00:2bff:fe2d:02b2` address into the packet header and passes the information to the IPv6 module for transmission.

Section 7.6.2.1 contains sample program code that demonstrates these steps.

The following sections show how to convert an existing AF_INET application to an AF_INET6 application that is capable of communicating over both IPv4 and IPv6.

## 7.2 Name Changes

Most of the changes required are straightforward and mechanical, though some may require a bit of code restructuring. For example, a routine that returns an int data type holding an IPv4 address may need to be modified to take as an extra parameter a pointer to an `in6_addr` into which it writes the IPv6 address. Table 7–1 summarizes the changes you must make to your application's code.

**Table 7–1 Name Changes**

| Search file for | Replace with | Comments |
|---|---|---|
| AF_INET | AF_INET6 | Replace with IPv6 address family macro. |
| PF_INET | PF_INET6 | Replace with IPv6 protocol family macro. |
| INADDR_ANY | in6addr_any | Replace with IPv6 global variable. |

## 7.3 Structure Changes

The structure names and field names have changed for the following structures:

- `in_addr`
- `sockaddr_in`
- `sockaddr`
- `hostent`

The following sections discuss these changes.

### 7.3.1 in_addr Structure

Applications that use the IPv4 `in_addr` structure must be changed to use the IPv6 `in6_addr` structure, as follows:

| IPv4 Structure | IPv6 Structure |
|---|---|
| `struct in_addr`<br>`unsigned int s_addr` | `struct in6_addr`<br>`uint8_t s6_addr` |

Make the following changes to your application, as needed:

1. Change the structure name `in_addr` to `in6_addr`.
2. Change the data type from `unsigned int` to `uint8_t` and the field name `s_addr` to `s6_addr`.

### 7.3.2 sockaddr Structure

Applications that use the generic socket address structure (sockaddr) to hold an AF_INET socket address (`sockaddr_in`) must be changed to use the AF_INET6 `sockaddr_in6` structure, as follows:

| AF_INET Structure | AF_INET6 Structure |
|---|---|
| `struct sockaddr` | `struct sockaddr_in6` |

Make the following change to your application, as needed:

1. Change structure name `sockaddr` to `sockaddr_in6`.

   For example, `sizeof(struct sockaddr)`.

   **Note**

   A `sockaddr_in6` structure is larger than a `sockaddr` structure.

### 7.3.3 sockaddr_in Structure

Applications that use the BSD Version 4.4 IPv4 `sockaddr_in` structure must be changed to use the IPv6 `sockaddr_in6` structure, as follows:

| IPv4 Structure | IPv6 Structure |
|---|---|
| `struct sockaddr_in` | `struct sockaddr_in6` |
| `unsigned char sin_len` | `uint8_t sin6_len` |
| `sa_family_t sin_family` | `sa_family_t sin6_family` |
| `in_port_t sin_port` | `int_port_t sin6_port` |
| `struct addr sin_addr` | `struct in6_addr sin6_addr` |

Make the following changes to your application, as needed:

1. Change structure name `sockaddr_in` to `sockaddr_in6`. Initialize the entire `sockaddr_in6` structure to zero after your structure declarations.
2. Change the data type `unsigned char` to `uint8_t` and the field name `sin_len` to `sin6_len`.
3. Change the field name `sin_family` to `sin6_family`.
4. Change the field name `sin_port` to `sin6_port`.
5. Change the field name `sin_addr` to `sin6_addr`.

### 7.3.4 hostent Structure

Applications that use the `hostent` structure must be changed to use the `addrinfo` structure, as follows:

| AF_INET Structure | AF_INET6 Structure |
|---|---|
| `struct hostent` | `struct addrinfo` |

Make the following change to your application, as needed:

1. Change the structure name `hostent` to `addrinfo`.

## 7.4 Function Call Changes

The names and parameters have changed for the following function calls:

- gethostbyaddr
- gethostbyname
- inet_ntoa
- inet_addr

The following sections discuss these changes.

### 7.4.1 gethostbyaddr Function Call

Applications that use the IPv4 gethostbyaddr function call must be changed to use the IPv6 getnameinfo function call, as follows:

| AF_INET Call | AF_INET6 Call |
|---|---|
| gethostbyaddr(*xxx*,4,AF_INET) | err=getnameinfo(&*sockaddr*,*sockaddr_len, node_name, name_len, service, service_len, flags*); |

Make the following change to your application, as needed:

1. Change the function name from gethostbyaddr to getnameinfo and provide a pointer to the socket address structure, a character string for the returned node name, an integer for the length of the returned node name, a character string to receive the returned service name, an integer for the length of the returned service name, and an integer that specifies the type of address processing to be performed.

### 7.4.2 gethostbyname Function Call

Applications that use the gethostbyname function call must be changed to use the getaddrinfo function call, as folllows:

| AF_INET Call | AF_INET6 Call |
|---|---|
| gethostbyname(*name*) | err=getaddrinfo(*node_name, service_name, &hints, &result*);<br>.<br>.<br>.<br>freeaddrinfo(&*result*); |

Make the following changes to your application, as needed:

1. Change the function name from gethostbyname to getaddrinfo and provide a character string that contains the node name, a character string that contains the service name to use, a pointer to a hints structure that contains processing options, and a pointer to an addrinfo structure or structures for the returned address information.
2. Add a call to the freeaddrinfo routine to free the addrinfo structure or structures when your application is finished using them.

### 7.4.3 inet_ntoa Function Call

Applications that use the `inet_ntoa` function call must be changed to use the `getnameinfo` function call, as follows:

| AF_INET Call | AF_INET6 Call |
|---|---|
| inet_ntoa(*addr*) | err=getnameinfo(*&sockaddr,sockaddr_len, node_name, name_len, service, service_len,* NI_NUMERICHOST); |

Make the following change to your application, as needed:

1. Change the function name from `inet_ntoa` to `getnameinfo` and provide a pointer to the socket address structure, a character string for the returned node name, an integer for the length of the returned node name, a character string to receive the returned service name, an integer for the length of the returned service name, and the NI_NUMERICHOST flag.

### 7.4.4 inet_addr Function Call

Applications that use the `inet_addr` function call must be changed to use the `getaddrinfo` function call, as follows:

| AF_INET Call | AF_INET6 Call |
|---|---|
| result=inet_addr(*&string*) | err=getaddrinfo(*node_name, service_name, &hints, &result*);<br>.<br>.<br>.<br>freeaddrinfo(*&result*); |

Make the following change to your application, as needed:

1. Change the function name from `inet_addr` to `getaddrinfo` and provide a character string that contains the node name, a character string that contains the service name to use, a pointer to a hints structure that contains the AI_NUMERICHOST option, and a pointer to an `addrinfo` structure or structures for the returned address information.
2. Add a call to the `freeaddrinfo` routine to free the `addrinfo` structure or structures when your application is finished using them.

## 7.5 Other Application Changes

In addition to the name changes, you should review your code for specific uses of IP address information and variables.

### 7.5.1 Comparing IP Addresses

If your application compares IP addresses or tests IP addresses for equality, the `in6_addr` structure changes (see in Section 7.3.1) will change the comparison of *int* quantities to a comparison of structures. This will break the code and cause compiler errors.

Make either of the following changes to your application, as needed:

| AF_INET Code | AF_INET6 Code |
|---|---|
| (addr1->s_addr == addr2->s_addr) | (memcmp(addr1, addr2, sizeof(struct in6_ addr)) == 0) |

1. Change the equality expression to one that uses the `memcmp` (memory comparison) function.

| AF_INET Code | AF_INET6 Code |
|---|---|
| (addr1->s_addr == addr2->s_addr) | IN6_ARE_ADDR_EQUAL(addr1, addr2) |

1. Change the equality expression to one that uses the `IN6_ARE_ADDR_EQUAL` macro.

### 7.5.2 Comparing an IP Address to the Wildcard Address

If your application compares an IP address to the wildcard address, the `in6_addr` structure changes (see Section 7.3.1) will change the comparison of int quantities to a comparison of structures. This will break the code and cause compiler errors.

Make either of the following changes to your application, as needed:

| AF_INET Code | AF_INET6 Code |
|---|---|
| (addr->s_addr == INADDR_ANY) | IN6_IS_ADDR_UNSPECIFIED(addr) |

1. Change the equality expression to one that uses the `IN6_IS_ADDR_UNSPECIFIED` macro.

| AF_INET Code | AF_INET6 Code |
|---|---|
| (addr->s_addr == INADDR_ANY) | (memcmp(addr, in6addr_any, sizeof(struct in6_addr)) == 0) |

1. Change the equality expression to one that uses the `memcmp` (memory comparison) function.

### 7.5.3 Using int Data Types to Hold IP Addresses

If your application uses `int` data types to hold IP addresses, the in6_addr structure changes (see Section 7.3.1 ) will change the assignment. This will break the code and cause compiler errors.

Make the following changes to your application, as needed:

| AF_INET Code | AF_INET6 Code |
|---|---|
| struct in_addr foo;<br>int bar;<br>.<br>.<br>.<br>bar = foo.s_addr; | struct in6_addr foo;<br>struct in6_addr bar;<br>.<br>.<br>.<br>bar = foo; |

1. Change the data type for `bar` from `int` to a struct `in6_addr`.
2. Change the assignment statement for `bar` to remove the `s_addr` field reference.

### 7.5.4 Using Functions that Return IP Addresses

If your application uses functions that return IP addresses as `int` data types, the `in6_addr` structure changes (see Section 7.3.1 will change the destination of the return value from an `int` to an array of char. This will break the code and cause compiler errors.

Make the following changes to your application, as needed:

| AF_INET Code | AF_INET6 Code |
|---|---|
| struct in_addr *addr;<br>addr->s_addr = foo(*xxx*); | struct in6_addr **addr*;<br>foo(*xxx*, *addr*); |

1. Restructure the function to enable you to pass the address of the structure in the call. In addition, modify the function to write the return value into the structure pointed to by `addr`.

### 7.5.5 Changing Socket Options

If your application uses IPv4 IP-level socket options, change them to the corresponding IPv6 options.

## 7.6 Sample Client/Server Programs

This section contains sample client and server programs that demonstrate the differences between IPv4 and IPv6 coding conventions:

- Section 7.6.1 contains sample programs using IPv4 AF_INET sockets.
- Section 7.6.2 contains sample programs using IPv6 AF_INET6 sockets.

To build the examples, use the following commands:

```
$ DEFINE DECC$SYSTEM_INCLUDE TCPIP$EXAMPLES:
$ DEFINE ARPA TCPIP$EXAMPLES:
$ DEFINE NET TCPIP$EXAMPLES:
$ DEFINE NETINET TCPIP$EXAMPLES:
$ DEFINE SYS TCPIP$EXAMPLES:

$ CC/NOOPT/STANDARD=VAXC/PREFIX=ALL/EXTERN_MODEL=STRICT_REFDEF/DEFINE=(INET6,_SOCKADDR_LEN) client.c
$ LINK/MAP client,TCPIP$LIBRARY:TCPIP$LIB/lib

$ CC/NOOPT/STANDARD=VAXC/PREFIX=ALL/EXTERN_MODEL=STRICT_REFDEF/DEFINE=(INET6,_SOCKADDR_LEN) server.c
$ LINK/MAP server,TCPIP$LIBRARY:TCPIP$LIB/lib
```

### 7.6.1 Programs Using AF_INET Sockets

This section contains a client and a server program that use AF_INET sockets.

#### 7.6.1.1 Client Program

The following is a sample client program that you can build, compile and run on your system. The program sends a request to and receives a response from the system specified on the command line.

```
/*
 * ********************************************************************
 * *                                                                  *
 * *    Copyright 2000 Compaq Computer Corporation                    *
 * *                                                                  *
 * *   The software contained on this media  is  proprietary  to      *
 * *   and  embodies  the  confidential  technology  of  Compaq       *
 * *   Computer  Corporation.  Possession, use,  duplication  or      *
 * *   dissemination of the software and media is authorized only     *
 * *   pursuant to a valid written license from Compaq Computer       *
 * *   Corporation.                                                   *
 * *                                                                  *
 * *   RESTRICTED RIGHTS LEGEND   Use, duplication, or disclosure     *
 * *   by the U.S. Government is subject to restrictions  as  set     *
 * *   forth in Subparagraph (c)(1)(ii)  of  DFARS  252.227-7013,     *
 * *   or  in  FAR 52.227-19, as applicable.                          *
 * *                                                                  *
 * ********************************************************************
 */
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/errno.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <arpa/inet.h>

#define SERVER_PORT     7639
#define CLIENT_PORT     7739

#define MAXBUFSIZE 4096

int main (
    int argc,
    char **argv )
{
    int             s;
    char            databuf[MAXBUFSIZE];
    int             dcount;
    struct sockaddr_in     serveraddr; 1
    struct sockaddr_in     clientaddr;
    int             serveraddrlen;
    const char      *ap;
    const char      *request = "this is the client's request";
    struct hostent  *hp;
    char            *server;

    if (argc < 2) {
        printf("Usage: client <server>\n");
        exit(1);
    }
    server = argv[1];

    bzero((char *) &serveraddr, sizeof(struct sockaddr_in)); 2
    serveraddr.sin_family = AF_INET;
    if ((hp = gethostbyname(server)) == NULL) {  3
        printf("unknown host: %s\n", server);
        exit(2);
    }
    serveraddr.sin_port = htons(SERVER_PORT);
```

```
    while (hp->h_addr_list[0] != NULL) {
        if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0) { 4
            perror("socket");
            exit(3);
        }
        memcpy(&serveraddr.sin_addr.s_addr, hp->h_addr_list[0],
                        hp->h_length);

        if (connect(s, (struct sockaddr *)&serveraddr, sizeof(serveraddr)) < 0) {
            perror("connect");
            close(s);
            hp->h_addr_list++;
            continue;
        }
        break;
    }
    if (send(s, request, strlen(request), 0) < 0) { 5
        perror("send");
        exit(5);
    }
    dcount = recv(s, databuf, sizeof(databuf), 0);
    if (dcount < 0) {
        perror("recv");
        exit(6);
    }
    databuf[dcount] = '\0';

    hp = gethostbyaddr((char *)&serveraddr.sin_addr.s_addr, 6
        sizeof(serveraddr.sin_addr.s_addr), AF_INET);
    ap = inet_ntoa(serveraddr.sin_addr);  7
    printf("Response received from");
    if (hp != NULL)
        printf(" %s", hp->h_name);
    if (ap != NULL)
        printf(" (%s)", ap);
    printf(": %s\n", databuf);

    close(s);
}
```

**1** Declares sockaddr_in structures.

**2** Clears the server address and sets up server variables.

**3** Calls gethostbyname to obtain the server address.

**4** Creates AF_INET socket.

**5** Sends a request to the server.

**6** Calls gethostbyaddr to retrieve the server name.

**7** Calls inet_ntoa to convert the server address to a text string.

#### 7.6.1.2 Server Program

The following is a sample server program that you can build, compile, and run on your system. The program receives requests from and sends responses to client programs on other systems.

```
/*
 * ****************************************************************
 * *                                                              *
 * *    Copyright 2000 Compaq Computer Corporation                *
 * *                                                              *
 * *   The software contained on this media  is  proprietary  to  *
 * *   and  embodies  the  confidential  technology  of  Compaq   *
 * *   Computer  Corporation.  Possession, use,  duplication  or  *
 * *   dissemination of the software and media is authorized only *
 * *   pursuant to a valid written license from Compaq Computer   *
 * *   Corporation.                                               *
 * *                                                              *
 * *   RESTRICTED RIGHTS LEGEND   Use, duplication, or disclosure *
 * *   by the U.S. Government is subject to restrictions  as  set *
 * *   forth in Subparagraph (c)(1)(ii)  of  DFARS  252.227-7013, *
 * *   or  in  FAR 52.227-19, as applicable.                      *
 * *                                                              *
 * ****************************************************************
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/errno.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <arpa/inet.h>

#define SERVER_PORT     7639
#define CLIENT_PORT     7739

#define MAXBUFSIZE 4096

int main (
    int argc,
    char **argv )
{
    int             s;
    char            databuf[MAXBUFSIZE];
    int             dcount;
    struct sockaddr_in    serveraddr; 1
    struct sockaddr_in    clientaddr;
    int                   clientaddrlen;
    struct hostent        *hp;
    const char            *ap;
    const char            *response = "this is the server's response";
    u_short               port;

    if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0) { 2
        perror("socket");
        exit(1);
    }

    bzero((char *) &serveraddr, sizeof(struct sockaddr_in)); 3
    serveraddr.sin_family      = AF_INET;
    serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
    serveraddr.sin_port        = htons(SERVER_PORT);
```

```
    if (bind(s, (struct sockaddr *)&serveraddr, sizeof(serveraddr)) < 0) {
        perror("bind");
        exit(2);
    }
    if (listen(s, SOMAXCONN) < 0) {
        perror("Listen");
        close(s);
        exit(3);

    while (1) {
        int new_s;
        clientaddrlen = sizeof(clientaddr);
        new_s = accept(s, (Struct sockaddr*)&clientaddr, &clientaddrlen);

        dcount = recv(new_s, databuf, sizeof(databuf), 0);
        if (dcount <= 0) {
            perror("recv");
            close(new_s);
            continue;
        }
        databuf[dcount] = '\0';
        hp = gethostbyaddr((char *)&clientaddr.sin_addr.s_addr, 4
            sizeof(clientaddr.sin_addr.s_addr), AF_INET);
        ap = inet_ntoa(clientaddr.sin_addr); 5
        port = ntohs(clientaddr.sin_port);
        printf("Request received from");
        if (hp != NULL)
            printf(" %s", hp->h_name);
        if (ap != NULL)
            printf(" (%s)", ap);
        printf(" port %d \"%s\"\n", port, databuf);

        if (send(new_s, response, strlen(response), 0) < 0) {
            perror("send");
            continue;
        }
        close(new_s);
    }
    close(s);
}
```

**1** Declares sockaddr_in structures.

**2** Creates an AF_INET socket.

**3** Clears the server address and sets up server variables.

**4** Calls gethostbyaddr to retrieve client name.

**5** Calls inet_ntoa to convert the client address to a text string.

## 7.6.2 Programs Using AF_INET6 Sockets

This section contains a client and a server program that use AF_INET6 sockets.

### 7.6.2.1 Client Program

The following is a sample client program that you can build, compile and rin on your system. The program sends a request to and receives a response from the system specified on the command line.

```
/*
 * ****************************************************************
 * *                                                              *
 * *    Copyright 2000 Compaq Computer Corporation                *
 * *                                                              *
 * *   The software contained on this media  is  proprietary  to  *
 * *   and  embodies  the  confidential  technology  of  Compaq   *
 * *   Computer Corporation.  Possession, use,  duplication  or   *
 * *   dissemination of the software and media is authorized only *
 * *   pursuant to a valid written license from Compaq Computer   *
 * *   Corporation.                                               *
 * *                                                              *
 * *   RESTRICTED RIGHTS LEGEND   Use, duplication, or disclosure *
 * *   by the U.S. Government is subject to restrictions  as  set *
 * *   forth in Subparagraph (c)(1)(ii)  of  DFARS  252.227-7013, *
 * *   or  in  FAR 52.227-19, as applicable.                      *
 * *                                                              *
 * ****************************************************************
 */
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/errno.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <arpa/inet.h>

#define SERVER_PORT     7639
#define CLIENT_PORT     7739

#define MAXBUFSIZE 4096

int main (
    int argc,
    char **argv )
{
    int             s;
    char            databuf[MAXBUFSIZE];
    int             dcount;
    struct addrinfo *server_info;  1
    struct addrinfo *cur_info;
    struct addrinfo hints;
    struct sockaddr_in6    serveraddr;
    char            addrbuf[INET6_ADDRSTRLEN];
    char            node[MAXDNAME];
    char            service[MAXDNAME];
    int             ni;
    int             err;
    int             serveraddrlen;
    const char      *request = "this is the client's request";
    char            *server;

    if (argc < 2) {
        printf("Usage: client client <server>\n");
        exit(1);
    }
    server = argv[1];
    bzero((char *) &hints, sizeof(hints)); 2
    hints.ai_family = AF_INET6;
    hints.ai_protocol = IPPROTO_TCP;
    hints.ai_flags = AI_DEFAULT;

    sprintf(service, "%d", SERVER_PORT);
```

```
    err = getaddrinfo(server, service, &hints, &server_info);  3
    if (err != 0) {
        if (err == EAI_SYSTEM)
            perror("getaddrinfo");
        else
            printf("%s", gai_strerror(err0));
        exit(2);
    }
    cur_info = server_info;

    while (cur_info != NULL) {
        if ((s = socket(cur_info->ai_family, cur_info->ai_socktype, 0)) < 0) { 4
            perror("socket");
            exit(3);
        }
        if (connect(s, cur_info->ai_addr, cur_info->ai_addrlen) <0 {
            close(s);
            cur_info = cur_info->ai_next;
            continue;
        }
        break;
    }
    freeaddrinfo(server_info); 5

    if (send(s, request, strlen(request), 0) < 0) { 6
        perror("send");
        exit(5);
    }
    dcount = recv(s, databuf, sizeof(databuf), 0);
    if (dcount < 0) {
        perror("recv");
        exit(6);
    }
    databuf[dcount] = '\0';
    serveraddrlen = sizeof(serveraddr);
    if (getpeername(s, (struct sockaddr*) &serveraddr, &serveraddrlen) < 0 {
        perror("getpeername");
        exit(7);
    }
    printf("Response received from");
    ni = getnameinfo((struct sockaddr*)&serveraddr, serveraddrlen, 7
                        node, sizeof(node), NULL, 0, NI_NAMEREQD);
    if (ni == 0)
        printf(" %s", node);
    ni = getnameinfo((struct sockaddr*)&serveraddr, serveraddrlen, 8
                        addrbuf, sizeof(addrbuf), NULL, 0, NI_NUMERICHOST);
    if (ni == 0)
        printf(" (%s)", addrbuf);

    printf(": %s\n", databuf);

    close(s);
}
```

**1** Declares addrinfo structures, hints structure, sockaddr_in6 structure, address string buffer, node name string buffer, service name string buffer, error number variable, and server address length variable.

**2** Clears the hints structure and sets up hints variables.

**3** Calls getaddrinfo to obtain the server address.

**4** Creates an AF_INET6 socket.

**5** Frees all addrinfo structures.

6 Sends a request to the server.

7 Calls getnameinfo to obtain the server name.

8 Calls getnameinfo to obtain the server's numeric address and message data.

#### 7.6.2.2 Server Program

The following is a sample server program that you can build, compile, and run on your system. The program receives requests from and sends responses to client programs on other systems.

```
/*
 * ********************************************************************
 * *                                                                  *
 * *    Copyright 2000 Compaq Computer Corporation                    *
 * *                                                                  *
 * *   The software contained on this media  is  proprietary  to      *
 * *   and  embodies  the  confidential  technology  of  Compaq       *
 * *   Computer Corporation.  Possession, use,  duplication  or       *
 * *   dissemination of the software and media is authorized only     *
 * *   pursuant to a valid written license from Compaq Computer       *
 * *   Corporation.                                                   *
 * *                                                                  *
 * *   RESTRICTED RIGHTS LEGEND   Use, duplication, or disclosure     *
 * *   by the U.S. Government is subject to restrictions  as  set     *
 * *   forth in Subparagraph (c)(1)(ii)  of  DFARS  252.227-7013,     *
 * *   or  in  FAR 52.227-19, as applicable.                          *
 * *                                                                  *
 * ********************************************************************
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/errno.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <arpa/inet.h>

#define SERVER_PORT     7639
#define CLIENT_PORT     7739

#define MAXBUFSIZE 4096

int main (
    int argc,
    char **argv )
{
    int             s;
    char            databuf[MAXBUFSIZE];
    int             dcount;
    struct sockaddr_in6   serveraddr; 1
    struct storage        clientaddr;
    char                  addrbuf[INET6_ADDRSTRLEN];
    char                  node[MAXDNAME];
    char                  port[MAXDNAME];
    int                   err;
    int                   ni;
    int                   clientaddrlen;
    const char            *response = "this is the server's response";
```

```
    if ((s = socket(AF_INET6, SOCK_STREAM, 0)) < 0) { 2
        perror("socket");
        exit(1);
    }

    bzero((char *) &serveraddr, sizeof(struct sockaddr_in6)); 3
    serveraddr.sin6_family      = AF_INET6;
    serveraddr.sin6_addr        = in6addr_any;
    serveraddr.sin6_port        = htons(SERVER_PORT);

    if (bind(s, (struct sockaddr *)&serveraddr, sizeof(serveraddr)) < 0) {
        perror("bind");
        exit(2);
    }
    if (listen(s, SOMAXCONN) < 0) {
        perror("listen");
        close(s);
        exit(3);
    }
    while (1) {
        int new_s;
        clientaddrlen = sizeof(clientaddr);
        bzero((char *)&clientaddr, clientaddrlen); 4
        new_s = accept(s, (struct sockaddr*)&clientaddr, &clientaddrlen);
        if (new_s < 0) {
            perror("accept");
            continue;
        }
        dcount = recv(s, databuf, sizeof(databuf), 0);
        if (dcount <= 0) {
            perror("recv");
            close(new_s);
            continue;
        }
        databuf[dcount] = '\0';

        printf("Request received from");
        ni = getnameinfo((struct sockaddr *)&clientaddr, 5
            clientaddrlen, node, sizeof(node), NULL, 0, NI_NAMEREQD);
        if (ni == 0)
            printf(" %s", node);
        ni = getnameinfo((struct sockaddr *)&clientaddr, 6
            clientaddrlen, addrbuf, sizeof(addrbuf), port, sizeof(port),
            NI_NUMERICHOST|NI_NUMERICSERV);
        if (ni == 0)
            printf(" (%s) port %d, addrbuf, port);
        printf(" \"%s\"\n", port, databuf);

        if (send(new_s, response, strlen(response), 0) < 0) {
            perror("send");
            close(new_s);
            continue;
        }
        close(new_s);
    }
    close(s);
}
```

**1** Declares sockaddr_in6 structures, address string buffer, and error number variable.

**2** Creates an AF_INET6 socket.

**3** Clears the server address and sets up the server variables.

**4** Clears the client address.

5 Calls getnameinfo to retrieve the client name.

6 Calls getnameinfo to obtain the client's address, port, and message data.

### 7.6.3 Sample Program Output

This section contains sample output from the preceding server and client programs. The server program makes and receives all requests on an AF_INET6 socket using sockaddr_in6. For requests received over IPv4, sockaddr_in6 contains an IPv4-mapped IPv6 address.

The following example shows a client program running on node hostb6 and sending a request to node hosta6. The program uses an AF_INET6 socket. The node hosta6 has the IPv6 address 3ffe:1200::a00:2bff:fe97:7be0 in the Domain Name System (DNS).

```
$ client :== $client.exe
$ client hosta6
Response received from hosta6.ipv6.corp.example (3ffe:1200::a00:2bff:fe97:7be0):
 this is the server's response
```

On the server node, the following example shows the server program invocation and the request received from the client node hostb6:

```
$ run server
Request received from hostb6.ipv6.corp.example (3ffe:1200::a00:2bff:fe2d:02b2
 port 7739 "this is the client's request"
```

The following example shows the client program running on node hostb and sending a request to node hosta. The program uses an AF_INET6 socket. The hosta node has the IPv4 address 10.10.10.13 in the DNS.

```
$ client :== $client.exe
$ client hosta
Response received from hosta.corp.example (::ffff:10.10.10.13): this is the
 server's response
```

On the server node, the following example shows the server program invocation and the request received from the client node hostb:

```
$ run server
Request received from hostb.corp.example (::ffff:10.10.10.251) port 7739
 "this is the client's request"
```

The following example shows the client program running on node hostc and sending a request to node hosta. The program was built and run on an IPv4-only system using an AF_INET socket.

```
$ client :== $client.exe
$ client hosta
Response received from hosta.corp.example (10.10.10.13): this is the
 server's response
```

On the server node, the following example shows the server program invocation and the request received from the client node hostc:

```
$ run server
Request received from hostc.corp.example (::ffff:10.10.10.63) port 7739
 "this is the client's request"
```

# A

# Supported IPv6 RFCs

The following are supported IPV6 Request for Comments (RFCs):

- Internet Protocol Version 6 (IPv6) Specification, RFC 2460 (December 1998)
- Internet Control Message Protocol (ICMPv6) for Internet Protocol Version 6 (IPv6), RFC 2463 (December 1998)
- Neighbor Discovery for IP Version 6 (IPv6), RFC 2461 (December 1998)
- IPv6 Stateless Address Autoconfiguration, RFC 2462 (December 1998)
- Path MTU Discovery for IP Version 6, RFC 1981 (August 1996)
- Transition Mechanisms for IPv6 Hosts and Routers, RFC 1933 (April 1996)
- IP Version 6 Addressing Architecture, RFC 2373 (July 1998)
- An IPv6 Aggregatable Global Unicast Address Format, RFC 2374 (July 1998)
- IPv6 Testing Address Allocation, RFC 2471 (December 1998)
- Transmission of IPv6 Packets over Ethernet Networks, RFC 2464 (December 1998)
- Transmission of IPv6 Packets over FDDI Networks, RFC 2467 (December 1998)
- Basic Socket Interface Extensions for IPv6, RFC 2553 (April 1999)
- Advanced Sockets API for IPv6, RFC 2292 (February 1998)
- DNS Extensions to Support IP version 6, RFC 1886 (December 1995)
- Dynamic Updates in the Domain Name System (DNS UPDATE), RFC 2136 (April 1997)
- RIPng, RFC 2080 (January 1997)

# B

# IPv6 Extensions to Management Commands and IPv6 Processes

## B.1 IPv6 Extensions to Management Commands

The *Compaq TCP/IP Services for OpenVMS Management Command Reference* decribes the basic management commands, including the UNIX commands, you can use to manage the TCP/IP Services software. The *Compaq TCP/IP Services for OpenVMS Tuning and Troubleshooting* contains more detailed information on the UNIX management commands. The following sections describe only IPv6 extensions to those UNIX management commands.

To use UNIX management commands at the DCL prompt, execute the following command procedure (or put it into your LOGIN.COM so that it executes each time you log in):

```
$ @SYS$MANAGER:TCPIP$DEFINE_COMMANDS
```

**Note**

UNIX flags and OpenVMS interface names are case sensitive. When entering UNIX management commands at the DCL prompt, you must enclose uppercase UNIX flags and OpenVMS interface names in quotes to preserve the case of the input.

### B.1.1 ifconfig Command

For the AF_INET6 address family, use the following syntax:

```
ifconfig interface_id address_family [[ip6prefix]
  address[/bitmask] [dest_address]] [parameters]
```

For the AF_INET6 address family, the address argument is either a host name or the 128-bit IPv6 address, as follows:

```
x:x:x:x:x:x:x:x
```

In this format, each *x* is the hexadecimal value of a 16-bit piece of the address.

The **ip6prefix** argument specifies that the interface identifier is to be appended to the **address** argument when configuring an address on the interface. The interface identifier uniquely identifies an interface on a subnet and is typically the interface's link-layer address. The following are the parameters for the `ifconfig` command.

**Parameters** [AF_INET6 only]:

- **ip6interfaceid id**

  Overrides the default interface ID, which depends on the underlying link type (for example, Ethernet, FDDI), and specifies an `inet6` interface ID for the interface. For example, if your system has the Ethernet hardware address 08-00-2b-2a-1e-d3, the following command generates the `inet6` link-local address `fe80::a00:2bff:fe2a:1ed3` for the interface:

  ```
  $ ifconfig "WE0" ipv6
  ```

  On the same system, the following command generates the `inet6` interface ID `abcd:1234` for the interface:

  ```
  $ ifconfig "WE0" ip6interfaceid ::abcd:1234 ipv6
  ```

- **ipv6**

  Initializes IPv6-related data structures and assigns an IPv6 link-local address to the interface.

- **-ipv6**

  Removes any IPv6 configuration associated with the interface, including all IPv6 addresses and IPv6 routes through the interface. This command is equivalent to the `ifconfig interface inet6 delete` command.

- **ip6dadtries value**

  Specifies the number of consecutive neighbor solicitation messages that your system transmits as it performs duplicate address detection on a tentative address.

- **ip6hoplimit hops**

  Sets the default number of hops to be included in transmitted unicast IP packets.

- **ip6mtu mtu_value**

  Alters the maximum trasmission unit (MTU) for messages that your system transmits on the link.

- **ip6nonud**

  Disables Neighbor Unreachability Detection (NUD) on the interface.

- **ip6reachabletime time**

  Sets the time, in milliseconds, that your system considers a neighbor is reachable after your system receives a reachability confirmation message.

- **ip6retranstimer value**

  Sets the time interval, in milliseconds, between neighbor solicitation messages to a neighbor.

### B.1.2 iptunnel Command

The `iptunnel` command creates configured tunnels for sending and receiving IPv6 or IPv4 packets that are encapsulated as the payload of an IPv4 datagram.

The iptunnel command can perform the following three operations:

- **create**

  Creates a tunnel interface, which you must subsequently configure by using the ifconfig command. The syntax of the create operation is as follows:

  ```
  iptunnel create [-I int-name] [v4-dest] [v4-src]
  ```

  **Parameters**

  - **-I int-name**

    Specifies the interface unit of the tunnel to be created. This is an optional parameter. The **int-name** parameter has the form ipt*x*, where *x* is the interface unit number. By default, the interface name selected for the tunnel is ipt*x*+1, or the value of the interface unit number of the last tunnel created plus 1.

  - **v4-dest**

    Specifies the remote endpoint to which a tunnel is to be created.

  - **v4-src**

    Sets the IPv4 source address in the encapsulating header. The tunnel is enabled (packets are sent and received on the tunnel) only if *v4-src* is a valid address on the system. This is an optional parameter.

- **delete**

  Deletes a tunnel interface. You must disable the tunnel before you can delete it by executing the following command:

  ```
  $ ifconfig tunnel name down  delete abort
  ```

- **show**

  Shows the tunnel attributes (name, tunnel endpoints, next hop for tunneled packets).

For related information, see RFC 2003, IP Encapsulation within IP, Perkins, C., October 1996.

### B.1.3 netstat Command

The netstat command displays network-related data in various formats.

The parameters **-f address_family** limit reports to the specified address family. The address families that can be specified include the following:

- inet—Specifies reports of the AF_INET family, if present in the kernel.
- inet6—Specifies reports of the AF_INET6 family, if present in the kernel.

To display IPv6 routing entries, enter this command:

```
$ netstat -rnf inet6
```

To display active IPv6 connections, enter this command:

```
$ netstat -af inet6
```

### B.1.4 traceroute Command

The `traceroute` command with the **host** argument prints the route that packets take to both IPv4 and IPv6 hosts.

The **-G @addr1@addr2...** parameters (IPv6 only) specify the source route for packets to travel. The route consists of one or more IPv6 node names or addresses. Use the ampersand character (&) to separate multiple addresses. You can specify up to 10 addresses.

The **-V version** parameter specifies the Internet Protocol (IP) version number to enable the resolver to return the correct address. Use the **-V 4** option if you want to issue a `traceroute` command to a host name (not an IP address) that has both IPv4 and IPv6 addresses, and you want to trace the route to the IPv4 address.

**Note**

By default, `traceroute` tries to resolve destination host names as an IPv6 address. If that fails, it resolves the host name as an IPv4 address. You can override this behavior with the **-V** option.

## B.2 IPv6 Processes

### B.2.1 TCPIP$ND6HOST

The TCPIP$ND6HOST process receives and processes IPv6 Router Advertisement (RA) packets of the Neighbor Discovery Protocol. This enables a system to autoconfigure itself without manual intervention.

The TCPIP$ND6HOST process performs the following functions, based on the contents of IPv6 Router Advertisements it receives:

- Router discovery—Learns the IPv6 address of default routers and installs default routes in the kernel routing table.
- On-link prefix discovery—Learns IPv6 on-link prefixes (ranges of IPv6 addresses that are directly reachable on a given link).
- Stateless address configuration—Automatically creates and deletes interface addresses.
- Interface attribute configuration—Automatically configures datalink attributes, such as hop limit, reachable time, retransmit time, and link MTU.

**Caution**

Do not run the TCPIP$ND6HOST and TCPIP$IP6RTRD processes on the same host, since this may produce unpredictable results.

## B.2.2 TCPIP$IP6RTRD Process

The TCPIP$IP6RTRD process sends IPv6 Router Advertisement (RA) packets of the Neighbor Discovery Protocol. These packets enable any listening host to autoconfigure itself without manual intervention. In addition, you can configure TCPIP$IP6RTRD to send and process RIPng messages.

At startup, the TCPIP$IP6RTRD process reads its configuration file for startup information.

**Caution**

Do not run the TCPIP$ND6HOST and TCPIP$IP6RTRD processes on the same host, since this may produce unpredictable results.

The TCPIP$IP6RTRD.CONF file contains configuration information that is read by the TCPIP$IP6RTRD process at initialization time. This file contains statements that control information sent in Router Advertisements and RIPng messages.

The TCPIP$IP6RTRD.CONF file consists of structured information for each interface in the following format:

```
interface interface-name {
   # interface keyword-value pairs, one per line
   Prefix prefix/length {
       # prefix keyword-value pairs, one per line
   }
}
```

Comments begin with the pound sign (#) and continue to the end of the line.

### B.2.2.1 Interface Keyword Information

Table B–1 lists the interface keywords and range of accepted values described in RFC 2461.

**Table B–1 RFC 2461 Interface Keywords and Values**

| Keyword | Values | Default |
|---|---|---|
| AdvSendAdvertisements | YES/NO | YES |
| MaxRtrAdvInterval | 4–1800 seconds | 600 |
| MinRtrAdvInterval | 3–(0.75 * MaxRtrAdvInterval) | 200 |
| AdvManagedFlag | 0/1 | 0 |
| AdvOtherConfigFlag | 0/1 | 0 |
| AdvLinkMTU | Nonnegative integer | 0 |
| AdvReachableTime | 0–3,600,000 milliseconds | 0 |
| AdvRetransTimer | Nonnegative integer | 0 |
| AdvDefaultLifetime | 0, or MaxRtrAdvInterval - 9000 seconds | 1800 |

In addition, the following interface keywords are accepted:

- AdvCurHopLimit

  The value to be placed in the Cur Hop Limit field in the Router Advertisement messages sent by the router. The value 0 means unspecified (by this router). Valid values are any nonnegative integer. The default is 0.

- AdvSendLinkLayerAddress

  Sends the interface link-layer address option in outgoing router advertisements. Valid values are YES and NO. The default is YES.

- ripng

  Enables (YES) or disables (NO) participation in RIPng on the interface. If enabled, RIPng updates are sent on the interface, and received RIPng updates are processed as defined in RFC 2080. You cannot specify YES for automatic tunnels (the tun0 interface). The default is YES (except for tun0).

- SplitHorizon

  Enables (1) or disables (0) the Split Horizon algorithm as specified in RFC 2080. The default is 1.

- PoisonReverse

  Enables (1) or disables (0) the Poisoned Reverse algorithm as specified in RFC 2080. The default is 1.

#### B.2.2.2 Address-Prefix Keyword Information

Each address prefix to be configured on the interface must be defined within a prefix block that begins with the keyword Prefix followed by the prefix and length (separated by a slash [/] and optionally followed by an additional address-prefix information block of keyword-value pairs).

Table B–2 lists address prefix keywords and values that are described in RFC 2461.

**Table B–2 RFC 2461 Prefix Keywords**

| Prefix Keyword | Values | Default |
|---|---|---|
| AdvValidLifetime | Integer | 2592000 seconds |
| AdvPreferredLifetime | Integer | 604800 seconds |
| AdvOnLinkFlag | 0/1 | 1 |
| AdvAutonomousFlag | 0/1 | 1 |

Table B–3 lists address prefix keywords and values that are described in RFC 2080.

**Table B–3 RFC 2080 Prefix Keywords**

| Prefix Keyword | Values | Default |
|---|---|---|
| RouteMetric | 1–16 (inclusive) | 1 |
| RouteTag | Integer | 0 |

In addition, you can specify the following address-prefix keywords:

- ConfigureThisPrefix

  The TCPIP$IP6RTRD process will configure the advertised prefix on the interface if ConfigureThisPrefix is specified and set to 1, or if ConfigureThisPrefix is not specified and AdvAutonomousFlag is set to 1.

  The prefix is not autoconfigured in all other cases. Valid values are 0 and 1. The default value is the value of AdvAutonomousFlag.

For related information, see the following RFCs:

- RFC 2461, Neighbor Discovery for IP version 6 (IPv6), Narten, T.; Nordmark, E., Simpson W. A., December 1998
- RFC 2462, IPv6 Stateless Address Autoconfiguration, Thompson, S.; Narten, T., December 1998
- RFC 2080, RIPng for IPv6, Malkin, G., Minnear, R., January 1997

# C
# Deprecated Library Functions

This appendix describes deprecated library functions that were provided in previous Early Adopter Kits (EAKs). Do not use these functions if you are developing new applications. If your existing applications use these functions, see Chapter 7 for changes you should make to your code.

The following table shows the deprecated functions and their replacements:

| Deprecated Function | Replacement Function |
|---|---|
| getipnodebyname | getaddrinfo |
| getipnodebyaddr | getnameinfo |
| freehostent | freeaddrinfo |

## C.1 getipnodebyname Function

The getipnodebyname function has the following syntax:

```
#include <netdb.h>
struct hostent *getipnodebyname(
        const char *name,
        int addr_family,
        int flags,
        int *error_num );
```

**Parameters**:

- **name**

  Specifies the official network node name, alias, or numeric node address (for example, an IPv4 dotted-decimal address or an IPv6 hexadecimal address).

- **addr_family**

  Specifies the address family. This can be AF_INET for IPv4 addresses or AF_INET6 for IPv6 addresses.

- **flags**

  Specifies the type of addresses for which to search and the types of addresses that are returned. Table C–1 describes how the processing is affected by the values of the **af** parameter and commonly used flag values.

- **error_num**

  Specifies an error return code value if the function is not successful.

**Description**

The getipnodebyname( ) routine is an evolution of the gethostbyname( ) routine that enables name lookups in address families other than AF_INET.

## C.1 getipnodebyname Function

The getipnodebyname( ) routine returns a pointer to a structure of type hostent. Its members specify data obtained from the local TCPIP$ETC:IPNODES.DAT file, TCPIP$HOSTS.DAT file or from one of the files distributed by DNS/BIND.

If multiple addresses are found, the h_addr_list field in the hostent structure contains the addresses.

The <netdb.h> header file defines the hostent structure.

If you are using DNS/BIND, the information is obtained from a name server as configured. When the name server is not running, the getipnodebyname( ) routine searches both the local TCPIP$ETC:IPNODES.DAT name file for IPv6 and IPv4 addresses and the hosts name file for IPv4 addresses, if the addresses not are found in the TCPIP$ETC:IPNODES.DAT file.

Table C–1 lists the flags parameters and how the processing is affected by the value of the *af* parameters.

**Table C–1 Node Name to Address Processing**

| Flag Value | af Value is AF_NET | af Value is AF_INET6 |
|---|---|---|
| 0 | Searches for A records.<br>If found, returns IPv4 addresses (h_length=4).<br>If not, returns a NULL pointer.<br>Provides backward compatibility for existing IPv4 applications. | Searches for AAAA records.<br>If found, returns IPv6 records (h_length=16).<br>If not, returns a NULL pointer. |
| AI_V4MAPPED | Ignored. | Searches for AAAA records.<br>If found, returns IPv6 records (h_length=16).<br>If not, searches for A records.<br>If A records are found, returns IPv4-mapped IPv6 addresses (h_length=16).<br>If no A records are found, returns a NULL pointer. |
| AI_ALL \| AI_ V4MAPPED | Ignored. | Searches for AAAA records.<br>If found, returns IPv6 addresses (h_length=16). Then searches for A records.<br>If A records are found, returns IPv4-mapped IPv6 addresses (h_length=16).<br>If no A records are found, returns a NULL pointer. |

All flags can be used in any combination to achieve finer control of the translation process. The AI_ADDRCONFIG flag is typically used in combination with other flags to modify the search based on the source address or addresses configured on the system. Table C–2 describes how the AI_ADDRCONFIG flag works by itself.

**Table C–2 AI_ADDRCONFIG Flag**

| Flag Value | af Value is AF_NET | af Value is AF_INET6 |
|---|---|---|
| AI_ADDRCONFIG | Searches for A records only if an IPv4 source address is configured on the system. | Searches for AAAA records only if an IPv6 source address is configured on the system.<br>Searches for A records only if an IPv4 source address is configured on the system. |

Most applications will use a combination of the AI_ADDRCONFIG and AI_V4MAPPED flags to control their search. To simplify this for the programmer, the AI_DEFAULT symbol, which is a logical OR of AI_ADDRCONFIG and AI_V4MAPPED, is defined. Table C–3 describes how AI_DEFAULT directs the search.

**Table C–3 AI_DEFAULT Flag**

| Flag Value | af Value is AF_NET | af Value is AF_INET6 |
|---|---|---|
| AI_DEFAULT | Searches for A records only if an IPv4 source address is configured on the system.<br>If found, returns IPv4 addresses (h_length=4).<br>If not, returns a NULL pointer. | Searches for AAAA records only if an IPv6 source address is configured on the system.<br>If found, returns IPv6 records (h_length=16).<br>If not found and if an IPv4 address is configured on the system, searches for A records.<br>If A records are found, returns IPv4-mapped IPv6 addresses (h_length=16).<br>If no A records are found, returns a NULL pointer. |

The `hostent` structure returned by the `getipnodebyname` function is dynamically allocated. You should free this structure and dynamic storage by using the `freehostent` function (see Section C.3).

**Errors**

If the `getipnodebyname( )` routine call fails, **error_num** is set to one of the following values:

- HOST_NOT_FOUND

  The name you have used is not an official node name or alias; another type of name server request may be successful.

- NO_ADDRESS

  The server recognized the request and the name, but no address is available for the name. Another type of name server request may be successful.

- NO_RECOVERY

  An unexpected server failure occurred. This is a nonrecoverable error.

- TRY_AGAIN

  A transient error occurred, for example, the server did not respond. A retry at some later time may be successful.

## C.2 getipnodebyaddr Function

The getipnodebyaddr function has the following syntax:

```
#include <netdb.h>

struct hostent *getipnodebyaddr(
          const void *src,
          size_t len,
          int af,
          int *error_num);
```

**Parameters**

- **src**

  Specifies an Internet address in network order.

- **len**

  Specifies the number of bytes in an Internet address.

- **af**

  Specifies the Internet domain address format. Valid values are AF_INET and AF_INET6.

- **error_num**

  Specifies an error return code value if the function is not successful.

**Description**

The getipnodebyaddr( ) routine is an evolution of the gethostbyaddr( ) routine that enables address lookups in address families other than AF_INET.

The getipnodebyaddr( ) routine returns a pointer to a structure of type hostent. Its members specify data obtained from the local TCPIP$ETC:IPNODES.DAT file, the TCPIP$HOSTS.DAT file, or one of the files distributed by DNS/BIND.

The getipnodebyaddr( ) routine searches the network host database sequentially until a match with the **src** and **af** parameters occurs. The **len** parameter must specify the number of bytes in an Internet address. The **src** parameter must specify the address in network order. The **af** parameter can be either the constant AF_INET or AF_INET6, which specifies the IPv4 address format or the IPv6 address format, respectively. When EOF (end-of-file) is reached without a match, an error value is returned.

If the **src** parameter is either an IPv4-mapped IPv6 address or an IPv4-compatible IPv6 address, the routine performs the following steps:

1. If the **af** parameter is AF_INET6, the **len** parameter is 16, and the **src** parameter is either an IPV4-mapped IPv6 address or an IPv4-compatible IPv6 address, the routine skips the first 12 bytes of the address, sets **af** to AF_INET and **len** to 4.
2. If the **af** parameter is AF_INET, the routine queries for a PTR record in the in-addr.arpa domain.
3. If the **af** parameter is AF_INET6, the routine queries for a PTR record in the ip6.int domain.

4. If the routine returns success, the single address and address family returned in the hostent structure are copies of the **src** parameter and the **af** parameter, respectively, that were passed to the routine.

**Note**

The double colon (::) and ::1 IPv6 addresses are not considered IPv4-compatible addresses.

If you are using DNS/BIND, the address is obtained from a name server as configured. When the name server is not running, the getipnodebyaddr( ) routine searches the local TCPIP$ETC:IPNODES.DAT name file for IPv6 and IPv4 addresses and the hosts name file for IPv4 addresses, if the addresses are not found in the TCPIP$ETC:IPNODES.DAT file.

The getipnodebyaddr( ) routine dynamically allocates the hostent structure. Use the freehostent( ) routine to free the allocated memory. (See Section C.3.

**Errors**

If the getipnodebyaddr( ) routine call fails, **error_num** is set to one of the following the values:

- HOST_NOT_FOUND

  The name you have used is not an official node name or alias; another type of name server request may be successful.

- NO_ADDRESS

  The server recognized the request and the name, but no address is available for the name. Another type of name server request may be successful.

- NO_RECOVERY

  An unexpected server failure occurred. This is a nonrecoverable error.

- TRY_AGAIN

  A transient error occurred, for example, the server did not respond. A retry at some later time may be successful.

## C.3 freehostent Function

The freehostrent function returns hostent structures and dynamic storage to the system. You should use this function to free hostent structures and storage that were returned by getipnodebyname and getipnodebyaddr.

This function has the following syntax:

```
void freehostent(
        struct hostent *ptr );
```

The **ptr** parameter is a pointer to the hostent structure to be freed.

**Note**

Do not use the freehostent function with hostent structures returned by gethostbyname and gethostbyaddr.