z/OS Communications Server
Version 2 Release 4

IPv6 Network and Application Design Guide

IBM

SC27-3663-40

Note:

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 149.

This edition applies to Version 2 Release 4 of z/OS® (5650-ZOS), and to subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2019-07-01

Contents

Figures

Tables

About this document

This document contains information relating to the IPv6 protocol and the implementation of the protocol on z/OS Communications Server Version 2 Release 3.

Who should read this document

This information is intended for programmers and system administrators who are familiar with the IPv6 protocol, TCP/IP, MVS™, and z/OS UNIX.

How this document is organized

This document contains the following information:

- Chapter 1, "Internet Protocol Version 6," on page 1 provides an introduction to IPv6 for z/OS Communications Server Version 2 Release 1.
- Chapter 2, "IPv6 addressing," on page 7 contains a discussion of the IPv6 addressing model and the different IPv6 address types.
- Chapter 3, "IPv6 protocol," on page 17 provides a description of the z/OS Communications Server Version 2 Release 13 implementation of the IPv6 protocol.
- Chapter 4, "Configuring support for z/OS," on page 49 describes the IPv6 function provided in z/OS Communications Server Version 2 Release 13 and how to enable it.
- Chapter 5, "Configuration guidelines," on page 61 contains recommendations and guidance information for implementing the IPv6 functions provided in z/OS Communications Server Version 2 Release 13.
- Chapter 6, "API support," on page 67 describes the various z/OS socket APIs and the level of IPv6 present for each API.
- Chapter 7, " Basic socket API extensions for IPv6," on page 71 describes basic socket API changes that most applications use.
- Chapter 8, "Enabling an application for IPv6," on page 85 describes common issues and considerations involved in enabling existing IPv4 socket applications for IPv6 communications.
- Chapter 9, "Advanced socket APIs," on page 95
- Chapter 10, "Advanced concepts and topics," on page 111
- Appendix A, "IPv6 support tables," on page 119
- Appendix B, "Related protocol specifications," on page 127 lists the related protocol specifications for TCP/IP.
- Appendix C, "Accessibility," on page 147 describes accessibility features to help users with physical disabilities.
- "Notices" on page 149 contains notices and trademarks used in this document.
- "Bibliography" on page 153 contains descriptions of the documents in the z/OS Communications Server library.

How to use this document

To use this document, you should be familiar with z/OS TCP/IP Services and the TCP/IP suite of protocols.

How to contact IBM service

For immediate assistance, visit this website: http://www.software.ibm.com/support

Most problems can be resolved at this website, where you can submit questions and problem reports electronically, and access a variety of diagnosis information.

For telephone assistance in problem diagnosis and resolution (in the United States or Puerto Rico), call the IBM Software Support Center anytime (1-800-IBM®-SERV). You will receive a return call within 8 business hours (Monday – Friday, 8:00 a.m. – 5:00 p.m., local customer time).

Outside the United States or Puerto Rico, contact your local IBM representative or your authorized IBM supplier.

If you would like to provide feedback on this publication, see "Communicating your comments to IBM" on page 163.

Conventions and terminology that are used in this information

Commands in this information that can be used in both TSO and z/OS UNIX environments use the following conventions:

- When describing how to use the command in a TSO environment, the command is presented in uppercase (for example, NETSTAT).
- When describing how to use the command in a z/OS UNIX environment, the command is presented in bold lowercase (for example, **netstat**).
- When referring to the command in a general way in text, the command is presented with an initial capital letter (for example, Netstat).

All the exit routines described in this information are *installation-wide exit routines*. The installation-wide exit routines also called installation-wide exits, exit routines, and exits throughout this information.

The TPF logon manager, although included with VTAM®, is an application program; therefore, the logon manager is documented separately from VTAM.

Samples used in this information might not be updated for each release. Evaluate a sample carefully before applying it to your system.

Note: In this information, you might see the following Shared Memory Communications over Remote Direct Memory Access (SMC-R) terminology:

- RoCE Express®, which is a generic term representing IBM 10 GbE RoCE Express, IBM 10 GbE RoCE Express2, and IBM 25 GbE RoCE Express2 feature capabilities. When this term is used in this information, the processing being described applies to both features. If processing is applicable to only one feature, the full terminology, for instance, IBM 10 GbE RoCE Express will be used.
- RoCE Express2, which is a generic term representing an IBM RoCE Express2® feature that might operate in either 10 GbE or 25 GbE link speed. When this term is used in this information, the processing being described applies to either link speed. If processing is applicable to only one link speed, the full terminology, for instance, IBM 25 GbE RoCE Express2 will be used.
- RDMA network interface card (RNIC), which is used to refer to the IBM® 10 GbE RoCE Express, IBM® 10 GbE RoCE Express2, or IBM 25 GbE RoCE Express2 feature.
- Shared RoCE environment, which means that the "RoCE Express" feature can be used concurrently, or shared, by multiple operating system instances. The feature is considered to operate in a shared RoCE environment even if you use it with a single operating system instance.

Clarification of notes

Information traditionally qualified as Notes is further qualified as follows:

Attention
Indicate the possibility of damage

Guideline
Customary way to perform a procedure

Note
Supplemental detail

Rule
Something you must do; limitations on your actions

Restriction
Indicates certain conditions are not supported; limitations on a product or facility

Requirement
Dependencies, prerequisites

Result
Indicates the outcome

Tip
Offers shortcuts or alternative ways of performing an action; a hint

Prerequisite and related information

z/OS Communications Server function is described in the z/OS Communications Server library. Descriptions of those documents are listed in "Bibliography" on page 153, in the back of this document.

Required information

Before using this product, you should be familiar with TCP/IP, VTAM, MVS, and UNIX System Services.

Softcopy information

Softcopy publications are available in the following collection.

Titles	Description
IBM Z Redbooks	The IBM Z®® subject areas range from e-business application development and enablement to hardware, networking, Linux, solutions, security, parallel sysplex, and many others. For more information about the Redbooks® publications, see http://www.redbooks.ibm.com/ and http://www.ibm.com/systems/z/os/zos/zfavorites/.

Other documents

This information explains how z/OS references information in other documents.

When possible, this information uses cross-document links that go directly to the topic in reference using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see z/OS Information Roadmap (SA23-2299). The Roadmap describes what level of documents are supplied with each release of z/OS Communications Server, and also describes each z/OS publication.

To find the complete z/OS library, visit the z/OS library in IBM Knowledge Center (www.ibm.com/support/knowledgecenter/SSLTBW/welcome).

Relevant RFCs are listed in an appendix of the IP documents. Architectural specifications for the SNA protocol are listed in an appendix of the SNA documents.

The following table lists documents that might be helpful to readers.

Title	Number
DNS and BIND, Fifth Edition, O'Reilly Media, 2006	ISBN 13: 978-0596100575
Routing in the Internet, Second Edition, Christian Huitema (Prentice Hall 1999)	ISBN 13: 978-0130226471
sendmail, Fourth Edition, Bryan Costales, Claus Assmann, George Jansen, and Gregory Shapiro, O'Reilly Media, 2007	ISBN 13: 978-0596510299
SNA Formats	GA27-3136
TCP/IP Illustrated, Volume 1: The Protocols, W. Richard Stevens, Addison-Wesley Professional, 1994	ISBN 13: 978-0201633467
TCP/IP Illustrated, Volume 2: The Implementation, Gary R. Wright and W. Richard Stevens, Addison-Wesley Professional, 1995	ISBN 13: 978-0201633542
TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP, and the UNIX Domain Protocols, W. Richard Stevens, Addison-Wesley Professional, 1996	ISBN 13: 978-0201634952
TCP/IP Tutorial and Technical Overview	GG24-3376
Understanding LDAP	SG24-4986
z/OS Cryptographic Services System SSL Programming	*SC14-7495*
z/OS IBM Tivoli Directory Server Administration and Use for z/OS	*SC23-6788*
z/OS JES2 Initialization and Tuning Guide	*SA32-0991*
z/OS Problem Management	*SC23-6844*
z/OS MVS Diagnosis: Reference	*GA32-0904*
z/OS MVS Diagnosis: Tools and Service Aids	*GA32-0905*
z/OS MVS Using the Subsystem Interface	*SA38-0679*
z/OS Program Directory	*GI11-9848*
z/OS UNIX System Services Command Reference	*SA23-2280*
z/OS UNIX System Services Planning	*GA32-0884*
z/OS UNIX System Services Programming: Assembler Callable Services Reference	*SA23-2281*
z/OS UNIX System Services User's Guide	*SA23-2279*
z/OS XL C/C++ Runtime Library Reference	*SC14-7314*
z Systems: Open Systems Adapter-Express Customer's Guide and Reference	*SA22-7935*

Redbooks publications

The following Redbooks publications might help you as you implement z/OS Communications Server.

Title	Number
IBM z/OS Communications Server TCP/IP Implementation, Volume 1: Base Functions, Connectivity, and Routing	SG24-8096
IBM z/OS Communications Server TCP/IP Implementation, Volume 2: Standard Applications	SG24-8097

Title	Number
IBM z/OS Communications Server TCP/IP Implementation, Volume 3: High Availability, Scalability, and Performance	SG24-8098
IBM z/OS Communications Server TCP/IP Implementation, Volume 4: Security and Policy-Based Networking	SG24-8099
IBM Communication Controller Migration Guide	SG24-6298
IP Network Design Guide	SG24-2580
Managing OS/390 TCP/IP with SNMP	SG24-5866
Migrating Subarea Networks to an IP Infrastructure Using Enterprise Extender	SG24-5957
SecureWay Communications Server for OS/390 V2R8 TCP/IP: Guide to Enhancements	SG24-5631
SNA and TCP/IP Integration	SG24-5291
TCP/IP in a Sysplex	SG24-5235
TCP/IP Tutorial and Technical Overview	GG24-3376
Threadsafe Considerations for CICS	SG24-6351

Where to find related information on the Internet

z/OS

This site provides information about z/OS Communications Server release availability, migration information, downloads, and links to information about z/OS technology

http://www.ibm.com/systems/z/os/zos/

z/OS Internet Library

Use this site to view and download z/OS Communications Server documentation

http://www.ibm.com/systems/z/os/zos/library/bkserv/

IBM Communications Server product

The primary home page for information about z/OS Communications Server

http://www.software.ibm.com/network/commserver/

z/OS Communications Server product

The page contains z/OS Communications Server product introduction

http://www.ibm.com/software/products/en/commserver-zos

IBM Communications Server product support

Use this site to submit and track problems and search the z/OS Communications Server knowledge base for Technotes, FAQs, white papers, and other z/OS Communications Server information

http://www.software.ibm.com/support

IBM Communications Server performance information

This site contains links to the most recent Communications Server performance reports

http://www.ibm.com/support/docview.wss?uid=swg27005524

IBM Systems Center publications

Use this site to view and order Redbooks publications, Redpapers, and Technotes

http://www.redbooks.ibm.com/

IBM Systems Center flashes

Search the Technical Sales Library for Techdocs (including Flashes, presentations, Technotes, FAQs, white papers, Customer Support Plans, and Skills Transfer information)

http://www.ibm.com/support/techdocs/atsmastr.nsf

Tivoli® NetView® for z/OS

Use this site to view and download product documentation about Tivoli NetView for z/OS

http://www.ibm.com/support/knowledgecenter/SSZJDU/welcome

RFCs

Search for and view Request for Comments documents in this section of the Internet Engineering Task Force website, with links to the RFC repository and the IETF Working Groups web page

http://www.ietf.org/rfc.html

Internet drafts

View Internet-Drafts, which are working documents of the Internet Engineering Task Force (IETF) and other groups, in this section of the Internet Engineering Task Force website

http://www.ietf.org/ID.html

Information about web addresses can also be found in information APAR II11334.

Note: Any pointers in this publication to websites are provided for convenience only and do not serve as an endorsement of these websites.

DNS websites

For more information about DNS, see the following USENET news groups and mailing addresses:

USENET news groups
comp.protocols.dns.bind

BIND mailing lists
https://lists.isc.org/mailman/listinfo

BIND Users

- Subscribe by sending mail to bind-users-request@isc.org.
- Submit questions or answers to this forum by sending mail to bind-users@isc.org.

BIND 9 Users (This list might not be maintained indefinitely.)

- Subscribe by sending mail to bind9-users-request@isc.org.
- Submit questions or answers to this forum by sending mail to bind9-users@isc.org.

The z/OS Basic Skills Information Center

The z/OS Basic Skills Information Center is a web-based information resource intended to help users learn the basic concepts of z/OS, the operating system that runs most of the IBM mainframe computers in use today. The Information Center is designed to introduce a new generation of Information Technology professionals to basic concepts and help them prepare for a career as a z/OS professional, such as a z/OS systems programmer.

Specifically, the z/OS Basic Skills Information Center is intended to achieve the following objectives:

- Provide basic education and information about z/OS without charge
- Shorten the time it takes for people to become productive on the mainframe
- Make it easier for new people to learn z/OS

To access the z/OS Basic Skills Information Center, open your web browser to the following website, which is available to all users (no login required): https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zbasics/homepage.html?cp=zosbasics

Summary of changes for IPv6 Network and Application Design Guide

This document contains terminology, maintenance, and editorial changes, including changes to improve consistency and retrievability. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Changes made in z/OS Communications Server Version 2 Release 4

This information contains no technical change for this release.

Changes made in z/OS Communications Server Version 2 Release 3

This document contains information previously presented in z/OS Communications Server: IPv6 Network and Application Design Guide, which supported z/OS Version 2 Release 2.

Changed information

- IPv6 getaddrinfo() API standards compliance, see "Protocol-independent node name and service name translation" on page 73.

Changes made in z/OS Version 2 Release 2

This document contains information previously presented in z/OS Communications Server: IPv6 Network and Application Design Guide, SC27-3663-00, which supported z/OS Version 2 Release 1.

Changed information

- Removed support for the GATEWAY statement in the TCP/IP profile, see the following topics:
  - "TCP/IP profile configuration statements for configuring IPv6" on page 54
  - "z/OS-specific features" on page 121

Chapter 1. Internet Protocol Version 6

Internet Protocol Version 6 (IPv6) is the next generation of the Internet protocol designed to replace the current version, Internet Protocol Version 4 (IPv4). Most of today's internets use IPv4, for which there is a growing shortage of addresses. In theory, 32 bits provide over 4 billion nodes, each with a globally unique address. In practice, the interaction between routing and addressing makes it impossible to use more than a small fraction of that number of nodes. Consequently, there is a growing concern that the continued growth of the Internet might lead to the exhaustion of IPv4 addresses early in the 21st century.

IPv6 fixes a number of problems in IPv4, such as the limited number of available IPv4 addresses. IPv6 uses 128-bit addresses, an address space large enough to last for the foreseeable future. It also adds many improvements to IPv4 in areas such as routing and network autoconfiguration. IPv6 is expected to gradually replace IPv4, with the two coexisting for a number of years during a transition period.

IPv6 is an evolutionary step from IPv4. Functions that work well in IPv4 were kept in IPv6, and functions that did not work well in IPv4 were removed.

z/OS Communications Server Version 1 Release 4 was the first release to incorporate IPv6 features. With z/OS Communications Server, you can accomplish the following tasks:

- Build an IPv6 network
- Start using IPv6-enabled applications
- Enable existing IPv4 applications to be IPv6 applications
- Access your SNA applications over an IPv6 network

Not all IPv6 features are supported by z/OS. This information describes the support available and how to implement it.

IPv6 provides the following advantages.

Expanded routing and addressing

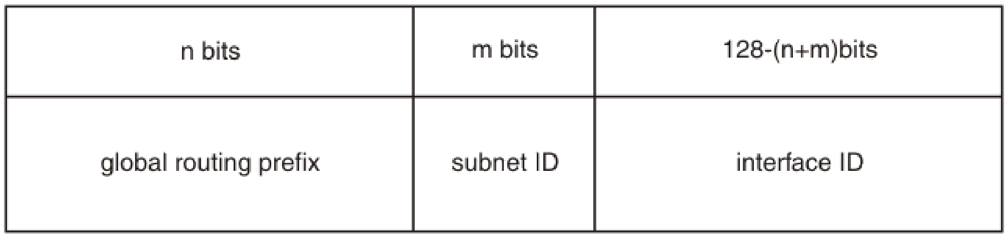
IPv6 uses a 128-bit address space, which has no practical limit on global addressability and provides 3.4 $\times 10^{50}$ unique addresses. This provides enough addresses so that every person could have a single IPv6 network with many nodes, and still the address space would be almost unused.

The greater availability of IPv6 addresses eliminates the need for private address spaces, which in turn eliminates one of the needs for network address translators (NATs) to be used between the private intranet and the public Internet.

Hierarchical addressing and routing infrastructure

The use of hierarchical address formats is equally important as the expanded address space. The IPv4 addressing hierarchy includes network, subnet, and host components in an IPv4 address. With its 128-bit addresses, IPv6 provides globally unique and hierarchical addressing based on prefixes rather than address classes, which keeps routing tables small and backbone routing efficient.

The general format is shown in the following figure:



n bits	m bits	128-(n+m)bits
global routing prefix	subnet ID	interface ID

Figure 1. IPv6 address space

The global routing prefix is a value (typically hierarchically structured) assigned to a site; the subnet ID is an identifier of a link within the site; and the interface ID is a unique identifier for a network device on a given link (usually automatically assigned).

Simplified IP header format

The IPv6 header has a fixed size and its format is more simplified than the IPv4 header. Some fields in the IPv4 header were dropped in IPv6 or moved to optional IPv6 extension headers to reduce the common-case processing cost of packet handling, as well as keep the bandwidth cost of the IPv6 header as low as possible despite increasing the size of addresses. While the IPv6 address is four times the size of the IPv4 address, the total IPv6 header size is only twice as large as the IPv4 header size.

Improved support for options

Changes in the way IP header options are encoded allows for more efficient forwarding, less stringent limits on the length of options, and greater flexibility for introducing new options in the future. Optional IPv6 header information is conveyed in independent extension headers located after the IPv6 header and before the transport-layer header in each packet. In contrast to IPv4, most IPv6 extension headers are not examined or processed by intermediate nodes.

Address autoconfiguration

IPv6 provides for both stateless and stateful autoconfiguration. Stateless autoconfiguration allows a node to be configured in the absence of any configuration server. Stateless autoconfiguration also makes it possible for a node to configure its own globally routable addresses in cooperation with a local IPv6 router, by combining the 48- or 64-bit MAC address of the adapter with network prefixes that are learned from the neighboring router.

IPv6 allows the use of DHCPv6 for stateful autoconfiguration. DHCPv6 relies on a configuration server that maintains static tables to determine the addresses that are assigned to newly connected nodes. z/OS Communications Server does not support DHCPv6.

Tip: You can manually configure addresses in environments in which complete local control is required (as with VIPA or additional LOOPBACK addresses).

Dual-mode stack support

z/OS Communications Server can be an IPv4-only stack or a dual-mode stack. Dual-mode stack refers to a single TCP/IP stack supporting both IPv4 and IPv6 protocols at the same time.

Restriction: IPv6-only stacks are not supported.

Running in a dual-mode stack configuration provides the following advantages:

- IPv4 and IPv6 applications can coexist on a single dual-mode stack.
- Unmodified applications can continue to send data over an IPv4 network.
- A single IPv6-enabled application can communicate using IPv4 and IPv6.
- IPv4 and IPv6 can coexist in the same devices and networks.

Neighbor discovery

Neighbor discovery (ND) corresponds to a combination of the IPv4 protocols ARP, ICMP Router Discovery, and ICMP Redirect. Nodes (hosts and routers) use ND to determine the link-layer addresses for neighbors that are known to reside on attached links and to quickly purge cached values that become invalid. Hosts also use ND to find neighboring routers that are able to forward packets on their behalf. ND also defines a Neighbor Unreachability Detection algorithm. IPv4 does not contain a generally agreed upon protocol for performing Neighbor Unreachability Detection, although Dead Gateway Detection does address a subset of the problems that Neighbor Unreachability Detection solves.

Neighbor Discovery is used to do the following tasks:

- Obtain configuration information that includes:

  Router Discovery
  Defines how hosts can automatically locate routers that reside on an attached link.

  Prefix Discovery
  Specifies how hosts discover the following sets of prefixes:

  - Prefixes that are defined as being on-link (IPv6 address prefixes that reside on the shared link, such as an Ethernet link)
  - Prefixes that are defined as being off-link (IPv6 address prefixes that can be reached by using an adjacent router)
  - Prefixes that are to be used for Stateless Address Autoconfiguration

  Parameter Discovery
  Allows a host to learn link parameters, such as the link MTU, and IP parameters, such as the hop limit to place in outgoing packets.

- Perform address resolution. Address resolution allows a node to determine the link-layer address of an on-link destination given the destination IP address.
- Dynamically learn routes which can be used in next-hop determination. This specifies the algorithm for mapping the IP destination address into the IP address of the neighbor to which traffic is to be sent. The next-hop can be either a router or the destination itself. Next-hop determination uses the on-link prefixes learned as part of Prefix Discovery to determine when the next hop is the destination itself.
- Determine when a neighbor is no longer reachable using Neighbor Unreachability Detection.
- Process Redirect messages. Routers use Redirect messages to notify a node that a better next-hop node is to be used when forwarding packets to a particular destination. The new next-hop could be the actual destination, if the destination is on-link, or a different router, if the destination is off-link.

Comparison of IPv6 and IPv4 characteristics

There are major differences between IPv4 and IPv6. Table 1 on page 4 lists these differences.

Table 1. Comparison of IPv4 and IPv6

IPv4	**IPv6**
Source and destination addresses are 32 bits (4 bytes) in length.	Source and destination addresses are 128 bits (16 bytes) in length. For more information, see Chapter 2, "IPv6 addressing," on page 7.
Uses broadcast addresses to send traffic to all nodes on a subnet.	There are no IPv6 broadcast addresses. Instead, multicast scoped addresses are used. For more information, see "Multicast scope" on page 14.
Fragmentation is supported at originating hosts and intermediate routers.	Fragmentation is not supported at routers. It is supported at the originating host only. For more information, see "Fragmentation in an IPv6 network" on page 17.
IP header includes a checksum.	IP header does not include a checksum.
IP header includes options.	All optional data is moved to IPv6 extension headers. For more information, see "Extension headers" on page 17.
IPSec support is optional.	IPSec support is required in a full IPv6 implementation.
No identification of payload for QoS handling by routers is present within the IPv4 header.	Payload identification for QoS handling by routers is included in the IPv6 header using the `Flow Label` field. For more information, see "Option to provide QoS classification data" on page 105.
ICMP Router Discovery is used to determine the IPv4 address of the best default gateway and is optional.	Uses ICMPv6 Router Solicitation and Router Advertisement to determine the IPv6 address of the best default gateway and is a required function. For more information, see "Router advertisements" on page 24. z/OS sends router solicitations and processes router advertisements but does not send router advertisements.
Address Resolution Protocol (ARP) uses broadcast ARP Request frames to resolve an IPv4 address to a link layer address.	Uses multicast Neighbor Solicitation messages for address resolution. For more information, see "Address resolution" on page 29.
Internet Group Management Protocol (IGMP) is used to manage local subnet group membership.	Uses Multicast Listener Discovery (MLD) messages to manage local subnet group membership. For more information, see "Multicast Listener Discovery" on page 23.
Addresses must be configured either manually or through DHCP. (DHCP is not supported in z/OS Communications Server.)	Addresses can be automatically assigned using stateless address autoconfiguration, assigned using DHCPv6, or manually configured. (DHCPv6 is not supported in z/OS Communications Server.)
Uses host address (A) resource records in the Domain Name System (DNS) to map host names to IPv4 addresses.	Uses host address (AAAA) resource records in the Domain Name System (DNS) to map host names to IPv6 addresses.
Uses pointer (PTR) resource records in the IN-ADDR.ARPA DNS domain to map IPv4 addresses to host names.	Uses pointer (PTR) resource records in the IP6.ARPA or IP6.INT DNS domain to map IPv6 addresses to host names.

Table 1. Comparison of IPv4 and IPv6 (continued)

IPv4	**IPv6**
For QoS, IPv4 supports both differentiated and integrated services.	Differentiated and integrated services are both supported. In addition, IPv6 provides a flow label that can be used for more granular treatment of packets.

Chapter 2. IPv6 addressing

This topic contains the following topics:

Textual representation of IPv6 addresses

IPv4 addresses are represented in dotted decimal format. The 32-bit address is divided along 8-bit boundaries. Each set of 8 bits is converted to its decimal equivalent and separated by periods. In contrast, IPv6 addresses are 128 bits divided along 16-bit boundaries. Each 16-bit block is converted to a 4-digit hexadecimal number and separated by colons. The resulting representation is called colon-hexadecimal.

The following forms are the three conventional forms for representing IPv6 addresses as text strings:

- The preferred form is x:x:x:x:x:x:x:x, where the x's are the hexadecimal values of the eight 16-bit pieces of the address. For example:

  ```
  2001:DB8:7654:3210:FEDC:BA98:7654:3210
  ```

  ```
  2001:DB8:0:0:8:800:200C:417A
  ```

  Guideline: You do not need to write the leading zeros in an individual field, but there must be at least one numeral in every field (except for the case described in the following item).

- As a result of some methods of allocating certain styles of IPv6 addresses, sometimes addresses contain long strings of zero bits. To make writing addresses containing zero bits easier, a special syntax is available to compress the zeros. A double colon (::) indicates multiple groups of 16 bits of zeros and can appear only once in an address. The double colon can also be used to compress both leading and trailing zeros in an address.

  For example the following addresses:

Table 2. Address types

Address type	Long form	Compressed form
Unicast	2001:DB8:0:0:8:800:200C:417A	2001:DB8::8:800:200C:417A
Multicast	FF01:0:0:0:0:0:0:101	FF01::101
Loopback	0:0:0:0:0:0:0:1	::1
Unspecified	0:0:0:0:0:0:0:0	::

- An alternative form that is sometimes more convenient when dealing with a mixed environment of IPv4 and IPv6 nodes is x:x:x:x:x:x:d.d.d.d, where the x's are the hexadecimal values of the six high-order 16-bit pieces of the address, and the d's are the decimal values of the four low-order 8-bit pieces of the address (standard IPv4 representation). This form is used for IPv4-mapped IPv6 addresses. This type of address is used to hold an embedded IPv4 address. The address can be expressed in the following manner:

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

The address can also be expressed in compressed form:

```
::FFFF:129.144.52.38
```

Textual representation of IPv6 prefixes

The text representation of IPv6 address prefixes is similar to the way IPv4 address prefixes are written in Classless Inter-Domain Routing (CIDR) notation. An IPv6 address prefix is represented by the notation `ipv6-address/prefix-length` where:

ipv6-address
An IPv6 address in any of the notations listed.

prefix-length
A decimal value specifying how many of the leftmost contiguous bits of the address comprise the prefix.

The following examples are legal representations of the 60-bit prefix 20010DB80000CD3 (hexadecimal):

```
2001:0DB8:0000:CD30:0000:0000:0000:0000/60
2001:DB8::CD30:0:0:0:0/60
2001:DB8:0:CD30::/60
```

The following examples are not legal representations of the preceding prefix:

- 2001:DB8:0:CD3/60

  Leading zeros might be dropped, but not trailing zeros, within any 16-bit chunk of the address.
- 2001:DB8::CD30/60

  Address to the left of the forward slash (/) expands to 2001:DB8:0000:0000:0000:0000:0000:CD30.

When writing both a node address and a prefix of that node address (for example, the node's subnet prefix), the two can be combined as in the following examples:

- Node address - 2001:DB8:0:CD30:123:4567:89AB:CDEF
- Subnet number - 2001:DB8:0:CD30::/60
- Combination of node address and subnet number - 2001:DB8:0:CD30:123:4567:89AB:CDEF/60

IPv6 address space

The type of an IPv6 address is identified by the high-order bits of the address as shown in Table 3 on page 8.

Table 3. Address type representation

Address type	**Binary prefix**	**IPv6 notation**
Unspecified	00...0 (128 bits)	::/128
Loopback	00...1 (128 bits)	::1/128
Unique local unicast	1111110	FC00::/7
Multicast	11111111	FF00::/8

Table 3. Address type representation (continued)

Address type	**Binary prefix**	**IPv6 notation**
Link-local unicast	1111111010	FE80::/10
Global unicast aggregatable	(everything else)	

Anycast addresses are taken from the unicast address spaces (of any scope) and are not syntactically distinguishable from unicast addresses. Anycast is described as a cross between unicast and multicast. Like multicast, multiple nodes might be listening on an anycast address. Like unicast, a packet sent to an anycast address is delivered to one (and only one) of those nodes. The exact node to which it is delivered is based on the IP routing tables in the network.

For more information about different IPv6 addresses, see "Categories of IPv6 addresses" on page 10.

IPv6 addressing model

IPv6 unicast addresses of all types (excluding the unspecified address) can be assigned to node interfaces. The loopback address can be assigned only to the loopback interface of a node.

All physical interfaces (excluding VIPA and loopback) are required to have at least one link-local unicast address. z/OS Communications Server allows only a single link-local address per interface. Other platforms might have more than one. A single interface can be assigned multiple unicast or anycast IPv6 addresses. Multiple IPv6 multicast groups of any scope can be joined on a single interface. A unicast address or a set of unicast addresses might be assigned to multiple physical interfaces if the implementation treats the multiple physical interfaces as one interface when presenting it to the Internet layer.

Currently, IPv6 continues the IPv4 model that a subnet prefix is associated with one link. Multiple subnet prefixes can be assigned to the same link.

Scope zones

Each IPv6 address has a specific scope in which it is defined. A scope is a topological area within which the IPv6 address can be used as a unique identifier for an interface or a set of interfaces. The scope for an IPv6 address is encoded as part of the address itself. A unicast address can have a link-local or global scope. A multicast address supports:

- Interface-local
- Link-local
- Subnet-local
- Admin-local
- Site-local
- Organization-local
- Global scopes

See "Unicast IPv6 addresses" on page 10 and "Multicast IPv6 addresses" on page 13 for more discussions about unicast and multicast scopes.

A scope zone is an instance of a given scope. For instance, a link and all directly attached interfaces comprise a single link-local scope zone. A scope zone has the following properties:

- A scope zone consists of a contiguous set of interfaces and the links to which the interfaces are attached.
- An interface can belong to only one scope zone of each possible scope.

- A node can be connected to more than one scope zone of a given scope. For instance, a node can be connected to multiple link-local scope zones if it is attached to more than one LAN.
- The scope zone for an IPv6 address is not encoded within the address itself, but is instead determined by the interface over which the packet is sent or received.
- There is a single scope zone for IPv6 addresses of global scope which comprises all interfaces and links in the Internet.
- Packets that contain a source or destination address of a given scope can be routed only within the same scope zone, and cannot be routed between different scope zone instances.
- Addresses of a given scope can be reused in different scope zones.
- Scope zones associated with the inbound and intended outbound interfaces are compared to determine whether packets containing a limited scope address (for example, an address of scope other than global) can be successfully routed.
- Scope zone representations (zone indices) are valid only on the node where they are defined. The same zone can have separate representations in each node that belongs to that zone.

To identify a specific instance of a scope zone, a node assigns a unique scope zone index to each scope zone of the same scope to which it is attached.

Categories of IPv6 addresses

An IPv6 address is identified by the high-order bits of the address. The following categories of IP addresses are supported in IPv6:

Unicast
An identifier for a single interface. A packet sent to a unicast address is delivered to the interface identified by that address. It can be link-local scope or global scope.

Multicast
An identifier for a group of interfaces (typically belonging to different nodes). A packet sent to a multicast address is delivered to all interfaces identified by that address.

Anycast
An identifier for a group of interfaces (typically belonging to different nodes). A packet sent to an anycast address is delivered to the closest member of a group, according to the routing protocols' measure of distance.

Restriction: Although z/OS Communications Server can send or forward datagrams to an anycast address, z/OS Communications Server does not support functioning as an anycast endpoint.

There are no broadcast addresses in IPv6. Multicast addresses have superseded this function.

Unicast IPv6 addresses

IPv6 unicast addresses can be aggregated with prefixes of arbitrary bit-length similar to IPv4 addresses under Classless Inter-Domain Routing (CIDR).

There are two types of unicast addresses in IPv6:

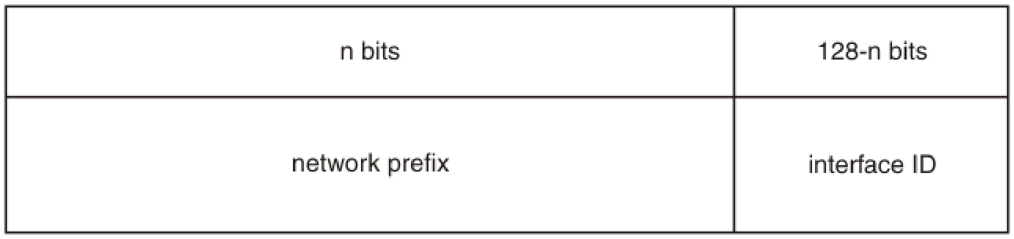
- Global unicast
- Link-local unicast

There is also a special-purpose subtype of global unicast:

- IPv6 addresses with embedded IPv4 addresses

Additional address types or subtypes can be defined in the future.

A unicast address has the following format:



n bits	128-n bits
network prefix	interface ID

Figure 2. Unicast address format

Aggregatable global addresses

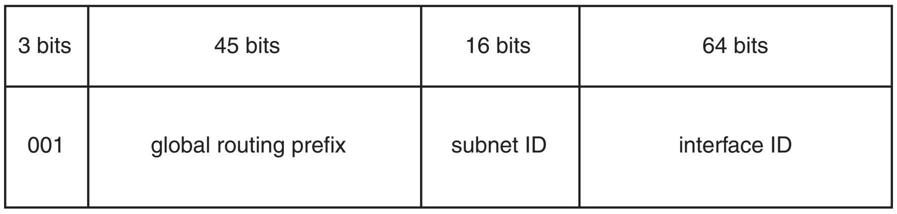
Aggregatable global unicast addresses are equivalent to public IPv4 addresses. They are globally routable and reachable on the IPv6 portion of the Internet.

A global unicast address has the following format:

Global routing prefix
: Used to identify a specific customer site. The size of the field is 48 bits and allows an ISP to create multiple levels of addressing hierarchy within the network to both organize addressing and routing for downstream ISPs and identify sites.

Subnet ID
: Used by an individual organization to identify subnets within its site. The organization can use these 16 bits to create 65536 subnets or multiple levels of addressing hierarchy.

Interface ID
: Indicates the interface on a specific subnet. The size of this field is 64 bits.

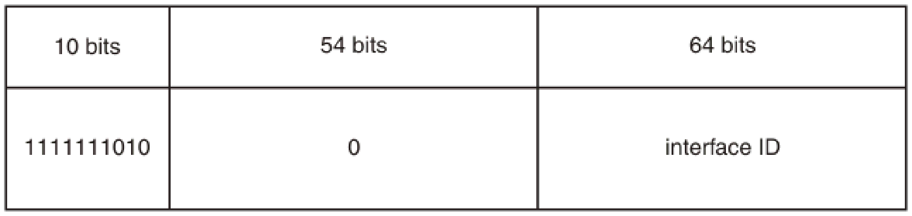


3 bits	45 bits	16 bits	64 bits
001	global routing prefix	subnet ID	interface ID

Figure 3. Global unicast address format

Local-use addresses

The link-local address is the one type of local-use unicast address defined. The link-local address is for use on a single link. Link-local addresses have the following format:



10 bits	54 bits	64 bits
1111111010	0	interface ID

Figure 4. Link-local address format

Restriction: A link-local address is required on each physical interface.

Link-local addresses are designed to be used for addressing on a single link for purposes such as automatic address configuration, neighbor discovery, or in the absence of routers. It also can be used to communicate with other nodes on the same link. A link-local address is automatically assigned.

Routers do not forward any packets with link-local source or destination addresses to other links.

Loopback address

The unicast address 0:0:0:0:0:0:0:1 is called the loopback address. It cannot be assigned to any physical or VIPA interface. It can be thought of as a link-local unicast address assigned to a virtual interface (typically called the loopback interface) that allows local applications to send messages to each other.

Restriction: The loopback address cannot be used as the source address in IPv6 packets that are sent outside of a node. An IPv6 packet with a destination address of loopback cannot be sent outside of a node and be forwarded by an IPv6 router. A packet received on an interface with destination address of loopback is dropped.

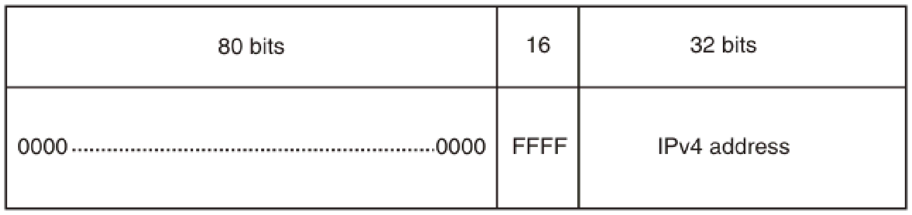
Unspecified address

The address 0:0:0:0:0:0:0:0 is called the unspecified address. It is not assigned to any node. It indicates the absence of an address. One example of its use is in the Source Address field of any IPv6 packets sent by an initializing host before it has learned its own address.

Restriction: The unspecified address cannot be used as the destination address of IPv6 packets or in IPv6 routing headers. An IPv6 packet with a source address of unspecified cannot be forwarded by an IPv6 router.

IPv4-mapped IPv6 addresses

These addresses hold an embedded global IPv4 address. They are used to represent the addresses of IPv4 nodes as IPv6 addresses to applications that are enabled for IPv6 and are using AF_INET6 sockets. This allows IPv6-enabled applications to always deal with IP addresses in IPv6 format regardless of whether the TCP/IP communications are occurring over IPv4 or IPv6 networks. The dual-mode TCP/IP stack performs the transformation of the IPv4-mapped addresses to and from native IPv4 format. IPv4-mapped addresses have the following format:



80 bits	16	32 bits
0000...0000	FFFF	IPv4 address

Figure 5. IPv4-mapped IPv6 address

For example:

```
::FFFF:129.144.52.38
```

IPv6 interface identifiers

Interface identifiers in IPv6 unicast addresses are used to identify interfaces on a link. They are required to be unique on that link. In some cases, an interface's identifier is derived directly from that interface's link-layer address. z/OS Communications Server does not allow two links to have the same local address. Some implementations might allow the same interface identifier to be used on multiple interfaces on a single node, as long as they are attached to different links.

z/OS Communications Server allows the interface identifier to be generated (the default) or manually configured. When the interface ID is generated, then z/OS builds the interface ID when the interface becomes active based on the interface type as follows:

- OSA-Express QDIO
  1. OSA-Express returns the MAC address and a unique instance value during the start of an interface.
  2. z/OS builds the interface identifier by inserting the unique instance value into the middle of the MAC address. This ensures that when multiple stacks share an OSA, each stack gets a unique interface ID. If a virtual MAC address is configured for this interface, then z/OS instead inserts the value 'FFFE'x into the middle of the MAC address.

- HiperSockets

  For HiperSockets interfaces, the interface ID generation works the same as for OSA-Express QDIO except that the HiperSockets device returns a 48-bit value that is unique for the HiperSockets CHPID rather than a MAC address. This ensures that when multiple stacks share a HiperSockets CHPID, each stack gets a unique interface ID.
- MPCPTP6

  For MPCPTP6 interfaces, z/OS randomly generates an interface ID.

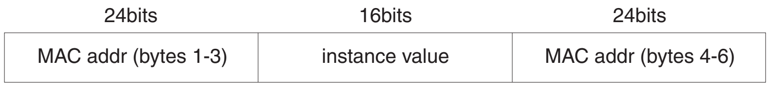


Figure 6. OSA-Express QDIO interface ID format

A node can choose to use a different algorithm available for generation of interface identifiers for IPv6 addresses on a different platform.

Randomly generated temporary IPv6 interface identifiers

In addition to the interface identifier that is derived directly from the link-layer address of the interface or that is manually configured, z/OS can also generate a random interface identifier for OSA-Express QDIO interfaces. The random interface identifier is used to generate temporary IPv6 addresses. A randomly generated interface identifier is regenerated after a specified time interval. See "IPv6 temporary addresses with random interface IDs" on page 32 for more information.

Multicast IPv6 addresses

An IPv6 multicast address is an identifier for a group of interfaces (typically on different nodes). It is identified with a prefix of 11111111 or FF in hexadecimal notation. It provides a way of sending packets to multiple destinations. An interface can belong to any number of multicast groups.

Multicast address format

Binary 11111111 at the start of the address identifies the address as being a multicast address. Multicast addresses have the following format:

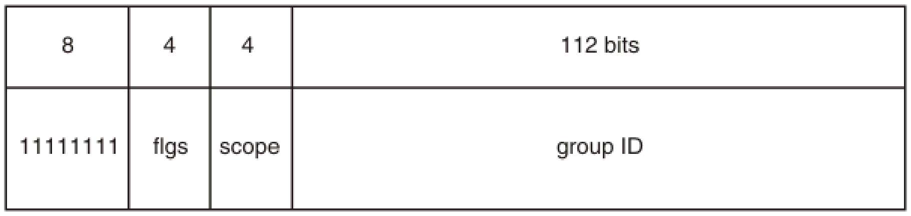


Figure 7. Multicast address format

flgs is a set of 4 flags:

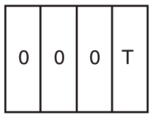


Figure 8. Flags in multicast address

- The 3 high-order flags are reserved, and must be initialized to 0.
- T = 0 indicates a permanently-assigned (well-known) multicast address, assigned by the Internet Assigned Number Authority (IANA).
- T = 1 indicates a non-permanently assigned (transient) multicast address.

Scope is a 4-bit multicast scope value used to limit the scope of the multicast group. Group ID identifies the multicast group, either permanent or transient, within the given scope.

Multicast scope

The scope field indicates the scope of the IPv6 internetwork for which the multicast traffic is intended. The size of this field is 4 bits. In addition to information provided by multicast routing protocols, routers use multicast scope to determine whether multicast traffic can be forwarded. For multicast addresses there are 14 possible scopes (some are still unassigned), ranging from interface-local to global (including both link-local and site-local).

Table 4 on page 14 lists the defined values for the scope field:

Table 4. Multicast scope field values

Value	Scope
0	Reserved
1	Interface-local scope (same node)
2	Link-local scope (same link)
3	Subnet-local scope
4	Admin-local scope
5	Site-local scope (same site)
8	Organization-local scope
E	Global scope
F	Reserved

Note: All other scope field values are currently undefined.

For example, traffic with the multicast address of FF0**2**::2 has a link-local scope. An IPv6 router never forwards this type of traffic beyond the local link.

Interface-local
The interface-local scope spans a single interface only. A multicast address of interface-local scope is useful only for loopback delivery of multicasts within a node, for example, as a form of interprocess communication within a computer. Unlike the unicast loopback address, interface-local multicast addresses can be joined on any interface.

Link-local
Link-local addresses are used by nodes when communicating with neighboring nodes on the same link. The scope of the link-local address is the local link.

Subnet-local
Subnet-local scope is given a different and larger value than link-local to enable possible support for subnets that span multiple links.

Admin-local
Admin-local scope is the smallest scope that must be administratively configured, that is, not automatically derived from physical connectivity or other, non-multicast-related configuration.

Site-local
The scope of a site-local address is the site or organization internetwork. Addresses must remain within their scope. A router must not forward packets outside of its scope.

Organization-local
This scope is intended to span multiple sites belonging to a single organization.

Global
Global scope is used for uniquely identifying interfaces anywhere in the Internet.

Multicast groups

Group ID identifies the multicast group, either permanent or transient, within the given scope. The size of this field is 112 bits. Permanently assigned groups can use the group ID with any scope value and still refer to the same group. Transient assigned groups can use the group ID in different scopes to refer to different groups. Multicast addresses from FF01:: through FF0F:: are reserved, well-known addresses. Use of these group IDs for any other scope values, with the T flag equal to 0, is not allowed.

All-nodes multicast groups

These groups identify all IPv6 nodes within a given scope. Defined groups include:

- Interface-local all-nodes group (FF01::1)
- Link-local all-nodes group (FF02::1)

All-routers multicast groups

These groups identify all IPv6 routers within a given scope. Defined groups include the following groups:

- Interface-local all-routers group (FF01::2)
- Link-local all-routers group (FF02::2)
- Site-local all-routers group (FF05::2)

Solicited-node multicast group

For each unicast address which is assigned to an interface, the associated solicited-node multicast group is joined on that interface. The solicited-node multicast address facilitates the efficient querying of network nodes during address resolution.

Anycast IPv6 addresses

An IPv6 anycast address is an identifier for a set of interfaces (typically belonging to different nodes). A packet sent to an anycast address is delivered to one of the interfaces identified by that address (the nearest interface), according to the routing protocols' measure of distance. It uses the same formats as a unicast address, so one cannot differentiate between a unicast and an anycast address simply by examining the address. Instead, anycast addresses are defined administratively.

Typical IPv6 addresses assigned to a node

An IPv6 host is required to recognize the following addresses as identifying itself:

- Link-local address for each active IPv6 physical interface (cannot be manually defined)
- Assigned unicast addresses (autoconfigured or manually defined)
- IPv6 loopback address (::1)
- All-nodes multicast address (interface-local and link-local)
- Solicited node multicast addresses for each of its assigned unicast and anycast addresses
- Multicast addresses of all other groups to which the host belongs

IPv6 address states

An address state defines and controls how other algorithms work with a particular address. There are four IPv6 address states: tentative, deprecated, preferred, and unavailable.

Tentative

A tentative address is an address whose uniqueness on a link is being verified before it is assigned to the interface. A tentative address is not considered assigned to the interface in the usual sense. An interface discards received packets that are addressed to a tentative address, unless those packets are related to

Duplicate Address Detection (DAD). For more information about DAD, see "Duplicate address detection" on page 28.

Deprecated

A deprecated address is an address that is assigned to an interface, and use of the address is discouraged but not forbidden. Packets that are sent from or to deprecated addresses are delivered as expected. A deprecated address continues to be used as a source address in existing communications where switching to a preferred address would be disruptive.

Preferred

A preferred address is an address that is assigned to an interface, and use of the address is unrestricted. Preferred addresses can be used as the source or destination address of packets that are sent from or to the interface.

Unavailable

An unavailable address is one that is not yet assigned to the interface.

Chapter 3. IPv6 protocol

This topic describes the IPv6 protocol implementation and contains the following topics:

Guideline: You should be familiar with the IPv6 protocol in general.

Extension headers

In IPv6, IP-layer options within a packet are encapsulated in independent headers called extension headers. In contrast, IPv4 options are contained in the IP header itself.

Restriction: Not all IPv6 extension headers are supported in z/OS Communications Server. The z/OS TCP/IP stack supports receipt of the following extension headers:

- Routing
- Fragmentation
- Hop-by-hop option
- Destination option
- Authentication (AH)
- Encapsulating Security Payload (ESP)

Fragmentation in an IPv6 network

Fragmentation is used by a source to send a packet larger than would fit in the path MTU to its destination. To send packets larger than the link minimum of 1280 bytes, a node must support determination of the minimum supported MTU along the path between the source and destination. This is accomplished by Path MTU discovery. For more information about path discovery, see "Path MTU discovery" on page 18.

The IPv6 IP header does not contain information about fragments. The fragmentation extension header carries this information. z/OS Communications Server allows for 2048 active IPv6 reassemblies in progress at any given time. z/OS Communications Server reassembly timeout for IPv6 reassemblies is 60 seconds. These two values are not configurable.

Fragmentation and UDP/RAW

Intermediate routers cannot fragment packets and UDP/RAW transports do not perform retransmission. To attempt to ensure that a UDP/RAW packet is not dropped because of fragmentation, one of the following conditions can occur:

- z/OS Communications Server always sends the packet using the minimum MTU (1280) unless the MTU for the destination is learned from an ICMPv6 Packet Too Big message.
- An application sends a packet using the IPV6_DONTFRAG socket option.

For example, a situation can occur where the MTU was learned by way of Path MTU discovery. In that case, the network topology changes, reducing the MTU to this particular destination. UDP/RAW sends with the original learned MTU and receives a Packet Too Big message. In this case, the packet is dropped, but subsequent sends learn the changed MTU and send with the appropriate size.

Path MTU discovery

When one IPv6 node has a large amount of data to send to another node, the data is transmitted in a series of IPv6 packets. It is preferable that these packets be of the largest size that can successfully traverse the path from the source node to the destination node. This packet size is referred to as the Path MTU (PMTU), and it is equal to the minimum link MTU of all the links in a path. IPv6 provides PMTU discovery as a standard mechanism for a node to discover the PMTU of an arbitrary path.

For IPv6, intermediate routers cannot fragment packets. An implementation must either support path MTU discovery or send using IPv6 minimum link MTU. z/OS Communications Server supports path MTU discovery.

Path MTU discovery supports multicast as well as unicast destinations. When PMTU information is learned, it is cached for a period of time and then deleted in order to learn of increases in the MTU value.

IPv6 routing

Both replaceable and non-replaceable IPv6 static routes are configured for the main route table by using BEGINROUTES profile statements. Both replaceable and non-replaceable IPv6 static routes can also be configured for policy-based route tables. Policy-based routing is configured by using one of the following options:

- Use the IBM Configuration Assistant for z/OS Communications Server.
- Manually create the policy-based routing configuration files and code the required policy statements.

For more information about these two configuration options for configuring policy-based routing, see z/OS Communications Server: IP Configuration Guide.

Dynamic routes for IPv6 are learned in the following ways:

By router discovery
: For policy-based route tables, the configured policy controls which IPv6 router advertisement routes are added to each table.

From packets that are redirected by ICMPv6

From dynamic routing protocols
: For policy-based route tables, the configured policy controls which OSPF and RIP routes are added to each table.

Replaceable static routes can be replaced by dynamic routes. If a replaceable static route is replaced by a dynamic route, and that dynamic route is later deleted, the replaceable static route is readded.

Router discovery

Hosts can learn the network prefixes for all directly attached links from the router advertisements received from their routers. To determine whether another host is on a directly attached link or on a remote link, determine whether that host's IPv6 address is constructed from a network prefix of one of the directly attached links. If it is on a directly attached link, data can be sent directly to that host without going through a router; otherwise, data must be sent through a router using a default route or an indirect prefix route that can also be learned from router advertisements.

Router advertisements are not a replacement for dynamic routing protocols such as IPv6 OSPF and IPv6 RIP. If a host is not using a dynamic routing protocol, some limitations apply.

If the host has multiple interfaces attached to more than one link, the host must decide which interface to use when sending a packet to a host on a remote link. If there are multiple routers on the link attached to the interface, the host must decide to which router it should send the packet. To make these decisions, the host needs a route in its routing table. When both of the following criteria are true, only default routes are available for accessing a host on a remote link:

- Neither the IPv6 OSPF nor the IPv6 RIP dynamic routing protocol of OMPROUTE is being used.
- Adjacent routers are not including indirect prefix routes (using the Route Information option as described in RFC 4191) in their router advertisement messages.

When there are multiple default routers on the same physical link, the host might select a router that is not optimal. This selection might not be a serious problem, because that router can send an ICMP Redirect, which indicates that future packets should be sent to the optimal router. However, if the default routers are on multiple physical links, the results might be more serious. A router on one link is not able to redirect the host to use a different physical link. If the selected router cannot reach the destination, attempts to send data fail, even if the destination could be reached by a default router on another physical link. To resolve these limitations when you are not using a dynamic routing protocol, static routes might be needed to direct the traffic over the best interface and using the appropriate router.

If a dynamic routing protocol is not used, routes to VIPAs cannot be advertised. For this reason, use a network prefix defined as being on-link for the interfaces that are associated with the VIPA. In this way, routers and hosts perceive that the VIPA is on a physical interface and sends Neighbor Discovery messages (the IPv6 equivalent of an ARP request) to get the MAC address of the interface. This is not the best method for setting up VIPAs if a dynamic routing protocol is being used. It is better to associate VIPAs with interfaces on different LANs. Without a dynamic routing protocol, you can use a network prefix defined as being on-link for the associated interfaces or define static routes at all routers on the same links as the z/OS system.

See "Router advertisements" on page 24 for more information about how received router advertisements are processed.

ICMPv6 redirects

Routes that are learned when packets are redirected by ICMPv6 replace static routes regardless of whether they are replaceable. Use the IGNOREREDIRECT keyword on the IPCONFIG6 statement in the TCP/IP profile to prevent the stack from adding routes learned when ICMPv6 redirects packets.

Rule: These routes are always ignored when an IPv6 dynamic routing protocol is being used.

Dynamic routing protocols

The z/OS Communications Server OMPROUTE routing daemon supports the IPv6 OSPF and IPv6 RIP dynamic routing protocols. A host using one of these protocols can learn, from adjacent routers that are also using that protocol, the network prefixes and host addresses that can be reached.

IPv6 OSPF, IPv6 RIP, and router discovery can be used together in the same network.

- IPv6 OSPF allows the host to learn the network prefixes and host addresses that can be reached indirectly by way of adjacent IPv6 OSPF routers (including default routes), as well as the network prefixes that can be reached directly on attached links in the IPv6 OSPF domain.

- IPv6 RIP allows the host to learn the network prefixes and host addresses that can be reached indirectly by way of adjacent IPv6 RIP routers (including default routes).
- Router discovery allows the host to learn which network prefixes can be reached indirectly by way of adjacent, participating routers (including default routes), as well as which network prefixes can be reached directly on attached links.

In addition, the network prefixes that can be reached directly on attached links can be manually configured using the Prefix keyword on the IPv6_Interface, IPv6_OSPF_Interface, or IPv6_RIP_Interface statements in the OMPROUTE configuration file. When IPv6 OSPF or IPv6 RIP is used together with router discovery, the following kinds of routes can be learned from both methods:

- Default routes

  Default routes are learned from both methods if adjacent routers are advertising themselves as default routers using both IPv6 OSPF or IPv6 RIP and router discovery. When this situation occurs, the default routes learned from IPv6 OSPF or IPv6 RIP take precedence and generate the default routes in the TCPIP stack's IPv6 route table. Any default routes learned from router discovery are ignored as long as the default routes learned from IPv6 OSPF or IPv6 RIP exist.
- Prefix routes

  Prefix routes are learned from both router discovery and OMPROUTE under each of the following conditions:

  – A router is advertising by way of router discovery that the prefix is on-link and the prefix is also manually configured to OMPROUTE using the Prefix keyword on an IPv6_Interface, IPv6_OSPF_Interface, or IPv6_RIP_Interface configuration statement.

    Guideline: Use the Prefix keyword only when the prefix is not learned dynamically (using router discovery or a dynamic routing protocol).

    For example, if there is a need to supplement the list of prefixes being advertised as on-link by the routers. If the same prefix is configured using the Prefix keyword and learned from router discovery, the route in the TCPIP stack's route table is the route added by OMPROUTE as a result of the Prefix keyword. Any route for the same prefix that is learned from router discovery is ignored as long as the OMPROUTE route exists.

    Restriction: Prefixes learned from only OMPROUTE are not used for address autoconfiguration. If a prefix is learned from both OMPROUTE and router discovery, it can still be used for autoconfiguration even though the route learned from OMPROUTE is the one in the TCPIP stack route table.

  – A router is advertising by way of router discovery that either the prefix is on-link or the prefix can be reached by way of an adjacent router, and a router is also advertising by way of IPv6 OSPF that the prefix is on-link.

    In this case, the route in the TCPIP stack route table is the route added by OMPROUTE as a result of the information received by way of IPv6 OSPF. Any route for the same prefix that is learned from router discovery is ignored as long as the OMPROUTE route exists. As in the previous condition, an on-link prefix that is learned from router discovery can still be used for address autoconfiguration.

  – A router is advertising by way of router discovery that the prefix is on-link and it is also learned, by way of IPv6 OSPF or IPv6 RIP, that the prefix can be reached by way of an adjacent router.

    In this case, the route in the TCPIP stack route table is the route added as the result of router discovery. This occurs because the router discovery information indicates that the prefix resides on a directly attached link, while the IPv6 OSPF or IPv6 RIP information indicates that the prefix can be reached indirectly, by way of the router from which the IPv6 OSPF or IPv6 RIP information was received. Any route for the prefix that is learned from IPv6 OSPF or IPv6 RIP is ignored as long as the router discovery route exists.

  – Router discovery advertisements are received that indicate that the prefix can be reached by way of an adjacent router. In addition, IPv6 OSPF or IPv6 RIP advertisements are received that indicate that the prefix can be reached by way of an adjacent router.

In this case, the route in the TCPIP stack route table is the route that was added by OMPROUTE as a result of the information that was received by way of IPv6 OSPF or IPv6 RIP. Any route for the same prefix that is learned from router discovery is ignored as long as the OMPROUTE route exists.

Tip for IPv6 OSPF routing protocol addressing conventions

IPv6 OSPF is based on IPv4 OSPF and has many similar concepts and controls. The primary difference between IPv6 OSPF and IPv4 OSPF is that for IPv6 OSPF, IP addresses are not used to communicate topology information. For example, in IPv4 OSPF, an interface is referred to by its IPv4 home address, but in IPv6 OSPF an interface is not referred to by any of its IPv6 home addresses. Instead, it is referred to by an integer interface ID. Similarly, IPv6 OSPF router IDs are not IPv6 home addresses; they are 32-bit integers written in IPv4-style dotted decimal notation. Area IDs in IPv6 OSPF are also 32-bit integers written in IPv4-style dotted decimal notation.

Guideline: Even though router IDs and area IDs in IPv6 OSPF are expressed similarly to the IPv4 equivalents, they are not the same constants. A router can have an IPv6 router ID which is different from its IPv4 router ID. If both IPv4 and IPv6 OSPF are running simultaneously, the area topology of each IP version can be different, with different area numbers and hierarchy.

Authentication with the IPv6 OSPF routing protocol

IPv4 OSPF includes authentication as part of the OSPF protocol. OMPROUTE supports both password authentication and MD5 cryptographic authentication for IPv4 OSPF. For IPv6 OSPF, authentication has been removed from OSPF itself. Instead, IPv6 OSPF relies on IPSec to ensure integrity and authentication of routing exchanges. As a result, OMPROUTE does not include any explicit authentication support, but instead relies on the underlying support provided by the z/OS TCP/IP stack.

To use IPSec to authenticate IPv6 OSPF routing exchanges on a link over which OMPROUTE establishes adjacencies, you must create a single manual security association (SA) for all traffic on that link, with corresponding filter definitions to permit the OSPF traffic. Use the interface SECCLASS to define different security associations for different links. This procedure is described in z/OS Communications Server: IP Configuration Guide.

Considerations for route selection

Route precedence is as follows:

- Host route to the destination.
- Route for a prefix of the destination. If there are routes to multiple prefixes of the destination, the route with the most specific prefix is chosen.
- Default route.

For IPv4, the concept exists of a special default multicast route with a destination of 224.0.0.0 and a netmask of 255.255.255.255. For IPv6, there is no special default multicast route. Because all IPv6 multicast addresses start with FF, the following prefix route serves the same function as the default multicast route:

```
destination = FF00::/8
```

Considerations for multipath routes

Multiple routes to the same destination are considered multipath routes. Multipath routes can be used for load balancing. Multipath route support for IPv6 is identical to multipath route support for IPv4. Define the MULTIPATH keyword on the IPCONFIG6 statement to control whether multipath routes in the main route table are used for load balancing. Define the Multipath6 parameter on the RouteTable policy statement to control whether multipath routes in a policy-based route table are used for load balancing.

Tips:

- If MULTIPATH is not enabled, the first active route added is selected.

- When a route that belongs to a multipath group is being used, the MTU that is used is the minimum MTU of all routes in the multipath group.

The VARY TCPIP,,OBEYFILE command and routes

When a VARY TCPIP,,OBEYFILE command is issued and the profile contains a BEGINROUTES block, the following results occur:

- All static routes (both replaceable and non-replaceable) are deleted and replaced by any static routes defined in the BEGINROUTES block.
- All routes learned by way of packets that were redirected by ICMPv6 are deleted.
- Routes learned by way of router advertisements or by way of a dynamic routing daemon are not affected by the processing of the VARY TCPIP,,OBEYFILE command, unless the profile data set specified on the VARY TCPIP,,OBEYFILE command contains a non-replaceable static route to the same destination for which a route exists that was learned by way of router advertisements or a dynamic routing daemon. In this case, the existing route is deleted and is replaced by the non-replaceable static route.

Policy-based routing

Policy-based routing enables the TCP/IP stack to make routing decisions that take into account criteria other than just the destination IP address. The additional criteria can include job name, source port, destination port, protocol type (TCP or UDP), source IP address, NetAccess security zone, and multilevel secure environment security label. Policy-based routing might be useful in the following sample scenarios:

- You might want to use high-bandwidth links for batch traffic, but for interactive traffic you prefer low-latency links. In this scenario, you can define policies such that Telnet traffic is routed over the low-latency links, and FTP traffic is routed over the high-bandwidth links.
- You could define a policy to ensure that traffic that is tagged with a security label and zone is routed to a secured network over an appropriate outbound interface.
- You might want to control the links that Enterprise Extender traffic uses to keep that traffic from being impacted by other IP traffic loads.

For more information about policy-based routing, see z/OS Communications Server: IP Configuration Guide.

Restrictions:

- Policy-based routing applies to only TCP and UDP traffic that originates at the TCP/IP stack. Traffic that is using protocols other than TCP and UDP and all traffic that is being forwarded by the TCP/IP stack is always routed by using the main route table, even when policy-based routing is in use.
- If Common INET (CINET) is used to run multiple z/OS Communications Server TCP/IP stacks concurrently, CINET has no knowledge of the policy-based route tables that those TCP/IP stacks use. CINET has knowledge only of the routes in the main route table of each TCP/IP stack. Avoid the use of policy-based routing in a CINET environment, unless at least one of the following conditions is true:
  - All applications establish affinity with a particular TCP/IP stack.
  - The route destinations in each TCP/IP stack route table are mutually exclusive with the route destinations on the other TCP/IP stacks, including the default route.

ICMPv6

The Internet protocol (IP) moves data from one node to another; however, for IP to perform this task successfully, there are other functions that need to be performed: error reporting, router discovery, diagnostics, and others. In IPv6, all these tasks are carried out by the Internet Control Message Protocol (ICMPv6).

In addition, ICMPv6 provides a framework for Multicast Listener Discovery (MLD) and Neighbor Discovery (ND), which carry out the tasks of conveying multicast group membership information (the equivalent of the IGMP protocol in IPv4) and address resolution (performed by ARP in IPv4).

The types of ICMPv6 messages include error messages and informational messages:

Error
Report errors in the forwarding or delivery of IPv6 packets.

Informational
Provide diagnostic functions and additional host functionality such as MLD and ND.

The following ICMPv6 messages are supported:

- Destination unreachable
- Packet too big
- Time exceeded (hop limit exceeded)
- Echo request/reply
- Parameter problem
- Multicast listener discovery:
  - Group membership query
  - Report
  - Done
- Neighbor discovery:
  - Router solicitation and advertisement
  - Neighbor solicitation and advertisement
  - Redirect

Tip: Not all ICMPv4 messages have equivalents in ICMPv6.

Multicast Listener Discovery

In early IP networks, a packet could be sent to either a single device (unicast) or to all devices (broadcast); a single transmission destined for a group of devices was not possible. IPv6 uses multicast for those purposes for which IPv4 used broadcast; consequently, IPv6 does not support broadcast.

Applications can use multicast transmissions to enable efficient communication between groups of devices. Data is transmitted to a single multicast IP address and received by any device that needs to obtain the transmission.

An IPv6 router uses Multicast Listener Discovery (MLD) protocol to discover the following information:

- The presence of multicast listeners (nodes wanting to receive multicast packets) on its directly attached links
- Which multicast addresses are of interest to those listeners

MLD provides this information to the multicast routing protocol the router is using. This ensures that multicast packets are delivered to all links where there are interested receivers. MLD is derived from IGMPv2.

Guideline: One important difference is that MLD uses ICMPv6 message types, rather than IGMP message types.

MLD has a router function and a listener function. The router function discovers the presence of multicast listeners and ensures delivery of multicast packets to listeners. The listener function informs routers when it starts listening for a multicast address, when it stops listening for a multicast address, and when it responds to queries about multicast addresses. z/OS Communications Server implements the listener function.

When a listener starts listening for a multicast address on an interface, it sends an MLD report message for that address on that interface.

When a listener stops listening for a multicast address on an interface, it sends a single MLD done message.

A router sends an MLD query message to query listeners about multicast addresses. A specific query is sent to listeners for a specific multicast address on a receiving interface. A general query is sent to listeners for all multicast addresses on a receiving interface. These query messages contain a maximum response delay (MRD). The MRD causes listeners to delay report messages and not send them if another listener reports first. If no reports for the address are received from the link after the response delay of the last query has passed, the routers on the link assume that the address no longer has any listeners there; the address is therefore deleted from the list and its disappearance is made known to the multicast routing component.

If you configure IP security for IPv6, see z/OS Communications Server: IP Configuration Guide for information about filter rules for MLD packets.

Neighbor discovery

Neighbor discovery (ND) is an ICMPv6 function that enables a node to identify other hosts and routers on its links. It corresponds to a combination of IPv4 protocols:

- ARP
- ICMP Router Discovery
- ICMP Redirect

It maintains routes, MTU, retransmit times, reachability time, and prefix information based on information received from the routers. ND uses duplicate address detection (DAD) to verify the host's home addresses are unique on the LAN.

ND uses address resolution to determine the link-layer addresses for neighbors on the LAN. ND uses reachability detection to determine neighbor reachability.

If you configure IP security for IPv6, see z/OS Communications Server: IP Configuration Guide for information about filter rules for neighbor discovery packets.

Router advertisements

Router advertisements are sent by routers to announce their availability. z/OS Communications Server receives router advertisements, but it does not originate them. The router advertisement includes information that is used by z/OS Communications Server, including an indication of whether the sending router should be used as the default router.

Sending router should be a default router

If the router advertisement indicates that the sending router should be used as a default router, z/OS Communications Server takes the following actions:

- If the dynamic default route that is to be added as the result of the received router advertisement has already been added by a previous advertisement, the length of time that that route remains valid is reset using the Lifetime value specified on the received advertisement. If no default route exists, a dynamic default route is added as a result of the received router advertisement.
- If a default route exists that has equal or lower precedence than the route that is to be added, a dynamic route is added as a result of the received router advertisement. If a route with lower precedence exists, it is removed but is reinstated later if the dynamic default route that is added is removed. The following types of routes have equal or lower precedence:
  – A router advertisement route that has a reachable gateway, an active interface, and the same preference value as the default router preference value that was received in the advertisement; this type of route has equal precedence

  - A router advertisement route that has an unreachable gateway, an inactive interface, or a lower preference value than the default router preference value that was received in the advertisement; this type of route has lower precedence
  - A replaceable static route; this type of route has lower precedence

  The dynamic default route that is added has the following characteristics:

  - The next-hop address is the source address of the advertisement.
  - The interface is the interface on which the advertisement was received.
  - The metric is set according to the preference value that was received in the advertisement. The setting 1 indicates high preference, 2 indicates medium preference, and 3 indicates low preference.
  - The length of time that the route remains valid is equal to the Lifetime value set on the advertisement.
- If a default route exists that has a higher precedence than the route that is to be added, a dynamic default route is not added as the result of the received router advertisement. A dynamic default route is added later if the route with the higher precedence is removed. The following types of routes have higher precedence:
  - A router advertisement route that has a reachable gateway, an active interface, and a higher preference value than the default router preference value that was received in the advertisement
  - A non-replaceable static route
  - An IPv6 OSPF route
  - An IPv6 RIP route
- A neighbor cache entry is created or updated for the sending router. The neighbor cache entry contains the following information obtained from the router advertisement:
  - An indication that the neighbor is a router
  - An indication that the neighbor is a default router
  - The link-local and link-layer addresses of the neighbor

Sending router should not be a default router

If the router advertisement indicates that the sending router should not be used as a default router, z/OS Communications Server takes the following actions:

- If an IPv6 dynamic default route exists that has the advertisement's source as its next hop and the receiving interface as its interface, and that route was added as the result of a received router advertisement (but not, for example, as the result of IPv6 OSPF or IPv6 RIP), that route is deleted.
- A neighbor cache entry is created or updated for the sending router. The neighbor cache entry contains the following information obtained from the router advertisement:
  - An indication that the neighbor is a router
  - An indication that the neighbor is not a default router
  - The link-local and link-layer addresses of the neighbor

Route information option for router advertisements

A router advertisement can contain route information options. Each route information option contains an IPv6 prefix and information that indicates whether the prefix can be reached by way of the router that originated the router advertisement.

If the option contains a nonzero Route Lifetime value, which indicates that the prefix can be reached by way of the router, the following actions occur:

- If the dynamic prefix route that is to be added as the result of the received router advertisement has already been added by a previous advertisement, the length of time that that route remains valid is reset using the Route Lifetime value from the route information option.

- If no route for the prefix exists, or if a route exists that has equal or lower precedence than the route that is to be added, then a dynamic prefix route is added as the result of the received router advertisement. If a route with lower precedence exists, it is removed but is reinstated later if the dynamic prefix route that is added is removed. The following types of routes have equal or lower precedence:
  - A router advertisement route that has a reachable gateway, an active interface, and the same preference value as the preference value that was received in the route information option; this type of route has equal precedence.
  - A router advertisement route that has an unreachable gateway, an inactive interface, or a lower preference value than the preference value that was received in the route information option; this type of route has lower precedence
  - A replaceable static route; this type of route has lower precedence

  The dynamic prefix route that is added has the following characteristics:
  - The next-hop address is the source address of the advertisement.
  - The interface is the interface on which the advertisement was received
  - The metric is set according to the preference value that was received in the Route Information option. The setting 1 indicates high preference, 2 indicates medium preference, and 3 indicates low preference.
  - The length of time that the route remains valid is equal to the Route Lifetime value set on the option.
- If a route for the prefix exists that has a higher precedence than the route that is to be added, a dynamic prefix route is not added as the result of the received router advertisement. A dynamic prefix route is added later if the route with the higher precedence is removed. The following types of routes have higher precedence:
  - A router advertisement route that has a reachable gateway, an active interface, and a higher preference value than the preference value that was received in the route information option
  - A non-replaceable static route
  - An IPv6 OSPF route
  - An IPv6 RIP route

If the option contains the value 0 for the Route Lifetime value, which indicates that the prefix can no longer be reached by way of the router, the following action occurs:

- If an IPv6 dynamic prefix route exists that has the source of the advertisement as its next hop and the receiving interface as its interface, and that route was added as the result of a received router advertisement (but not, for example, as the result of IPv6 OSPF or IPv6 RIP), that route is deleted.

Prefix information option for router advertisements
A router advertisement can contain prefix information options. Each prefix information option contains an IPv6 prefix and flags that indicate how the prefix can be used.

A prefix information option contains two flags:

- An on-link flag, which indicates whether on-link processing needs to be performed for the prefix on the shared link. When a prefix is on-link, the addresses in that prefix can be reached on that link without going through a router.
- An autonomous flag, which indicates whether autoconfiguration processing needs to be performed for the prefix on the shared link.

The sending router can set one flag or both flags in the prefix information option.

On-link processing

The sending router indicates that a prefix is on-link by setting the on-link flag and specifying a nonzero value for the Valid Lifetime value for the prefix. If the prefix information option indicates that the prefix is on-link, the following criteria are true:

- z/OS Communications Server adds an IPv6 dynamic direct route (if it was not already added by a previous advertisement).
- The destination of the route is the prefix that is being processed.
- The interface of the route is the interface on which the advertisement was received.
- The length of time that the route remains valid is set or is reset using the Valid Lifetime value from the Prefix Information option.

If a non-replaceable static route exists to this prefix or if a direct route to the prefix was added by OMPROUTE (because the PREFIX parameter was specified on the IPV6_INTERFACE, IPV6_OSPF_INTERFACE, or IPV6_RIP_INTERFACE statement in the OMPROUTE configuration file or because a router advertised by way of IPv6 OSPF that the prefix is on-link), then z/OS Communications Server does not add the dynamic direct route. If a replaceable static route exists to this prefix, the dynamic direct route is added, which replaces the replaceable route. The replaceable static route is reinstated if the dynamic direct route is removed later.

The sending router can indicate that a prefix is no longer on-link by setting the on-link flag and specifying the value 0 for the Valid Lifetime value for the prefix. If an IPv6 dynamic direct route exists for which the destination is the prefix that is being processed and for which the interface is the receiving interface, and that route was added as the result of a received router advertisement (for example, the route was not added by OMPROUTE), then z/OS Communications Server deletes the route.

Address autoconfiguration processing

The sending router can indicate that a prefix is to be used for address autoconfiguration by setting the autonomous flag and specifying a nonzero Valid Lifetime value for the prefix. If the Prefix Information option indicates that the prefix should be used for address autoconfiguration, z/OS Communications Server performs the following actions:

- Adds an IPv6 home address to the receiving interface for the public autoconfigured address (if that home address was not added by a previous advertisement)
- Adds an IPv6 implicit route for the receiving interface and the public autoconfigured address (if that route was not added by a previous advertisement)
- Sets or resets the length of time that the home address and implicit route remain valid, using the Valid Lifetime value from the Prefix Information option
- Sets or resets the length of time that the home address remains in the preferred state (not in the deprecated state), using the Preferred Lifetime value from the Prefix Information option

If you configured this interface to support temporary addresses (you configured the TEMPADDRS parameter on the IPCONFIG6 statement and the TEMPPREFIX parameter that is specified on the INTERFACE statement includes the prefix), z/OS Communications Server also performs the following actions:

- Adds an IPv6 home address for the receiving interface and for the temporary autoconfigured address (if that home address was not already added by a previous advertisement)
- Adds an IPv6 implicit route for the receiving interface and for the temporary autoconfigured address (if that route was not already added by a previous advertisement)
- Sets or resets the length of time that the home address and implicit route remain valid, using the Valid Lifetime value from the Prefix Information option and the TEMPADDRS VALIDLIFETIME value that is configured
- Sets or resets the length of time that the home address remains in the preferred state (not in the deprecated state), using the Preferred Lifetime value from the Prefix Information option and the TEMPADDRS PREFLIFETIME value that is configured

Restriction: Prefixes that are learned solely by using the Prefix parameter on the OMPROUTE IPV6_INTERFACE, IPV6_OSPF_INTERFACE, or IPV6_RIP_INTERFACE statement are never used for autoconfiguration.

If you manually configure addresses for an IPv6 interface using the INTERFACE statement, addresses for that interface cannot be autoconfigured. If a prefix is not 64 bits in length, it is not used for autoconfiguration of addresses. Unlike the prefix route and the default route, the implicit route and home address cannot immediately be deleted; these items must age out. If the Valid Lifetime value is set to infinity, the implicit route and home address for the public autoconfigured address do not time out. For more information about autoconfiguration, see "Stateless address autoconfiguration" on page 30.

Route timeouts

The valid lifetime for each type of route is updated (extending the life of the route) by the periodic receipt of router advertisements as long as the sending router is available and is not reconfigured relative to its defined prefixes or default router status.

When a Prefix Information option contains the Valid Lifetime value `infinity`, the implicit or prefix route associated with the public autoconfigured address is considered permanent and does not age unless a future Prefix Information option for the prefix contains a Valid Lifetime value that is not `infinity`.

Expiration of the valid lifetime for a default route is immediate if a future router advertisement indicates that the sending router is no longer a default router. Expiration of the valid lifetime for a prefix route is immediate if a future Prefix Information Option for the prefix contains the Valid Lifetime value 0 or if a future Route Information Option for the prefix contains the Route Lifetime value 0. The valid lifetime for an implicit route cannot expire immediately because the minimum lifetime allowed is 2 hours; the lifetime must age out naturally.

VARY TCPIP,,OBEYFILE command rules

Rules: Observe the following rules for the VARY TCPIP,,OBEYFILE command:

- If a non-replaceable static route in the profile data set specified on the VARY TCPIP,,OBEYFILE command has the same destination as an existing route that was added because of a received Router Advertisement, the existing route is replaced by the non-replaceable static route.
- If the profile data set specified on the VARY TCPIP,,OBEYFILE command specifies a manually configured home address for an interface that already has autoconfigured addresses, the autoconfigured addresses are deleted along with their associated implicit routes.

With the exception of these two rules, all autoconfigured home addresses and routes added because of received Router Advertisements are maintained through VARY TCPIP,,OBEYFILE command processing.

Redirect processing

A node can receive a Redirect message from an on-link router if the router determines that the destination is on-link or if there is a better first-hop router for the given destination. z/OS Communications Server can be configured to ignore the IPv6 Redirects sent by routers by defining the IGNOREREDIRECT keyword on the IPCONFIG6 statement. In addition, IPv6 Redirects are ignored if the IPv6 OSPF or IPv6 RIP protocol of the OMPROUTE routing daemon is being used. If processing of Redirect messages is enabled, z/OS Communications Server begins using the new first-hop information which is identified in the Redirect message. A router must use its link-local address as the source address in Redirects that it originates. A received Redirect is processed only if the current route to the destination in the IPv6 route table has the source address of the Redirect as its next hop. Therefore, if Redirects are to be accepted, all static indirect routes must be configured using the next-hop router's link-local address. If the previous route to the destination was a host route, it is deleted from the route table to keep it from being used by Multipath processing.

If Redirect processing is disabled, z/OS Communications Server silently discards the Redirect message.

Duplicate address detection

Duplicate address detection (DAD) is used to verify that an IPv6 home address is unique on the LAN before the address is assigned to a physical interface (for example, QDIO). z/OS Communications Server responds to other nodes that are doing DAD for IP addresses assigned to the interface. DAD is not done for VIPAs or loopback addresses. DAD for local addresses is done for physical interfaces when one of the following situations occur:

- The interface is started (the autoconfigured link-local address and manually configured addresses and / prefixes are checked).
- A VARY TCPIP,,OBEYFILE command is issued for a profile data set containing an INTERFACE ADDADDR for an already active interface.
- A router advertisement containing new prefix information and the autonomous bit set is received on an interface enabled for stateless autoconfiguration.
- A temporary autoconfigured address is generated.

To disable DAD checking, specify DUPADDRDET 0 on the INTERFACE statement.

DAD processing involves the following steps:

1. The host joins a link-local all-nodes multicast group at interface start processing.
2. The host joins a solicited-node group for the local address.
3. A neighbor solicitation is sent to the solicited-node multicast address with the tentative address for which DAD is being performed.
4. The host waits for a neighbor response (neighbor advertisement or neighbor solicitation) on the interface.
5. If no neighbor response is received within the specified retransmit time, the address is considered unique on the LAN.
6. If a neighbor response is received within the specified time, the address is not unique. The host leaves the solicited-node multicast group, issues a duplicated address detected console message, and marks the address unavailable because of a duplicate address.

Unless DAD is disabled, the address is not considered assigned to an interface until DAD is successfully completed for the local address. Packets can be received for the all-nodes or solicited-node multicast groups, but there is no response because the address is not yet assigned to the interface. If the local address is a manually configured address, the addresses are displayed in a Netstat Home/-h report as `Unavailable` (if the interface has not been started or if DAD failed).

In situations where DAD is not done for the IPv6 home address (by specifying DUPADDRDET 0 on the INTERFACE statement or if it is a VIPA), the z/OS Communications Server host still responds if another node is doing DAD for an IPv6 address assigned to the interface or for IPv6 VIPAs when the interface is assigned to handle VIPAs; responses are not sent for loopback addresses.

Address resolution

Address resolution in IPv6 is similar to ARP processing in IPv4, except ICMP neighbor solicitations, neighbor advertisements, router redirects, and router advertisements are used to obtain the link-layer (MAC) address. The host sends a neighbor solicitation to a solicited-node multicast address. It waits for a response for a period of time (retransmit time). If one is received, then the link-layer address contained in the neighbor advertisement is cached and any queued packets are sent to the address. If there is no response, the host repeats this process up to three times before it declares a neighbor unreachable.

A neighbor cache entry can also be built when a neighbor solicitation for a local address is received and the solicitation contains the sender's link-layer address (and the source address is not the unspecified address, that is, the sender is not performing DAD). The neighbor cache entry is built if it does not exist based on the assumption that a packet is soon sent to this neighbor. Building the cache entry reduces the overhead of having to perform the task of address resolution for the neighbor at a later time.

Issue the Netstat ND/-n command to display information for a specific neighbor or all neighbor cache entries. It displays the neighbor link-layer address, state, whether the neighbor is a router or host, and whether a router is a default router. The following states are possible neighbor states:

Incomplete
Address resolution is in progress.

Reachable
Positive confirmation of reachability was received.

Stale
An unsolicited neighbor discovery message has updated the link-layer address. Reachability is verified the next time the entry is used.

Delay
More than reachable time has elapsed since last positive confirmation of reachability. Default reachable time is 30 seconds. It can be overridden by data provided by neighbor advertisements. A small delay is experienced before starting a probe of neighbor (upper layers can provide confirmation).

Probe
Neighbor solicitations are sent to verify neighbor reachability.

Neighbor unreachability detection

Neighbor unreachability detection verifies that two-way communication with a neighbor node exists. The host sends a neighbor solicitation to a node and waits for a solicited neighbor advertisement. If a solicited neighbor advertisement is received, the node is considered reachable. If there is no response, the host can repeat this process before it declares a neighbor unreachable. If a neighbor is found to be unreachable, the neighbor cache entry is deleted.

Assigning IP addresses to interfaces

Stateless address autoconfiguration is always used to generate and assign a link-local address to a physical IPv6 interface. If it cannot assign a link-local address, interface activation fails. No other addresses are assigned to the interface (whether they are assigned using stateless address autoconfiguration or manual configuration) until a link-local address has been successfully assigned. Link-local addresses are not aged out.

Stateless address autoconfiguration

The larger address field of IPv6 solves a number of problems inherent in IPv4, but the size of the address itself might be a potential problem for the TCP/IP administrator. As a result, IPv6 has the capability to automatically assign an address to an interface at initialization time. In this way, a network can become operational with minimal action on the part of the TCP/IP administrator. Stateless autoconfiguration is supported for an OSA-Express QDIO interface in z/OS Communications Server if no manually configured addresses are defined on the interface. Manual configuration of the host's local addresses is not required except for VIPA interfaces. Stateless address autoconfiguration consists of the following steps:

1. During interface startup, the host generates its own addresses by using a combination of router advertised prefixes and the interface ID. If the INTFID parameter is configured on the INTERFACE statement, the value that is configured on the parameter is used as the interface ID. Otherwise, the host obtains an interface token from the interface hardware to create an interface ID.
2. If temporary addresses are supported on the interface (the TEMPADDRS parameter is configured on the IPCONFIG6 statement and the TEMPPREFIX parameter is configured on the INTERFACE statement), a random interface ID is generated. Temporary addresses are generated using a combination of router-advertised prefixes and the random interface ID.
3. Duplicate address detection is performed for each address. If a duplicate is not detected or Duplicate Address Detection (DAD) is disabled for the interface (DUPADDRDET 0 specified on the INTERFACE statement), the local address is added.
4. A stateless autoconfigured address is deleted when its valid lifetime expires or when a manually defined address is added to the interface.

   An IPv6 address generated using stateless address autoconfiguration has two timers associated with it: A preferred lifetime timer and a valid lifetime timer. Router advertisements contain the valid lifetime and preferred lifetime timers for a prefix. Temporary autoconfigured addresses also have a valid lifetime and preferred lifetime timer configured on the IPCONFIG6 statement (TEMPADDRS PREFLIFETIME *value* VALIDLIFETIME *value*). The valid and preferred lifetime timers for a temporary autoconfigured address are the lesser of the values contained in the router advertisement for the prefix and the value specified on the IPCONFIG6 statement. The valid and preferred lifetime timers for a public autoconfigured address are the values that are in the router advertisement for the prefix.

An IPv6 address goes through two phases to gracefully handle the address expiration:

Preferred
Use is unrestricted.

Deprecated
In anticipation of the expiration of the leased period, use of the address is discouraged.

When the preferred lifetime expires, the address created from the prefix is deprecated. When the valid lifetime expires, the address created from the prefix is deleted and an operator message is issued.

Autoconfiguration considerations

During autoconfiguration, make the following considerations:

- A manually configured address or prefix on an interface disables stateless autoconfiguration for the interface.
- INTERFACE *name* DELADDR *addr/prefix* and INTERFACE *name* DEPRADDR *addr/prefix* profile statements that are activated with the VARY TCPIP,,OBEYFILE command are not valid for autoconfigured addresses.
- A VARY TCPIP,,OBEYFILE command whose profile contains ADDADDR INTERFACE or DELADDR INTERFACE statements can affect stateless autoconfiguration:
  - An INTERFACE *name* ADDADDR *addr/prefix* profile statement that is activated with the VARY TCPIP,,OBEYFILE command results in the deletion of stateless autoconfigured addresses on the interface. Stateless autoconfiguration capability is disabled.
  - If the DELADDR profile statement removes the last manually configured address or prefix, stateless autoconfiguration is enabled and subsequent router advertisements can generate autoconfigured addresses.

Guidelines:

- Consider using VIPA addresses in conjunction with autoconfigured addresses because public autoconfigured addresses are not automatically added to the Domain Name System (DNS).
- Do not add temporary autoconfigured address to the DNS. Temporary autoconfigured addresses are regenerated periodically to prevent client activity from being correlated. If a DNS name is associated with the addresses, the DNS name might be used for correlation.

IP address takeover following an interface failure

The TCP/IP stack in z/OS Communications Server provides transparent fault-tolerance for failed (or stopped) IPv6 interfaces, when the stack is configured with redundant connectivity onto a LAN. This support is provided by the z/OS Communications Server interface-takeover function and applies to the IPv6 IPAQENET6 interface type.

At device or interface startup time, TCP/IP dynamically learns of redundant connectivity onto the LAN, and uses this information to select suitable backups in the case of a future failure of the device/interface. This support makes use of neighbor discovery flows for IPv6 interfaces, so upon failure (or stop) of an interface, TCP/IP immediately notifies stations on the LAN that the original IPv6 address is now reachable by way of the backup's link-layer (MAC) address. Users targeting the original IP address see no outage because of the failure, and they are unaware that any failure occurred.

Because this support is built upon neighbor discovery flows, no dynamic routing protocol in the IP layer is required to achieve this fault tolerance. To enable this support, you must configure redundancy onto the LAN by defining and activating multiple INTERFACEs onto the LAN. Note that an IPv4 device cannot back up an IPv6 interface, and an IPv6 interface cannot back up an IPv4 device.

The interface-layer fault-tolerance can be used in conjunction with VIPA addresses, where applications can target the VIPA address, and any failure of the real LAN hardware is handled by the interface-takeover function. This differs from traditional VIPA usage, where dynamic routing protocols are required to route around true hardware failures.

How to get addresses for VIPAs

VIPA interfaces are always active. IPv6 VIPA interfaces can have only global addresses that are assigned to them. IPv6 VIPA interfaces are not allowed to have link-local addresses because link-local addresses can be used only on the associated LAN and no VIPA LAN is available.

Rule: You must manually configure all addresses that are assigned to IPv6 VIPAs.

To globally enable SOURCEVIPA for IPv6, configure the SOURCEVIPA keyword on the IPCONFIG6 statement. Then, to enable SOURCEVIPA for particular interfaces, use the SOURCEVIPAINTERFACE parameter on the INTERFACE statement for those interfaces. The SOURCEVIPAINTERFACE parameter allows for the specification of the interface name of the VIRTUAL6 interface whose addresses should be used as SOURCEVIPA addresses.

Unlike IPv4, where the source VIPA selected is based upon the ordering of the HOME list, IPv6 SOURCEVIPA uses the addresses configured on the VIPA INTERFACE statement referenced by the SOURCEVIPAINTERFACE keyword on the INTERFACE statement for the outbound interface. When that VIPA interface has multiple addresses configured, the default source address selection algorithm selects among them. For detailed information about the algorithm, see "Default source address selection" on page 37.

Guidelines:

- Use different prefixes for IPv6 static VIPAs and for the IPv6 addresses assigned to real interfaces.
- Configure static VIPAs with different prefixes than real addresses. Configuring static VIPAs in this way reduces the likelihood of address collisions between the manually configured VIPAs and the autoconfigured addresses of the real interfaces. This kind of configuration is also necessary because duplicate address detection (DAD) is not performed for VIPA addresses.

See "Assigning IPv6 addresses" on page 62 for information about static VIPAs.

IPv6 temporary addresses with random interface IDs

RFC 4941 addresses a potential security concern that can occur when you are using stateless address autoconfiguration. You can use IPv6 temporary addresses with random interface IDs to mitigate this security issue.

An autoconfigured address contains an embedded static interface identifier. The static interface ID makes it possible to correlate independent transactions to and from the system using the adapter, even if the overall IPv6 address changes.

RFC 4941, *Privacy Extensions for Stateless Address Autoconfiguration in IPv6*, defines a mechanism to generate a random interface ID that changes over time. Temporary autoconfigured addresses are then generated from the random interface ID. A short-lived client application can use temporary addresses with changing embedded interface IDs to make it more difficult to correlate activity.

A history value is used as part of the algorithm that generates the random interface ID. The first time that an interface is started, a random number generator generates the history value. If cryptographic hardware is available, then the Integrated Cryptographic Service Facility (ICSF) callable service CSNBRNG is used to generate the history value. If cryptographic hardware is not available, then a software random number generator generates the history value. Message number EZD0043I indicates the source of the history value. See z/OS Cryptographic Services ICSF Application Programmer's Guide for more information about the CSNBRNG callable service.

Configuring a TCP/IP stack to generate IPv6 temporary addresses

To implement the mechanism defined in RFC 4941 regarding the use of randomly generated interface IDs, you must first configure a TCP/IP stack to generate IPv6 temporary addresses.

Before you begin

Before you configure a TCP/IP stack to use IPv6 temporary addresses:

- Understand IPv6 stateless address autoconfiguration. See "Stateless address autoconfiguration" on page 30 for a description of autoconfigured addresses, both public and temporary.
- Determine whether you have a client application that would benefit from using temporary autoconfigured addresses. Temporary addresses are designed to be used with short-lived client connections.
- Determine whether stateless address autoconfiguration is being used for one or more of the OSA-Express IPAQENET6 interfaces that are defined in the TCP/IP profile. Temporary autoconfigured addresses can be generated only for an OSA-Express IPAQENET6 interface that is using autoconfiguration (the IPADDR parameter is not specified with the IP address or prefix on the INTERFACE statement).

Procedure

Perform the following steps to configure a TCP/IP stack to generate IPv6 temporary addresses:

1. Enable the generation of temporary addresses by configuring the TEMPADDRS parameter on the IPCONFIG6 statement.

   For more information about the TEMPADDRS parameter, see the IPCONFIG6 statement in z/OS Communications Server: IP Configuration Reference.
2. (Optional) Set the preferred lifetime and the valid lifetime for temporary addresses by configuring the parameters PREFLIFETIME *preflifetime* VALIDLIFETIME *validlifetime* on the IPCONFIG6 statement.

   Default values are used if you do not configure these parameters. The preferred lifetime and valid lifetime values apply to all temporary addresses on the TCP/IP stack. For more information about preferred and valid lifetimes see "Stateless address autoconfiguration" on page 30. For more information about the PREFLIFETIME and VALIDLIFETIME parameters, see the information about the IPCONFIG6 statement in z/OS Communications Server: IP Configuration Reference.
3. (Optional) Limit the IPv6 prefixes for which temporary addresses can be generated by configuring the TEMPPREFIX parameter on one or more INTERFACE statements.

   In most cases, you can use the default value TEMPPREFIX ALL, which enables temporary addresses to be generated for all prefixes that are learned from router advertisements over the interface. If you need to limit the prefixes for which temporary addresses are generated for an interface, you can specify the TEMPPREFIX parameter on the INTERFACE statement. For more information about the TEMPPREFIX parameter, see the information about the IPAQENET6 INTERFACE statement in z/OS Communications Server: IP Configuration Reference.

   Guideline: If SOURCEVIPA is enabled and the SOURCEVIPAINT parameter is configured for an interface, the default source address selection algorithm selects an address from the addresses for the source VIPA interface, not from the addresses for the outbound interface. Specify TEMPPREFIX NONE to disable unnecessary generation of temporary addresses for the outbound interface. For more information, see "VIPA considerations with source address selection" on page 39.

What to do next

When you are done, configure the client application to use temporary addresses. See "Enabling a client application to use IPv6 temporary or public addresses" on page 34.

Enabling a client application to use IPv6 temporary or public addresses

After you have configured a TCP/IP stack to generate IPv6 temporary addresses, you must enable the client application to use these addresses or IPv6 public addresses.

Before you begin

You need to have configured the TCP/IP stack to generate temporary IPv6 addresses. See "Configuring a TCP/IP stack to generate IPv6 temporary addresses" on page 33.

Procedure

Perform the following steps to enable a client application to use IPv6 temporary or public addresses:

1. Identify the job name of the client application for which temporary or public addresses will be used.
2. Specify that temporary IPv6 addresses are preferred for an application:
   - Specify a JOBNAME *jobname* TEMPADDRS entry on the SRCIP statement.
   - Use the socket API extensions to specify source IP address preferences at the socket level.

   For more information about the SRCIP statement, see z/OS Communications Server: IP Configuration Reference. This information includes a description of how the job name is determined for an application.
3. Specify that public IPv6 addresses are preferred for an application:
   - Specify a JOBNAME *jobname* PUBLICADDRS entry on the SRCIP statement.
   - Use the socket API extensions to specify source IP address preferences at the socket level.

   For more information about the SRCIP statement, see z/OS Communications Server: IP Configuration Reference. The information includes a description of how the job name is determined for an application.

What to do next

When you are done, you can display the configured and generated temporary address information. See "Displaying the configured and generated temporary or public address information" on page 34.

Displaying the configured and generated temporary or public address information

After you have configured the TCP/IP stack and the client application, you can display the temporary or public address information.

Before you begin

You must do the following configuration:

- Configure the TCP/IP stack to generate IPv6 temporary addresses.
- Configure the client application to use IPv6 temporary or public addresses.

Procedure

Perform the following steps to display the configured and generated temporary or public address information:

1. Issue the Netstat CONFIG/-f command to display the TempAddresses setting and the PreferredLifetime and ValidLifetime values.

   For a description of these fields, see the Netstat CONFIG/-f report example in z/OS Communications Server: IP System Administrator's Commands.
2. Issue the Netstat DEvlinks/-d command to display the TempPrefix values.

   For a description of this field see the Netstat DEvlinks/-d report example in z/OS Communications Server: IP System Administrator's Commands.
3. Issue the Netstat HOme/-h command to display any generated temporary addresses.

The `Flags` field in the display indicates `Temporary` for a temporary address. The `ValidLifetimeExp` field in the display indicates when the temporary address will be deleted. For a description of this report, see the Netstat HOme/-h report example in z/OS Communications Server: IP System Administrator's Commands.

4. Issue the Netstat SRCIP/-J command to display entries in the SRCIP statement block.

   A `Job Name` entry indicates `TEMPADDRS` for the `Source` field if a temporary address is to be preferred for the client's source IP address. A `Job Name` entry indicates `PUBLICADDRS` for the `Source` field if a public address is to be preferred for the client's source IP address.

Default address selection

IPv6 addressing architecture allows multiple unicast addresses to be assigned to interfaces. These addresses might have different reachability scopes (link-local or global). These addresses can also be preferred or deprecated. Privacy considerations have introduced the concepts of public addresses and temporary addresses. The mobility architecture introduces home addresses and care-of addresses. In addition, multihoming situations result in more addresses per node. For example, a node can have multiple interfaces, some of them tunnels or virtual interfaces, or a site can have multiple ISP attachments with a global prefix per ISP.

The end result is that IPv6 implementations are often faced with multiple possible source and destination addresses when initiating communication. It is preferred to have default algorithms, common across all implementations, for selecting source and destination addresses so that developers and administrators can reason about and predict the behavior of their systems.

Furthermore, dual-mode stack implementations, which support both IPv6 and IPv4, very often need to choose between IPv6 and IPv4 when initiating communication. For example, DNS name resolution might yield both IPv6 and IPv4 addresses with the network protocol stack having both IPv6 and IPv4 source addresses available. In these cases, a policy that always prefers IPv6 or always prefers IPv4 might produce poor results. For example, if a DNS name resolves to a global IPv6 address and a global IPv4 address. If the node has assigned a global IPv6 address and a 169.254/16 autoconfigured IPv4 address, then IPv6 is the best choice for communication because the global address has a similar scope; therefore, a better chance of success. But if the node has assigned only a link-local IPv6 address and a global IPv4 address, then IPv4 is the best choice for communication because the scope more closely matches the scope of the destination to which you are communicating. The destination address selection algorithm solves this with a unified procedure for choosing among both IPv6 and IPv4 addresses.

Source address selection and destination address selection are discussed separately, but using a common framework enables the two algorithms together to yield useful results. The algorithms attempt to choose source and destination addresses of appropriate scope and configuration status (preferred or deprecated).

Policy table for IPv6 default address selection

The policy table for IPv6 default address selection is a longest-matching prefix lookup table, much like a routing table. You can configure this table to suit your environment.

Given an address, a lookup in the policy table produces two values: a precedence value for the address and a label for the address. In the table, IPv4 addresses are represented as IPv4-mapped IPv6 addresses. The default policy table for IPv6 default address selection contains the following values.

Table 5. Default policy table for IPv6 default address selection

Prefix	Precedence	Label
::1/128	50	0
::/0	40	1
2002::/16	30	2
::/96	20	3

Table 5. Default policy table for IPv6 default address selection (continued)

Prefix	Precedence	Label
::ffff:0.0.0.0/96	10	4

In the table, the prefix values specify the address prefix that is used to select the policy table entry that best matches a source or destination address; the precedence values specify how destination addresses are sorted; and the label values specify whether a given source address prefix is preferred for use with a given destination address prefix.

This default configuration produces the following results:

- Native source addresses are preferred for use with native destination addresses
- 6to4 source addresses are preferred for use with 6to4 destination addresses
- IPv4-compatible IPv6 source addresses are preferred for use with IPv4-compatible IPv6 destination addresses

  Guideline: IPv4-compatible IPv6 addresses are deprecated by RFC 4291, but are shown here because they are part of the default policy table that RFC 3484 defines.
- Communication using IPv6 addresses is preferred to communication using IPv4 addresses, if matching source addresses are available

You can use the DEFADDRTABLE TCP/IP profile statement to configure the policy table for IPv6 default address selection to better suit your environment. For example, you can specify that IPv4 addresses should be preferred over IPv6 addresses.

Default destination address selection

Resolver APIs can return multiple IP addresses as a result of a host name query; however, many applications use only the first address returned to attempt a connection or to send a UDP datagram. Therefore, sorting of these IP addresses is performed by the default destination address selection algorithm.

Establishing connectivity can depend on whether an IPv6 address or an IPv4 address is selected, which makes this sorting function even more important.

Default destination address selection occurs only when the system is enabled for IPv6 and the application is using the getaddrinfo() API to retrieve IPv6 addresses, IPv4 addresses, or both.

The default destination address selection algorithm sorts a list of destination addresses and generates a new list. The algorithm sorts together both IPv6 and IPv4 addresses by a set of rules. Rules are applied, in order, to the first and second address, choosing a best address. Rules are then applied to this best address and the third address. This continues until rules have been applied to all addresses and the entire list of addresses has been sorted. If one of the rules is able to select the best address between two addresses, remaining rules are bypassed for those two addresses. Subsequent rules act as tie-breakers for earlier rules.

The destination address selection algorithm attempts to predict what source address is selected by TCP/IP when the application initiates an outbound connection or sends a datagram by using the destination address. This source address is used for some of the destination address selection criteria rules. Source address prediction processing assumes that the application does not explicitly specify a source IP address (by using bind or ipv6_pktinfo) when it is initiating a connection or sending a datagram. If the application explicitly specifies a source address, then the destination address that this algorithm selects might not be optimal. The decision the algorithm makes might assume that a different source address is used.

Rules:

1. Avoid unusable destinations.

   If one address is reachable (the stack has a route to the particular address) and the other is unreachable, then place the reachable destination address before the unreachable address.

2. Prefer matching scope.

   If the scope of one address matches the scope of its source address and the other address does not meet this criteria, then the address with the matching scope is placed before the other destination address.

   The scopes of the destination addresses and their associated source addresses are determined by the high-order bits of the address. The destination address can be a multicast or unicast address. To compare scope, unicast link-local addresses are mapped to multicast link-local addresses and unicast global addresses are mapped to multicast global addresses.
3. Avoid deprecated addresses.

   If one address is deprecated and the other is non-deprecated, then the non-deprecated address is placed before the other address.
4. Prefer matching label.

   If the label of one destination address matches the label of its associated source address and the label of the other destination address does not match the label of its associated source address, then the destination with the matching label is placed before the other address.

   See "Policy table for IPv6 default address selection" on page 35 and "Configuring the policy table for default address selection" on page 39 for information about how labels are associated with destination addresses.
5. Prefer higher precedence.

   If the precedence of one address is higher than the precedence of the other address, then the address with the higher precedence is placed before the other destination address.

   See "Policy table for IPv6 default address selection" on page 35 and "Configuring the policy table for default address selection" on page 39 for more information about how precedence values are associated with destination addresses.
6. Prefer smaller scope.

   If the scope of one address is smaller than the scope of the other address, the address with the smaller scope is placed before the other destination address.
7. Use the longest matching prefix.

   If one destination address has a longer CommonPrefixLength with its associated source address than the other destination address has with its source address, then the address with the longer CommonPrefixLength is placed before the other address.
8. Leave the order unchanged.

   No rule selected a better address of these two addresses; they are equally good. Choose the first address as the better address of these two and the order is not changed.

Default source address selection

When the application or upper-layer protocol has not selected a source address for an outbound IPv6 packet (using bind or ipv6_pktinfo), the default source address selection algorithm selects one.

The goal of default source address selection is to select the address that is most likely to allow the packet to reach its destination and to support site renumbering. The group of candidate addresses consists of the addresses assigned to the outbound interface (both configured, dynamically generated, or both) or the addresses configured for the outbound interface's SOURCEVIPA interface. Any address that is preferred or deprecated is included in the candidate list. The algorithm is applied to the candidate address list to select the best source address for the packet. If there is only one address in the list of candidate source addresses, then that address is used. If there is more than one address in the candidate list, one is selected by applying the algorithm's rules to the addresses. Rules are applied, in order, to the first and second address, choosing a best address. Rules are then applied to this best address and the third address. This continues until rules have been applied to all addresses. If one of the rules is able to select the best address between two addresses, remaining rules are bypassed for those two addresses. Subsequent rules act as tie-breakers for earlier rules.

Rules:

1. Prefer the same address.

   If either address is the destination address, choose that address as the source address and terminate the entire algorithm.

2. Prefer the appropriate scope.

   If the scope of one address is preferable to the scope of the other address, then the address with the better scope is the better address of these two addresses.

   The following examples show how the scope of one source address (SA) is preferable to the scope of another source address (SB) for the particular destination address (D).

   - Assume that the scope of SA is less than the scope of SB. If the scope of SA is less than the scope of D, then SB is the best address; otherwise, SA is the best address.
   - Assume that the scope of SB is less than the scope of SA. If the scope of SB is less than the scope of D, then SA is the best address; otherwise, SB is the best address.

3. Avoid deprecated addresses.

   If one address is deprecated and the other is preferred, then the preferred address is the better address of the two addresses.

4. Prefer matching label.

   If the label of one source address matches the label of the destination address and the label of the other source address does not match, then the address with the matching label is the better address of the two addresses.

   See “Policy table for IPv6 default address selection” on page 35 and “Configuring the policy table for default address selection” on page 39 for information about how labels are associated with source and destination addresses.

5. Prefer public addresses over temporary addresses.

   If one address is a public address and the other is a temporary address, determine the preference of the application for public or temporary addresses by examining the SRCIP statement:

   - If the SRCIP statement has a JOBNAME PUBLICADDRS entry for this application, then the public address is the better address of the two addresses.
   - If the SRCIP statement has a JOBNAME TEMPADDRS entry for this application, then the temporary address is the better address of the two addresses.
   - If the application has specified the socket option to prefer temporary addresses and there is not an SRCIP statement with a JOBNAME PUBLICADDRS entry for the application, then the temporary address is the better address of the two addresses.
   - If none of the previously listed items are true, then the public address is the better of the two addresses.

6. Use the longest matching prefix.

   If one address has a longer common prefix length (CommonPrefixLength value) with the destination than the other address, then the address with the longer common prefix length is the better address of the two addresses.

7. Leave the order unchanged.

   No rule selected a better address of these two addresses; they are equally good. Choose the first address as the better address of the two addresses.

VIPA considerations with source address selection

If SOURCEVIPA is configured for the outbound interface and the application did not request that SOURCEVIPA is to be ignored (by way of the Ignore Source VIPA socket option), the source address is selected from the addresses of the SOURCEVIPA interface. Otherwise, the source address is selected from the addresses of the outbound interface. Selection of a source VIPA address for IPv6 is done differently from IPv4. It is determined by the SOURCEVIPAINTERFACE parameter that is configured on the outbound interface, rather than the order of the HOME list.

When a socket is used to establish a TCP connection to an IPv6 destination or to send a UDP or RAW IP datagram to an IPv6 destination, the local address of the socket is determined based on the set of rules listed in Table 6 on page 39:

Table 6. Source address selection

Source address selection for communication to IPv6 destinations		**TCP, UDP, and RAW**
IPCONFIG6 NOSOURCEVIPA	1. The socket is bound to a local IPv6 address.	Do not change the local address, use it as it is.
	2. The socket is unbound; it is bound to the unspecified IP address.	Use the IPv6 default source address selection algorithm (selecting an IPv6 address on the physical interface over which the IP packet is about to be sent).
IPCONFIG6 SOURCEVIPA	1. The socket is bound to a local IPv6 address.	Do not change the local address, use it as it is.
	2. A setsockopt() with the NOSOURCEVIPA option was issued for the socket.	Use the IPv6 default source address selection algorithm (selecting an IPv6 address on the physical interface over which the IP packet is about to be sent).
	3. A SOURCEVIPAINTERFACE parameter is configured on the IPv6 INTERFACE definition for the interface over which the IP packet is about to be sent.	Use the IPv6 default source address selection algorithm to select an IPv6 VIPA address from the IPv6 virtual interface pointed to by the SOURCEVIPAINTERFACE parameter.
	4. A SOURCEVIPAINTERFACE parameter is not configured on the IPv6 INTERFACE definition for the interface over which the IP packet is about to be sent.	Use the IPv6 default source address selection algorithm (selecting an IPv6 address on the physical interface over which the IP packet is about to be sent).

Configuring the policy table for default address selection

You can configure the policy table for default address selection to better suit your environment by using the DEFADDRTABLE statement.

Before you begin

Determine whether the default policy table for IPv6 default address selection is appropriate for your environment. If it is not, determine the appropriate policy entries. For more information about the policy table for IPv6 default address selection, including the entries that are in the default table, see "Policy table for IPv6 default address selection" on page 35.

Procedure

Perform the following steps to configure the policy table for default address selection:

1. Take one of the following actions, depending on how you want to configure the table:
   - To configure a new policy table for IPv6 default address selection, add a DEFADDRTABLE block to your TCP/IP profile that contains the appropriate policy entries.
   - To change the policy table that is currently being used for IPv6 default address selection, create a DEFADDRTABLE block that contains the existing set of policies (default or configured policies) and update the policy entries that you want to change.
   - To remove all policies that are currently configured and revert to the default entries, create a DEFADDRTABLE block that does not contain any policies:

     ```
     DEFADDRTABLE
     ENDDEFADDRTABLE
     ```
2. Issue the VARY TCPIP,,OBEYFILE command to replace the existing or default policy entries and to activate the configuration changes.

Example

To prefer IPv4 addresses over IPv6 addresses, you can change the precedence of the ::ffff:0.0.0.0/96 prefix to 100. This precedence value gives the IPv4-mapped IPv6 prefix (::ffff:0.0.0.0/96) a higher precedence than all IPv6 prefixes.

```
DEFADDRTABLE
; Prefix              Precedence  Label
  ::1/128             50          0
  ::/0                40          1
  2002::/16           30          2
  ::/96               20          3
  ::ffff:0.0.0.0/96   100         4
ENDDEFADDRTABLE
```

To sort global destinations before link-local destinations, you can add an entry to the table for the fe80::/10 prefix. This entry gives link-local destinations (which match the fe80::/10 prefix) a lower precedence than all global addresses (which match the ::/0 prefix).

```
DEFADDRTABLE
; Prefix              Precedence  Label
  ::1/128             50          0
  ::/0                40          1
  fe80::/10           33          1
  2002::/16           30          2
  ::/96               20          3
  ::ffff:0.0.0.0/96   100         4
ENDDEFADDRTABLE
```

For information about the DEFADDRTABLE statement, see z/OS Communications Server: IP Configuration Reference.

What to do next

After you have configured the policy table for default address selection, you can display the configured values by issuing the Netstat DEFADDRT/-l command. See "Displaying the policy table for default address selection" on page 40 for more information.

Displaying the policy table for default address selection

You can display the entries that are currently configured in the policy table for default address selection.

Procedure

Issue the Netstat DEFADDRT/-l command to display the values that are currently set in the policy table for default address selection.

The Netstat DEFADDRT/-l report is displayed. This report also indicates whether the policy table settings are the default settings or configured settings.

For more information about the Netstat DEFADDRT/-l report, see z/OS Communications Server: IP System Administrator's Commands.

Enabling IPv6 communication between IPv6 nodes or networks in an IPv4 environment

Figure 9 on page 41 shows how to enable communication between IPv6 nodes or networks in an IPv4 environment:

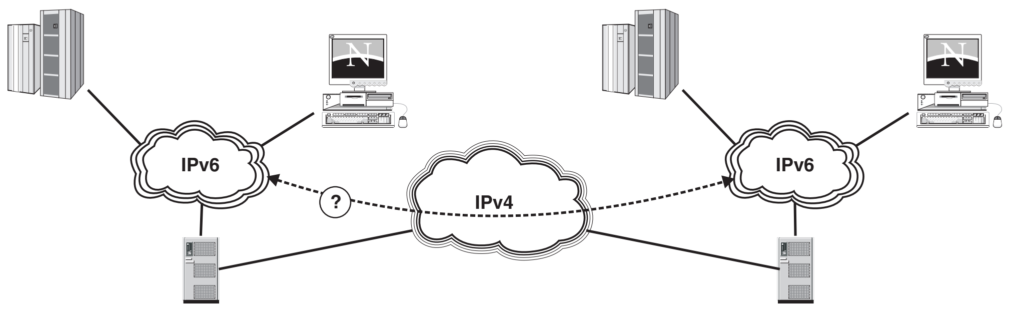


Figure 9. Communicating between IPv6 nodes or networks in an IPv4 environment

Tunneling provides a way to use an existing IPv4 routing infrastructure to carry IPv6 traffic. IPv6 nodes (or networks) that are separated by IPv4 infrastructure can build a virtual link by configuring a tunnel. IPv6-over-IPv4 tunnels are modeled as single-hop. In other words, the IPv6 hop limit is decremented by 1 when an IPv6 packet traverses the tunnel. The single-hop model serves to hide the existence of a tunnel. The tunnel is opaque to the network and is not detectable by network diagnostic tools such as **`traceroute`**.

z/OS Communications Server does not support being a tunnel endpoint. This means that the z/OS Communications Server stack must have an IPv6 interface connected to an IPv6 capable router. The router is relied on to handle all tunneling issues.

For more information, see "Tunneling" on page 111.

Enabling end-to-end communication between IPv4 and IPv6 applications

Figure 10 on page 41 shows communication between IPv4 and IPv6 applications:

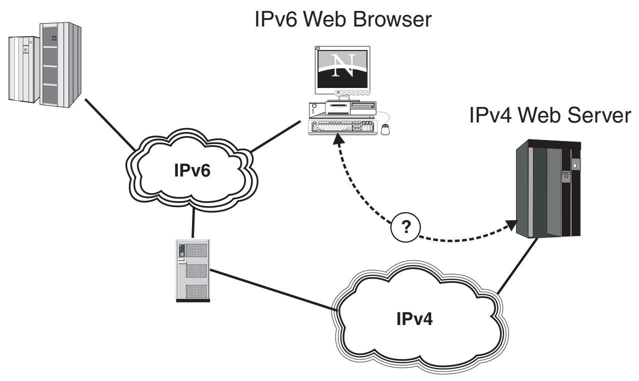


Figure 10. Communicating between IPv4 and IPv6 applications

z/OS Communications Server can be an IPv4-only or dual-mode stack.

There is no support for an IPv6-only stack. By default, IPv6-enabled applications can communicate with both IPv4 and IPv6 peers. A socket option makes an IPv6-enabled application require all peers to be

IPv6. See “Socket option to control IPv4 and IPv6 communications” on page 83 for detailed information about the IPV6_V6ONLY socket option.

IPv6 application on a dual-mode stack

An IPv6 application on a dual-mode stack can communicate with IPv4 and IPv6 partners as long as it does not bind to a native IPv6 address. If it binds to a native IPv6 address, it cannot communicate with an IPv4 partner because the native IPv6 address cannot be converted to an IPv4 address.

If a partner is IPv6, all communication uses IPv6 packets.

If a partner is IPv4, the following results occur:

- Both source and destination are IPv4-mapped IPv6 addresses.
- On inbound, the transport protocol layer maps the IPv4 addresses in received IPv4 packets to their corresponding IPv4-mapped IPv6 addresses before returning to the application with AF_INET6 addresses.
- On outbound, the transport protocol layer converts the IPv4-mapped IPv6 addresses to native IPv4 addresses and sends IPv4 packets.

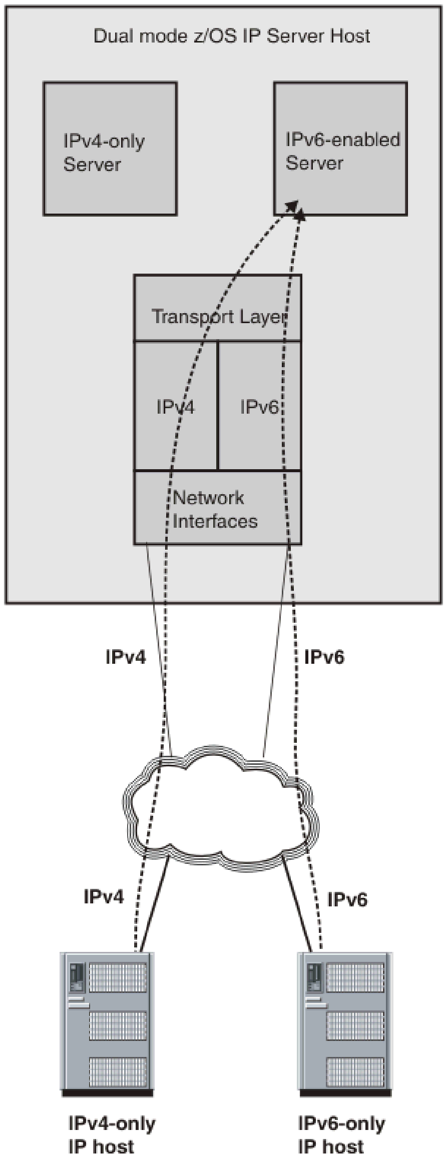


Figure 11. IPv6 application on dual-mode stack

IPv4 application on a dual-mode stack

An IPv4 application running on a dual-mode stack can communicate with an IPv4 partner. The source and destination addresses are native IPv4 addresses and the packet is an IPv4 packet.

If a partner is IPv6 enabled and running on an IPv6-only stack, then communication fails. The partner has only a native IPv6 address (not an IPv4-mapped IPv6 address). The native IPv6 address for the partner cannot be converted into a form that the AF_INET application understands.

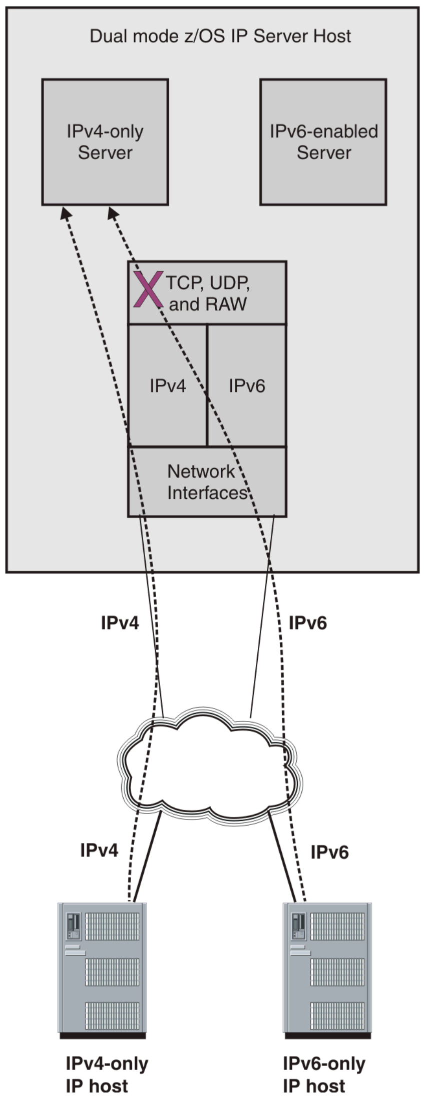


Figure 12. IPv4-only application on a dual-mode stack

Application layer gateways and protocol translation

When IPv6-only nodes begin to appear in the network, AF_INET6 applications on these nodes might need to communicate with AF_INET applications. For a multihomed dual-mode IP host, it is a likely that the host has both IPv4 and IPv6 interfaces over which requests for host-resident applications are received or sent. IPv4-only (AF_INET sockets) applications are not generally able to communicate with IPv6 partners, which means that only the IPv4 partners in the IPv4 network can communicate with those applications; an IPv6 partner cannot.

As soon as IPv6-only hosts are being deployed in a network, applications on those IPv6-only nodes cannot communicate with the IPv4-only applications on the dual-mode hosts, unless one of multiple migration technologies are implemented either on intermediate nodes in the network or directly on the dual-mode hosts.

Numerous RFCs describe solutions in this area. One solution is a SOCKS64 implementation that works as a Sockets Secure (SOCKS) server that relays communication between IPv4 and IPv6 flows. SOCKS is a well-known technology, and the issues around it are familiar. Servers do not require any changes, but client applications (or the stack on which the client applications reside) need to be socks-enabled to be able to reach out through a SOCKS64 server to an IPv4-only partner.

Other solutions are based on a combination of network address translation, IP-level protocol translation, and DNS-flow catcher/interpreter. These solutions all have problems with application-level IP address awareness and end-to-end security.

Requirement: z/OS Communications Server TCP/IP does not provide a SOCKS64 server and does not contain NAT-PT functionality. If an IPv6-only client requires access to an IPv4-only server on z/OS, an external SOCKS64 or NAT-PT node is required to translate the IPv6 packet to a corresponding IPv4 packet and vice versa.

Network address translation

IPv4 NAT translates one IPv4 (private) address into another IPv4 (external) address. IPv6 NAT-PT translates an IPv4 address into an IPv6 address.

Rules: There are several limitations with NAT-PT:

- All requests and responses pertaining to a session must be routed through the same NAT-PT translator.
- There is a protocol translation limitation because a number of IPv4 fields have changed meaning in IPv6. Details of IPv4 to IPv6 protocol translation can be found in the Stateless IP/ICMP Translation Algorithm (SIIT) RFC.
- If an application carries the IP address in the payload, ALGs must be incorporated.
- Lack of end-to-end security. The two end nodes that seek IPSec network level security must both use IPv4 or IPv6.
- DNS messages and DNSSEC translation. An IPv4 end-node that demands DNS replies be signed rejects replies that have been tampered with by NAT-PT.

Considerations for configuring z/OS for IPv6

This topic describes some general considerations for configuring IPv6 on z/OS, including cases where multiple types of TCP/IP stacks are present.

In this topic, stack or TCP/IP stack is used as a generic term to describe a protocol stack that can be defined as a z/OS UNIX System Services AF_INET physical file system (PFS) in the BPXPRMxx parmlib member (for example, z/OS CS TCP/IP).

IPv4-only stack

Some TCP/IP stacks support only IPv4 interfaces and are capable of sending or receiving only IPv4 packets. These TCP/IP stacks are generally referred to as IPv4-only stacks, as they support IPv4 but do not support communication over IPv6 networks.

An IPv4-only stack supports AF_INET socket applications, but does not support AF_INET6 socket applications.

Guideline: z/OS Communications Server TCP/IP can be started as an IPv4-only stack.

IPv6-only stack

An IPv6-only stack supports IPv6 interfaces, but it does not support IPv4 interfaces. These TCP/IP stacks support AF_INET6 sockets and applications that use them, as long as the IP addresses that are used are not IPv4-mapped IPv6 addresses. They do not support AF_INET sockets. Applications can send and receive IPv6 packets by way of an IPv6-only stack, but they cannot send and receive IPv4 packets.

Restriction: z/OS Communications Server TCP/IP cannot be started as an IPv6-only stack.

Dual-mode stack

Many IPv6 TCP/IP stacks support both IPv4 and IPv6 interfaces and are capable of receiving and sending IPv4 and IPv6 packets over the corresponding interfaces. These TCP/IP stacks are generally referred to as dual-mode stacks. This does not indicate that there are two separate TCP/IP stacks running on such a node, but it does indicate that the TCP/IP stack has built-in support for both IPv4 and IPv6.

A dual-mode stack supports AF_INET and AF_INET6 socket applications. AF_INET applications can communicate using IPv4 addresses. IPv6-enabled applications that use AF_INET6 sockets can communicate using both IPv6 addresses and IPv4 addresses (using the IPv4-mapped IPv6 address format).

Guideline: z/OS Communications Server TCP/IP can be started as a dual-mode stack.

INET considerations

This topic describes the INET considerations for IPv4-only and dual-mode IPv4/IPv6 stacks.

IPv4-only stack

An IPv4-only stack supports AF_INET applications, but it does not support AF_INET6 applications. Start an IPv4-only stack in an integrated sockets environment in one of the following ways:

- Do not code an AF_INET6 statement in BPXPRMxx. This method is the easier of the two. When AF_INET6 is not enabled, the underlying TCP/IP stack is started as an IPv4-only stack, even if it is capable of supporting IPv6.

  Restriction: This is the only way to start z/OS Communications Server TCP/IP as an IPv4-only stack in an integrated sockets environment.
- Run a TCP/IP stack that is not capable of supporting IPv6. When starting a TCP/IP stack that does not support IPv6, the stack ignores any AF_INET6 definitions that might appear in BPXPRMxx. As a result, the stack is started as an IPv4-only stack, even when AF_INET6 is coded in BPXPRMxx.

When a TCP/IP stack is started as an IPv4-only stack in an Integrated Sockets environment, applications can open AF_INET sockets and can send and receive IPv4 packets over IPv4 interfaces only. However, applications are unable to open AF_INET6 sockets.

Dual-mode IPv4/IPv6 stack

When both AF_INET and AF_INET6 are coded in BPXPRMxx and a dual-mode-capable stack is started, both AF_INET and AF_INET6 sockets are supported by the stack, and applications can send and receive IPv4 and IPv6 packets.

Requirements: To enable AF_INET6 support in an integrated sockets environment, the following two conditions must exist:

- AF_INET6 must be configured in BPXPRMxx. Note that AF_INET6 support can be dynamically enabled by configuring AF_INET6 in BPXPRMxx and then issuing the SETOMVS RESET= command to activate the new configuration.
- A dual-mode capable stack must be started after AF_INET6 is configured in BPXPRMxx. If a TCP/IP stack that is capable of being a dual-mode stack is started before BPXPRMxx is configured, the stack remains an IPv4-only stack as long as it remains active; however, if the stack is stopped and then restarted, it restarts as a dual-mode TCP/IP stack if AF_INET6 is configured in BPXPRMxx at the time it is restarted.

Requirement: To enable AF_INET6 support for z/OS Communications Server TCP/IP, z/OS Communications Server TCP/IP must be started as a dual-mode stack. z/OS Communications Server TCP/IP does not support being started as an IPv6-only stack. In other words, if AF_INET6 is coded in BPXPRMxx, AF_INET must also be coded. If it is not, then the z/OS Communications Server TCP/IP stack fails to initialize.

Common INET considerations

This topic describes Common INET considerations.

Enabling AF_INET6 support in a Common INET environment

Requirements: To enable AF_INET6 support in a Common INET environment, the following conditions must exist:

- AF_INET6 must be configured in BPXPRMxx. AF_INET6 support can be dynamically enabled by configuring AF_INET6 in BPXPRMxx and then issuing the SETOMVS RESET= command to activate the new configuration.
- At least one dual-mode-capable stack must be started after AF_INET6 is configured in BPXPRMxx. Note that any dual-mode capable TCP/IP stack started before configuring BPXPRMxx remains an IPv4-only stack as long as it remains active. However, if it is stopped and then restarted, it restarts as a dual-mode TCP/IP stack if AF_INET6 is configured in BPXPRMxx at the time it is restarted.

Guideline: Do not start some z/OS Communications Server TCP/IP stacks with AF_INET6 support and some without AF_INET6 support. If AF_INET6 support is dynamically enabled, you should stop and restart all TCP/IP stacks which were active when AF_INET6 support was enabled. This allows these TCP/IP stacks to become dual-mode stacks. After this occurs, all applications which are capable of opening AF_INET6 sockets should be stopped and restarted. This allows the restarted applications to communicate over IPv4 and IPv6 networks.

Disabling AF_INET6 support in a Common INET environment

You can disable AF_INET6 support in a Common INET environment in one of two ways.

Procedure

- Stop all active dual-mode TCP/IP stacks while IPv4-only stacks remain active.

  Applications are no longer able to open AF_INET6 sockets, although they can continue to use any AF_INET6 sockets that are already open and not bound to one of the stopped dual-mode TCP/IP stacks. However, applications are able to open AF_INET sockets.
- Dynamically disable AF_INET6 in BPXPRMxx and stop all active dual-mode TCP/IP stacks.

  When restarted, the dual-mode-capable TCP/IP stacks start as IPv4-only stacks. In effect, this is a subset of the previous case. To disable AF_INET6 support, issue the SETOMVS RESET= command to set the AF_INET6 MAXSOCKETS value to 0.

Supporting a mixture of dual-mode stacks and IPv4-only stacks

When AF_INET6 sockets are supported, an IPv6-enabled application can use an AF_INET6 socket to send and receive data with both IPv4 and IPv6 partners. When communicating with an IPv6 partner, a native IPv6 address is used. When communicating with an IPv4 partner, the IPv4 address is encoded as an IPv4- mapped IPv6 address. When an IPv4-mapped IPv6 address is used on an AF_INET6 socket, a dual-mode TCP/IP stack realizes the partner is attached to the IPv4 network and routes packets over IPv4 interfaces.

As long as all TCP/IP stacks started in a Common INET environment provide native support AF_INET6 sockets, socket calls can be passed directly to the underlying TCP/IP stack. However, when both dual-mode stacks and IPv4-only stacks are started in a Common INET environment, the IPv4-only stacks are not able to process the native AF_INET6 socket calls. As a result, an application which uses IPv4-mapped IPv6 addresses on an AF_INET6 socket needs transformations done by Common INET to communicate with partners over any active IPv4-only stack.

Common INET provides AF_INET6 transformations that allow AF_INET6 applications to communicate with an IPv4 peer over IPv4-only stack. The AF_INET6 transformations convert AF_INET6 socket calls to the corresponding AF_INET socket calls before sending them to an IPv4-only stack and converts AF_INET responses received from the IPv4-only stack to the corresponding AF_INET6 responses before making them available to the AF_INET6 application. Even with this transformation, AF_INET6 applications must use IPv4-mapped IPv6 addresses to communicate with IPv4 applications.

Figure 13 on page 47 shows a mixture of dual-mode stacks and IPv4-only stacks:

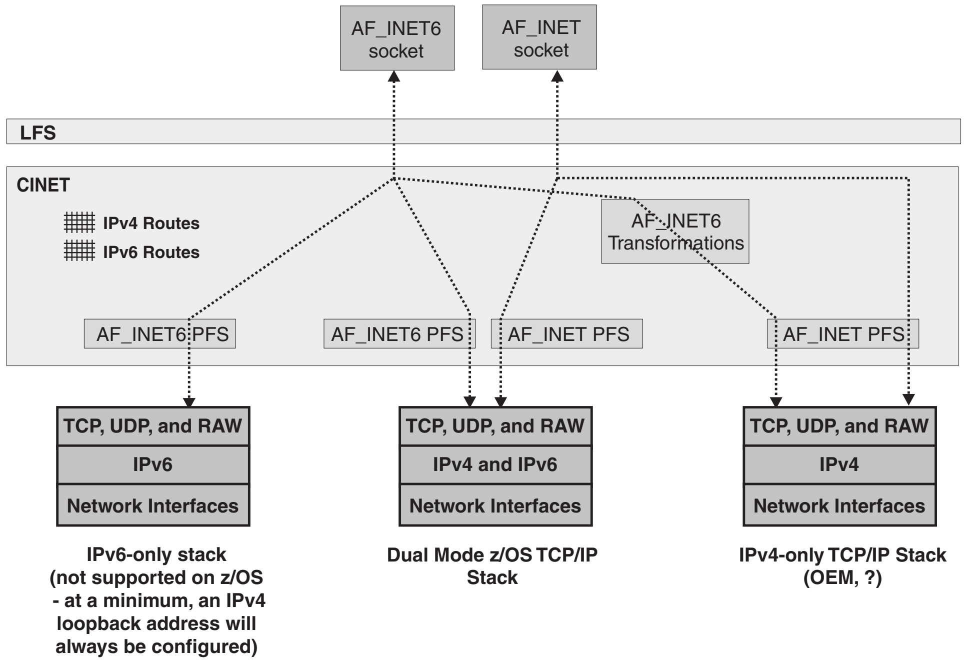


Figure 13. Mixing dual-mode and IPv4-only stacks

Configuring a Common INET environment

If a mixture of dual-mode capable stacks and IPv4-only stacks are started in a Common INET environment, the default stack should be one of the dual-mode capable stacks. Common INET routes certain requests to the default stack, and this enables the stack with more functional capability to process these requests.

If AF_INET6 support is dynamically configured in BPXPRMxx, stop and restart all dual-mode-capable TCP/IP stacks. After the TCP/IP stacks have been stopped and restarted, stop and restart all IPv6-enabled applications.

Chapter 4. Configuring support for z/OS

This topic describes the configuration support that is needed for z/OS and contains the following subtopics:

Ensure that important features are supported over IPv6

See Appendix A, “IPv6 support tables,” on page 119 to ensure that all needed features are supported over IPv6.

Assess automation and application impacts because of Netstat and message changes

Netstat output for stacks that are IPv6 enabled has a different format in order to accommodate the longer IPv6 address. This becomes an issue when applications that parse Netstat output are used. The same considerations also apply to applications which use IP addresses in their automation because IP addresses now have a longer format.

Determine how remote sites connect to the local host

It is likely that clients that are not connected to a link that is directly attached to a z/OS image require access to servers that run on that z/OS image. Because z/OS provides a dual-stack implementation, z/OS can send IPv4 packets to partner nodes that are connected to the IPv4 network and IPv6 packets to partner nodes that are connected to the IPv6 network. If the client node is connected to the same routing infrastructure as the z/OS node, traffic is routed between z/OS and the client node by way of the native network transport.

In some cases, the two nodes might not be connected to the same routing infrastructure. For instance, each node might be attached to distinct IPv6 networks that are separated by an intermediate IPv4 network. When this occurs, tunneling might be used to transmit the native IPv6 packets across the IPv4 network. This allows nodes in the disjoint IPv6 networks to send packets to one another.

z/OS does not support functioning as an endpoint for this type of tunnel. However, z/OS might route traffic over a tunnel in the intermediate network. In this case, the tunnel endpoint used by z/OS would be an IPv6/IPv4 router in the network that supports one of several tunneling protocols. The tunnel endpoint used by z/OS might be attached to the same LAN to which z/OS attaches or might be attached to a remote network link. In either case, the presence of the tunnel endpoint is transparent to z/OS; from the z/OS perspective, traffic is routed over the native IPv6 network.

SNA access

Both Enterprise Extender and TN3270 allow access to SNA applications over an IPv6 network as well as an IPv4 network. For both protocols, it is possible to simultaneously support connectivity over IPv4 and IPv6 networks. Enterprise Extender uses separate path statements and connection networks for each protocol. By assigning different weights to Transmission Groups that use different network protocols, it is possible to have SNA traffic prefer being routed over the IPv6 network or the IPv4 network. For TN3270, the network protocol used is determined by the remote TN3270 client.

Guideline: For Enterprise Extender and TN3270, use global unicast addresses. Although link-local addresses might work in certain configurations, they are not suitable for use when connecting between partner companies. There are few, if any, IPv6 NAT devices which can perform the necessary mappings between limited scope addresses and globally routable addresses and, given the vast number of globally unique IPv6 addresses available, are not necessary.

Avoid using IP addresses for identifying remote hosts

In IPv4 networks, some sites and applications attempt to use the remote IP address to identify the client node that is connecting. In general for IPv4, do not use the IP address to identify the remote host. The client address can often be unpredictable, either because the client is using DHCP to obtain its address or because the client is accessing the server from behind a network address translation (NAT) device.

In IPv6, the client address is likely to become even more volatile than it is in IPv4 networks. Using Stateless Address Autoconfiguration, a client's address is dynamically derived from the MAC address of the network adapter used for connectivity. IPv6 also allows clients to pseudo-randomly generate IP addresses, referred to as temporary addresses, which can be used for one or more connections. These temporary addresses can be generated as frequently as the client desires- once a day, once an hour, or even more frequently. In general, the temporary addresses are not placed in the DNS, making it impossible to use DNS to map the IP address to a host name.

Result: The client IP addresses are unpredictable and subject to frequent change. In addition, it is possible, and even likely, that a server is unable to map the client address to a host name. If a mechanism to identify the remote host is required, then a different mechanism (client certificate, password, and so on) should be used to identify the remote host. For example, this approach is used by Enterprise Extender. For IPv6, Enterprise Extender does not support configuring or passing IPv6 addresses. Instead, it uses host names to identify Enterprise Extender nodes.

Using the BIND parameter on the PORT statement

The PORT statement reserves a port for the use of a particular server. The statement does not typically distinguish between IPv4 and IPv6; the port is reserved regardless of which type of address the application uses.

Use the BIND keyword on the PORT statement to force a listener that binds to the IPv4 INADDR_ANY address, or to the IPv6 unspecified address (in6addr_any), to listen on a particular IP address. If you specify an IPv4 address on the BIND keyword, listeners that are bound to the INADDR_ANY address are converted to the specified IPv4 address, and listeners that are bound to the in6addr_any address are converted to the IPv4-mapped IPv6 form of the specified address. If you specify an IPv6 address on the BIND keyword, the address is ignored for IPv4 listeners that are bound to the INADDR_ANY address, and listeners that are bound to the in6addr_any address are converted to the specified IPv6 address.

If you use the BIND option, your server can listen for either IPv4 connections or IPv6 connections, but not both. To have the same service serve both IPv4 and IPv6 clients, you might need to start two instances of it, one bound to an IPv4 address and one to an IPv6 address.

With SHAREPORT or SHAREPORTWLM keyword, you can start multiple instances of the server and have connections automatically load balanced between them. This function is supported for TCP listeners only. All IPv4 connection requests are load balanced between the set of IPv4 listeners (including AF_INET6 listeners bound to the IPv6 unspecified address in6addr_any), while all IPv6 connection requests are load balanced between the set of IPv6 listeners. See z/OS Communications Server: IP Configuration Reference for information about the load balancing algorithms used by each of these parameters.

Security considerations

On z/OS Communications Server, not all security features that are supported over an IPv4 transport are enabled when communicating by an IPv6 transport. For example, IPSec, Network Access Control, Stack and Port Access Control, TLS, SSL, and Kerberos (Kerberos Version 5 and GSSAPIs) are enabled for both IPv4 and IPv6, whereas NAT traversal is enabled for IPv4 only. See Table 39 on page 124 for a list of features supported for IPv4 or IPv6.

When a security function is supported over IPv4 but not over IPv6, the security feature is exercised when data is transmitted over the IPv4 transport. This is true whether the application uses AF_INET or AF_INET6 sockets. However, when an AF_INET6 socket application communicates over the IPv6 transport, security features that are supported over IPv4 only are not exercised.

Result: For the same local application, some security features can be exercised when communicating by way of IPv4, but not when communicating by way of IPv6.

To avoid creating a potential security exposure, it is important to determine if any important security features are supported over IPv4 but not over IPv6 before enabling AF_INET6 on a given LPAR. If only a subset of applications uses such a security feature, then it is sufficient to ensure that those applications communicate only over the IPv4 transport.

To ensure that the IPv4 transport is used, the following methods are available:

- Verify that the application uses AF_INET sockets. Applications that use AF_INET sockets are able to communicate only by way of the IPv4 transport.
- Configure the application to bind to an IPv4 address. Applications that bind to an IPv4 address are able to communicate using the IPv4 transport only.
- Use the BIND parameter on the PORT statement to cause the application to bind to an IPv4 address.

Support for scope information

Scope information defines an outbound routing interface. Scope information can be an interface name that you configure or an interface index value that z/OS assigns. The z/OS resolver supports the inclusion of scope information about host names or IPv6 addresses that are resolved using getaddrinfo; this support can also return scope information about host names that are resolved from IPv6 link-local addresses that are input using getnameinfo. Applications such as Ping, Traceroute, FTP, and others, use the z/OS resolver getaddrinfo and getnameinfo processing for resolving host name information and can use this scope information support when appropriate. Within z/OS, scope information is applicable only to IPv6 link-local addresses.

Restriction: Scope information that is specified for other IPv6 addresses, or for host names that resolve to other types of addresses, is ignored. Scope information that is appended to an IPv4 address is treated as an error.

This resolver capability can be useful in situations where locally attached devices (for instance, a router) are not yet fully configured and can be reached only using the link-local IPv6 address that is associated with the interface that connects this host to the device. It can also be useful if locally attached devices are malfunctioning or cannot be reached through normal routing mechanisms; diagnostic efforts are directed over a specific interface to the malfunctioning device. Finally, in installations that use static routing, scope information can be useful with applications such as FTP and Traceroute for identifying the correct interface to be used when a local IPv6 address is specified as the target address. For a list of z/OS applications that support use of scope information, see "Application support of scope information specified on host name or IP address" on page 120.

For details and restrictions about the z/OS resolver support for scope information about getaddrinfo and getnameinfo, see "Name and address resolution functions" on page 72.

For details about the interaction of scope information and advanced IPv6 socket options for specifying the outgoing interface, see "Options for specifying the outgoing interface" on page 108.

Considerations for choosing interface name or interface index

The interface index for an interface is assigned by the stack during interface definition processing; the value remains constant until either the interface is deleted from the stack or the stack is stopped. The same interface can be assigned a different interface index value when the stack is reactivated. Because of this, a constant value for the interface index for a given interface should not be assumed.

In a CINET environment, the interface index includes a stack identifier (known as the transport driver index). The transport driver index makes the interface index for an interface unique across the entire CINET environment, but reduces the predictability of the interface index value for an interface. Applications or users that provide scope information about host names should specify an interface name, instead of an interface index, for more predictable processing. This includes cases in which host names or IPv6 addresses are specified in a configuration file (such as the *userid*.RHOSTS.DATA) that is used to match against command input host names or IPv6 addresses, or against remote partner host names or IPv6 addresses. Host names or IPv6 addresses in this situation should also use the interface name, not the interface index, as the scope information that is coded on the host name or IPv6 address for matching purposes, because the z/OS resolver returns interface names by default on getnameinfo calls that involve scope information.

Syntax for specifying scope information

Scope information is specified as part of host name information, in the form *host_identifier %scope_information*.

The following guidelines apply when specifying scope information:

- The *host_identifier* value is the host name or IPv6 link-local address of the host. Because scope information applies only to IPv6 link-local addresses, and IPv6 link-local addresses are not guaranteed to be unique, DNS host names are not typically created for IPv6 link-local addresses. Scope information is typically used as an appendage to a specified IPv6 address but not to an actual host name.
- The percent (%) character is a delimiter between the host identifier portion and the scope portion of the input character string.
- The *scope_information* value is the interface name or interface index used to identify the local outbound routing interface that is used with the *host_identifier*. This value should be an interface name; the name has a maximum length of 16 characters in the z/OS environment. If an interface index is used instead of an interface name, it must be in decimal format, and it must include the transport driver index value when operating in a CINET environment. See the SIOCGIFNAMEINDEX ioctl function call information in z/OS UNIX System Services Programming: Assembler Callable Services Reference for information about interface index in a CINET environment.

The following examples show how to specify scope information:

- When the scope information is an interface name, specify:
  - ping fe80::9:47:100:112%*interfacename*

- When the scope information is an interface index, specify:
  - ping fe80::9:47:100:112%65541

  The decimal value, 65541, represents the hexadecimal interface index value '00010005'x. The first halfword of the value (the transport driver index value) indicates which stack under CINET the interface belongs to. The second halfword contains the interface index value assigned by that stack to represent this interface.

The combined length of the *host_identifier* value and the *scope_information* cannot exceed 255 characters. This restriction applies to both values that are specified as input and values that are received or displayed as output. If host names are used for IPv6 link-local addresses, assign host names such that the 255 character limitation, with scope information appended, is maintained. The getaddrinfo invocations fail for host names longer than 255 characters, and the getnameinfo invocations return truncated host name information if the resolved name (and scope) exceeds the 255 character maximum.

Enabling IPv6 support

z/OSCommunications Server can be run as an IPv4-only stack or as a dual-mode stack (IPv4 and IPv6). The BPXPRMxx parmlib member determines which mode is used. The following configurations are possible:

- INET IPv4 only
- INET IPv4/IPv6 dual-mode stack
- CINET IPv4 only
- CINET IPv4/IPv6 dual-mode stack

Restriction: After a stack has been started, you must stop and restart the stack to change the mode of the stack.

You can configure AF_INET alone or both AF_INET and AF_INET6. Although coding AF_INET6 alone is not prohibited, TCP/IP does not start because the master socket is AF_INET and the call to open it fails.

INET IPv4-only BPXPRMxx sample definition

IPv4-only stack support is defined by using one NETWORK statement (for AF_INET) in the BPXPRMxx parmlib member. For example:

```
FILESYSTYPE Type(INET) Entrypoint(EZBPFINI)
NETWORK DOMAINNAME(AF_INET)
        DOMAINNUMBER(2)
        MAXSOCKETS(2000)
        TYPE(INET)
```

INET IPv4/IPv6 dual-mode stack BPXPRMxx sample definition

Dual-mode stack support is defined by using two NETWORK statements (one for AF_INET and one for AF_INET6) in the BPXPRMxx parmlib member. For example:

```
FILESYSTYPE Type(INET) Entrypoint(EZBPFINI)
NETWORK DOMAINNAME(AF_INET)
        DOMAINNUMBER(2)
        MAXSOCKETS(2000)
        TYPE(INET)
NETWORK DOMAINNAME(AF_INET6)
        DOMAINNUMBER(19)
        MAXSOCKETS(3000)
        TYPE(INET)
```

Separate MAXSOCKETS values are supported. The IPv6 default is the IPv4 specified value.

CINET IPv4-only BPXPRMxx sample definition

Multiple TCP/IP stacks in one MVS image or LPAR are supported only by using Common INET (CINET). Each TCP/IP stack is defined in the BPXPRMxx parmlib member using a SUBFILESYSTYPE statement. These definitions are identical to what was used before IPv6 support. The following example shows the definitions for three IPv4-only stacks:

```
FILESYSTYPE TYPE(CINET) ENTRYPOINT (BPXTCINT)
NETWORK DOMAINNAME(AF_INET)
        DOMAINNUMBER(2)
        MAXSOCKETS(2000)
        TYPE(CINET)
        INADDRANYPORT(20000)
        INADDRANYCOUNT(100)
SUBFILESYSTYPE NAME(TCPCS)  TYPE(CINET) ENTRYPOINT(EZBPFINI)
SUBFILESYSTYPE NAME(TCPCS2) TYPE(CINET) ENTRYPOINT(EZBPFINI)
SUBFILESYSTYPE NAME(TCPCS3) TYPE(CINET) ENTRYPOINT(EZBPFINI)
```

CINET IPv4/IPv6 dual-mode stack BPXPRMxx sample definition

Dual-mode (IPv4/IPv6) stack support is defined by using two NETWORK statements in the BPXPRMxx member. Each TCP/IP stack is defined in the BPXPRMxx parmlib member by using a SUBFILESYSTYPE statement. All z/OS Communications Server stacks that are defined under the two NETWORK statements are dual-mode (IPv4/IPv6) stacks. The following example shows the definitions for three dual-mode stacks:

```
FILESYSTYPE TYPE(CINET) ENTRYPOINT(BPXTCINT)
NETWORK DOMAINNAME(AF_INET)
         DOMAINNUMBER(2)
         MAXSOCKETS(2000)
         TYPE(CINET)
         INADDRANYPORT(20000)
         INADDRANYCOUNT(100)
NETWORK DOMAINNAME(AF_INET6)
         DOMAINNUMBER(19)
         MAXSOCKETS(3000)
         TYPE(CINET)
SUBFILESYSTYPE NAME(TCPCS)  TYPE(CINET) ENTRYPOINT(EZBPFINI)
SUBFILESYSTYPE NAME(TCPCS2) TYPE(CINET) ENTRYPOINT(EZBPFINI)
SUBFILESYSTYPE NAME(TCPCS3) TYPE(CINET) ENTRYPOINT(EZBPFINI)
```

TCP/IP profile configuration statements for configuring IPv6

IPv6 must be enabled before IPv6 addresses can be coded on the following configuration statements:

- BEGINROUTES
- DELETE PORT
- INTERFACE
- IPCONFIG6
- IPSEC
- OSAENTA
- PKTTRACE
- PORT
- VIPABACKUP
- VIPADEFINE
- VIPADISTRIBUTE
- VIPARANGE
- VIPAROUTE

You need to be aware of the following configuration items when you are configuring IPv6:

- Use the BEGINROUTES statement to add static IPv6 routes to the IP routing table.

- You can still use DEVICE and LINK statements to define IPv4 interfaces on an IPv6-enabled stack. However, you cannot use DEVICE and LINK statements to define IPv6 interfaces. You must use the INTERFACE statement to define IPv6 interfaces.
- The SOURCEVIPA option of the IPCONFIG6 statement has a dependency on the INTERFACE statement. You must specify the SOURCEVIPAINTERFACE parameter on the INTERFACE statement for each interface on which you want SOURCEVIPA to take effect.

For more information about these statements, see z/OS Communications Server: IP Configuration Reference.

Resolver processing

IPv6 support introduces several changes to how host name and IP address resolution is performed. These changes affect several areas of resolver processing, including:

- New resolver APIs are introduced for IPv6-enabled applications. For more information, see “Name and address resolution functions” on page 72.
- New DNS IPv6 AAAA records are defined in place of DNS IPv4 A records to represent hosts with IPv6 addresses, resulting in new network flows between resolvers and name servers.
- A new algorithm is defined to describe how a resolver needs to sort a list of IP addresses returned for a multihomed host. For more information, see “Default destination address selection” on page 36.
- New statements in the resolver configuration files are defined, and new search orders are implemented for local host tables processing.

Resolver configuration

To avoid impacting existing IPv4 queries, the use of the /etc/hosts, HOSTS.LOCAL, HOSTS.SITEINFO, and HOSTS.ADDINFO files continues to be supported for IPv4 addresses only. The HOSTS.SITEINFO and HOSTS.ADDRINFO files continue to be generated from the HOSTS.LOCAL file by way of the MAKESITE utility.

ETC.IPNODES is a new local host file (in the style of /etc/hosts) that might contain both IPv4 and IPv6 addresses. IPv6 addresses can be defined in ETC.IPNODES only. The introduction of this file allows the administration of local host files to more closely resemble that of other TCP/IP platforms and eliminates the requirement of post-processing the files (specifically, MAKESITE).

The following new search order is used for selecting new ETC.IPNODES local host files for IPv6 searches in MVS and UNIX environments:

1. GLOBALIPNODES
2. RESOLVER_IPNODES environment variable (UNIX only)
3. userid/jobname.ETC.IPNODES
4. *hlq*.ETC.IPNODES
5. DEFAULTIPNODES
6. /etc/ipnodes

The IPv6 search order is simplified, but to minimize migration concerns, the IPv4 search order continues to be supported as in previous releases. The side effect of this is that, by default, you need to maintain two different local host files for your system. For example, IPv4 addresses in HOSTS.LOCAL, and IPv6 and IPv4 addresses in ETC.IPNODES.

An easier approach is to use the new COMMONSEARCH statement in the resolver setup file. By specifying COMMONSEARCH, you indicate that only the new IPv6 search order should be used, regardless of whether the search is for IPv6 or IPv4 resources. This means that only one file (ETC.IPNODES) has to be managed for the system, and that all the APIs use the same single file. The use of COMMONSEARCH reduces IPv6 and IPv4 searching to a single search order, and also reduces the UNIX and native MVS environments to a single search order.

For detailed information about search orders, see z/OS Communications Server: IP Configuration Guide.

IPv4-only configuration statements

The TCPIP.DATA SORTLIST statement is used for sorting IPv4 addresses only; the default destination address selection algorithm is used to sort IPv6 addresses.

IPv6/IPv4 configuration statements

Use the following statements for IPv6/IPv4 configuration:

COMMONSEARCH/NOCOMMONSEARCH resolver setup statement
Use these statements when a common local host file search order is to be used or not used. The COMMONSEARCH statement allows the same search order of local host files be used for an IPv4 or IPv6 query. It also allows the same search order to be used in both the native MVS and z/OS UNIX environments.

DEFAULTIPNODES resolver setup statement
Use this statement to specify the default local host file.

GLOBALIPNODES resolver setup statement
Use this statement to specify the global local host file.

NAMESERVER/NSINTERADDR TCPIP.DATA statement
Use this statement to specify the IPv4 or IPv6 address of a name server.

Resolver communications with the Domain Name System

To retrieve IPv6 data from the correct name server, ensure that the resolver configuration data set points to name servers that can resolve the IPv6 queries. A resolver does not have to communicate with a name server over an IPv6 network in order to retrieve IPv6 Domain Name System (DNS) entries. The z/OS resolver can use IPv4, IPv6, or both to communicate with a name server.

IPv6 resource records are larger than IPv4 resource records; therefore, DNS response messages are larger for IPv6 resources than for IPv4 resources. If the number of resource records in a DNS response message is large, the response message from the name server might exceed 512 bytes of data. If more than 512 bytes of data is needed to send the message, the message is truncated to fit in 512 bytes of UDP packet data. The resolver then resends the request using TCP protocols so that the name server can send the entire response message.

To eliminate the performance costs associated with switching from UDP to TCP protocols, the z/OS resolver can use Extension mechanisms for DNS (EDNS0). EDNS0 uses UDP protocols to accept messages that are greater than 512 bytes, when the name server that sends the response messages also supports EDNS0. The z/OS resolver can accept up to 3072 bytes of DNS response message data in a single UDP packet. (If the name server does not support EDNS0, responses that are larger than 512 bytes in length are truncated and resent using TCP protocols.)

The resolver dynamically determines which name servers support EDNS0 processing and modifies the DNS requests that it sends to the name server. If a name server is upgraded to support EDNS0, the resolver discovers this upgrade dynamically. The length of time that the discovery process takes depends on the frequency with which DNS responses are truncated to use UDP protocols. You can issue the MODIFY REFRESH command to cause the resolver to discover the upgrade more quickly. See z/OS Communications Server: IP System Administrator's Commands for the syntax and description of the MODIFY REFRESH command.

User exits

Several TCP/IP applications provide exit facilities that can be used for a variety of purposes. Several of these exits include IP addresses or SOCKADDR structures as part of the parameters passed to the exits.

The following exits are available to support IPv6 addresses or SOCKADDR structures:

- FTP - All FTP exits have been enhanced to support IPv6 addresses except for FTPSMFEX. Samples for these exits are provided in SEZAINST. See z/OS Communications Server: New Function Summary for more information about changes to these exits:
  - FTCHKCMD
  - FTCHKCM1
  - FTCHKCM2
  - FTCHKJES
  - FTCHKPWD
  - FTPOSTPA
  - FTPOSTPR
- The TSO remote execution server user exit - RXEXIT.

Which applications started with inetd are IPv6 enabled?

The following z/OS UNIX applications support IPv6 addresses:

- Internet daemon (inetd) server
- Remote execution (orexecd) server
- Remote shell (orshd) server
- Telnet server (otelnetd)

Modifying the inetd.conf file

You must modify the `inetd.conf` file to support the IPv6-enabled applications. The z/OS UNIX rsh server and Telnet server support Kerberos for IPv4 connections, but not for IPv6 connections.

Procedure

- In the `inetd.conf` file, specify tcp6 for the protocol of the service name.

  For the z/OS UNIX servers to support IPv6 connections, you must specify this option in the `inetd.conf` file. When you specify tcp6, IPv4 clients are also supported.

IPv6 and SMF records

Most of the TCP/IP SMF records currently contain IP addresses as part of their content. The data in these records is typically processed by programs, some of which are real-time SMF exits and others that post-process the SMF records after the records are created. The TCP/IP SMF type 119 record provides a standardized structure for all SMF records that TCP/IP provides. This structure includes a standard representation of IP addresses across all type 119 records, where IPv4 addresses are included in IPv4-mapped form and IPv6 addresses are included as is.

Guideline: The type 119 records constitute a superset of the older type 118 records in terms of data that is available. IPv6 users should migrate to the SMF 119 record.

Type 118 FTP client and server transfer completion records are generated for IPv6 connections. In this case, the FTP records use IP addresses of 255.255.255.255 to indicate that the address cannot be included. All other type 118 SMF records are not generated for IPv6 connections.

For more information about SMF records, see z/OS Communications Server: IP Configuration Guide and z/OS Communications Server: IP Programmer's Guide and Reference.

IPv6 and the Policy Agent

The Policy Agent supports IPv6 in the following ways:

- Table 7 on page 58 lists the policy types that support IPv6.
- IPv6 XCF addresses can be specified in a sysplex distributor environment.

Table 7. IPv6 support for different policy types

Policy type	IPv6 supported?
AT-TLS	Yes
IDS	Yes
IPSec	Yes
QoS	Yes
Routing	Yes

When IPv6 addresses are used in policies for a particular stack, as configured to Policy Agent by using the TcpImage configuration statement, the stack must be IPv6 enabled. IPv6 policy is installed but is not enforceable in a stack that is not IPv6 enabled. If the corresponding stack is recycled later with IPv6 enabled, all policies are read and parsed again, and any policies with IPv6 addresses are enforced.

The use of IPv6 addresses in QoS policies is problematic because IPv6 interfaces can be assigned multiple IP addresses. As a result, the only way to specify IPv6 interfaces in policies is by the interface name that is specified on the INTERFACE statement. The interface name can also be used for IPv4 interfaces, as well as the IPv4 address. The name that is specified in the policies for IPv4 interfaces is the name that is specified on the LINK or INTERFACE statement in the TCP/IP profile. IPv6 interfaces can be specified for QoS policies and also for the SetSubnetPrioTosMask statement or LDAP object.

To support sysplex distributor policy performance monitoring, as specified by using the PolicyPerfMonitorForSDR configuration statement, the Policy Agent needs to establish TCP connections between the qosCollector threads that run on the distributing stacks and the qosListener threads that run on the target stacks. Depending on the sysplex configuration, either one or two connections between these threads are established. One connection is established for all target stacks that are configured for IPv4, and one connection is established for all target stacks that are configured for IPv6. Because a particular target can be configured for both IPv4 and IPv6, it is possible that two connections are established between a particular qosCollector and qosListener thread. When there are two connections, only information that is related to distributed IPv4 DVIPAs flows over the IPv4 connection, and likewise for the IPv6 connection.

IPv6 and SNMP

The following SNMP components operate over IPv6 networks and handle IPv6-related management data.

- SNMP agent
- z/OS UNIX **snmp/osnmp** command
- Trap Forwarder daemon
- Distributed Protocol Interface (DPI)
- TN3270 Telnet subagent

Requirement: The TCP/IP stack on your system must support IPv6 networking to take advantage of the IPv6 support offered by these components. If your system does not support IPv6 networking, then these applications operate in IPv4 mode.

The TCP/IP subagent supports IPv6 management data in the following MIB modules:

- IF-MIB from RFC 2233 - Interface data
- IP-MIB from RFC 4293 - IP and ICMP data
- IP-FORWARD-MIB from RFC 4292 - Route data
- TCP-MIB from RFC 4022 - TCP connection data
- UDP-MIB from RFC 4113 - UDP endpoint data
- TCP/IP Enterprise-specific MIB (IBMTCPIPMVS-MIB)

For more information about the TCP/IP subagent support, see z/OS Communications Server: IP System Administrator's Commands.

Monitoring the IP network

This topic describes how IPv6 affects reports.

IPv6 and Netstat

- To accommodate full IPv6 address information, Netstat reports were redesigned. If the TCP/IP stack is IPv6 enabled, reports are displayed in a different format than with IPv4. This different format might affect applications that parse Netstat output. The same considerations apply to applications that use IP addresses in their automation because IP addresses now have a longer size. If the TCP/IP stack is not IPv6 enabled, the report format is changed only when the FORMAT LONG parameter is specified on the Netstat command or on the IPCONFIG statement in the TCP/IP profile.
- IPv6 statistic information is added to the Netstat STATS/-S report.
- Information regarding whether the stack is IPv6 enabled is added to the Netstat UP/-u report.
- For a server that opens an AF_INET6 socket, binds to the IPv6 unspecified address (in6addr_any), and does a socketopt with IPv6_V6ONLY against the socket, the local address information in the connection-related reports contains the text `(IPV6_ONLY)`.

Netstat ALLCONN/-a example on an IPv6-enabled stack:

```
MVS TCP/IP NETSTAT CS V1R6       TCPIP NAME: TCPCS            17:40:36
User Id  Conn     State
-------  ----     -----
FTPABC1  00000021 Listen
  Local Socket:   0.0.0.0..21
  Foreign Socket: 0.0.0.0..0
FTPDV6   00000086 Listen
  Local Socket:   ::..21 (IPv6_ONLY)
  Foreign Socket: ::..0
```

For more detailed information, see z/OS Communications Server: IP System Administrator's Commands.

Control of output format

When the stack is IPv6 enabled, the report output is displayed in the new format, which is referred to as long format.

To allow the stack to be configured for IPv4-only operation (not IPv6 enabled and short format displays), but still allow a developer who needs to modify programs that rely on Netstat output to update and test new versions of these programs with long format output from Netstat, the following output format control options are available:

FORMAT SHORT
The output is displayed in the existing IPv4 format.

FORMAT LONG
The output is displayed in the format which supports IPv6 addresses.

A stack-wide output format parameter (FORMAT SHORT/LONG) can be specified on the IPCONFIG profile statement. It instructs Netstat to produce output in one of these formats. FORMAT SHORT is applicable only when the stack is not IPv6 enabled.

In addition to the stack-wide FORMAT parameter, a Netstat command-line option FORMAT/-M with keyword SHORT/LONG is supported to override the stack-wide parameter. When a user specifies the Netstat command-line format option, it overrides the stack-wide format parameter on an IPv4-only stack.

What is changed?

All Netstat reports were modified to support IPv6.

The following Netstat reports are added to display IPv6 information:

- Netstat ND/-n displays Neighbor Discovery cache information.
- Netstat DEFADDRT/-l displays the policy table for IPv6 default address selection.

Guideline: The Netstat GATE/-g report is not enhanced to support IPv6 routes. Netstat ROUTE/-r is the suggested alternative.

IPv6 and Ping and Traceroute

Ping and Traceroute provide the following support for IPv6:

- You can use IPv6 IP addresses, or host names that resolve to IPv6 IP addresses, for destinations. The IP address or host name can include scope information, which directs the Ping and Tracerte commands to use the specific outbound interface identified by the appended scope information. See "Support for scope information" on page 51 for guidelines about using this mechanism.
- You can use IPv6 IP addresses as the source IP address for the command's outbound packets.
- IPv6 IP addresses or interface names can be used as the outbound interface. This is analogous to specifying scope information as part of the destination IP address or host name.
- You can specify the new ADDRTYPE/-A command option to indicate whether an IPv4 or IPv6 IP address should be returned from host name resolution.
- IPv4-mapped IPv6 IP addresses are not supported for any option value.

Diagnosing problems with IPv6

This topic describes IPv6 problem diagnosis considerations.

IPv6 and IPCS

IPv6 is supported for IPCS formatting for TCPIPCS dump analysis and for all CTRACE components. For more information about IPCS, see z/OS Communications Server: IP Diagnosis Guide.

IPv6 and packet and data tracing

Packet trace, data trace, and OSA-Express Network Traffic Analyzer (OSAENTA) trace functions allow tracing of IPv6 addresses. For detailed information about trace functions, see z/OS Communications Server: IP Diagnosis Guide.

Chapter 5. Configuration guidelines

This topic describes IPv6 configuration guidelines and contains the following subtopics:

Connecting to an IPv6 network

z/OS Communications Server TCP/IP supports direct attachment to IPv6 networks in the following ways:

IPAQENET6 interface type
TCP/IP attaches to an IPv6 LAN by way of OSA-Express in QDIO mode, using either Fast Ethernet or Gigabit Ethernet. A single physical LAN can carry both IPv4 and IPv6 packets over the same media. While the physical network is shared, from a logical view there are two separate LANs, one carrying IPv4 traffic and one carrying IPv6 traffic. A single OSA-Express port can be used to carry both IPv4 and IPv6 traffic simultaneously. TCP/IP supports three CHPID types for IPAQENET6 (OSD, OSX, and OSM). If your configuration includes OSX or OSM CHPID types, see the information about TCP/IP in an ensemble in z/OS Communications Server: IP Configuration Guide for additional considerations for these CHPID types.

MPCPTP6 interface type
TCP/IP can directly communicate with other IPv6 z/OS Communications Server TCP/IP images, using ESCON channel-to-channel adapters, XCF connectivity (if the stacks are in the same sysplex), or the IUTSAMEH facility (if the stacks are on the same LPAR).

IPAQIDIO6 interface type
TCP/IP can directly communicate with other IPv6 z/OS Communications Server TCP/IP images and Linux for IBM Z® images using HiperSockets connectivity. This applies only to stacks running on the same central processor complex and running on IBM Z® that supports IPv6 HiperSockets.

IPCONFIG6 DYNAMICXCF
IPCONFIG6 DYNAMICXCF provides HiperSockets connectivity if available, XCF connectivity (if the stacks are in the same sysplex), or the IUTSAMEH facility (if the stacks are on the same LPAR).

Guideline: All of these interface types can be used for LPAR-to-LPAR IPv6 communication, best performance is achieved by using the IPAQIDIO6 interface type (if both stacks meet the criteria previously listed). The performance of the other interface types varies with the speed of the underlying media.

For stack-to-stack communications within a single LPAR, the MPCPTP6 interface type (using IUTSAMEH) provides the best performance.

To transport IPv6 traffic to another host, z/OS TCP/IP must send traffic using native IPv6 packets. Note that when communicating with another IPv6 host, a router within the network might tunnel the IPv6 packet across an IPv4 network to a remote IPv6 LAN or host. However, z/OS Communications Server TCP/IP cannot be the tunnel endpoint, and the tunneling by an intermediate router is transparent to z/OS Communications Server TCP/IP.

Assigning IPv6 addresses

When you are assigning IPv6 addresses, use the following guidelines.

Defining the interface ID for physical interfaces

If you do not manually configure the interface ID, the system selects an interface ID by using one of the following values:

- A random value on an MPCPTP6 interface
- A value that is derived from the MAC address on an IPAQENET6 interface
- A value that is derived from the IQD CHPID on an IPAQIDIO6 interface

The interface ID is used to form the link-local address for the interface. The interface ID is also appended to any manually configured prefixes for the interface, to form complete IPv6 addresses on the interface. If you do not configure manual IP addresses on the interface, the interface ID is appended to any prefixes that are learned over this interface by way of router advertisements, to form public IPv6 addresses on the interface.

To simplify the configuration effort, let the system select the interface ID. However, controlling all IPv6 addresses that are assigned to a physical adapter might be useful if other IPv6 nodes need to define static routes to this host, or if you use IPv6 addresses in multilevel security policies.

Use stateless address autoconfiguration for physical interfaces

IPv6 addresses for physical interfaces can be manually defined or can be automatically assigned by stateless address autoconfiguration. Use the stateless address autoconfiguration for this assignment. Using stateless address autoconfiguration reduces the amount of definition required to enable IPv6 support, while making future site renumbering easier.

Use VIPAs

Using static VIPAs removes hardware as a single point of failure for connections being routed over the failed hardware. If you are not using dynamic routing, configure at least one static VIPA for each LAN to which z/OS Communications Server TCP/IP is connected. Each VIPA configured this way should be associated with all physical adapters connected to that same LAN.

Requirement: Static VIPAs must be manually configured; z/OS Communications Server TCP/IP does not support stateless address autoconfiguration for VIPAs.

Dynamic VIPAs (DVIPAs) can also be used in an IPV6 network. The decision to use DVIPAs in an IPv6 network is similar to the decision to use DVIPAs in an IPv4 network. For detailed information, see z/OS Communications Server: IP Configuration Guide.

Selecting the network prefix

z/OS Communications Server TCP/IP does not perform duplicate address detection for VIPAs, because they are not assigned to a physical interface attached to the LAN.

Guideline: To avoid possible address collisions, the network prefix used for static VIPAs should be different from the network prefix used for physical interfaces (either manually configured or autoconfigured using stateless address autoconfiguration).

If either the IPv6 OSPF or IPv6 RIP dynamic routing protocol of OMPROUTE is being used, the network prefix for a static VIPA should not be the same as any prefix defined as on-link on a physical link. The VIPA can then be associated with interfaces attached to any physical link, thus enabling maximum redundancy. This association between VIPAs and interfaces attached to physical links is accomplished using the SOURCEVIPAINTERFACE parameter of the INTERFACE statement for the interface attached to the physical link.

If IPv6 OSPF or IPv6 RIP dynamic routing protocol of OMPROUTE is not being used, the network prefix for a static VIPA should be selected from the set of prefixes which are advertised by way of router discovery by one or more routers attached to the LAN. The prefix should be advertised as on-link and not to be used for address autoconfiguration. By using an on-link prefix, hosts and routers attached to the LAN use neighbor discovery address resolution to obtain a link-layer address for the VIPA. z/OS Communications Server TCP/IP selects a link-layer address of an attached physical interface when responding to the query, and the attached host or router forwards the packet to z/OS Communications Server TCP/IP. This eliminates the need to define static routes for VIPAs at hosts and routers attached to the same LAN as z/OS Communications Server TCP/IP. By using a prefix that is not being used for address autoconfiguration, the network prefix is not used by hosts for autoconfiguring addresses for physical interfaces.

Selecting the interface identifier

The VIPA interface identifier must be unique among all IP addresses that are created using the combination of network prefix and interface identifier. Any scheme can be used in generating the interface identifiers, as long as they are unique. By using a network prefix that is not used by stateless address autoconfiguration, it is necessary only to ensure the interface identifier is unique among all VIPAs that are sharing the same network prefix.

Effects of site renumbering on static VIPAs

When renumbering a site, new network prefixes are assigned to subnetworks. The existing network prefixes are marked as deprecated, during which time either the new prefixes or the old, deprecated prefixes can be used. After some time period, the deprecated network prefixes are deleted, along with all IPv6 addresses which use the network prefix.

For autoconfigured addresses, this process is automatically managed by stateless address autoconfiguration algorithms. For manually defined addresses, including all VIPAs, the process must be managed manually. When a prefix is to be deprecated, addresses that use the prefix should be deprecated using the INTERFACE DEPRADDR statement. After the prefix has expired, addresses that use the prefix should be deleted using the INTERFACE DELADDR statement.

Updating DNS definitions

This topic describes considerations for updating DNS definitions.

Including static VIPAs in DNS

Include static VIPAs in DNS, in both the forward and reverse zones. If VIPAs are used, it is unnecessary to include IPv6 addresses assigned to interfaces.

Requirement: IPv6 Enterprise Extender requires that host name resolution be used for the static VIPA. This host name resolution can be from a DNS or a local hosts file (/etc/ipnodes).

Defining IPv4-only host names and IPv4/IPv6 host names

In general, IPv6 connectivity between two hosts is preferred over IPv4 connectivity. In many cases, IPv4 is used only if one of the nodes does not support IPv6. This can lead to undesirable paths in the network being used for communication between two hosts. For instance, when a native IPv6 path does not exist, data can be tunneled over the IPv4 network, even when a native IPv4 path exists.

This can lead to longer connection establishment to an AF_INET application which resides on a dual-stack host. The client first attempts to connect using each IPv6 address defined for the dual-stack host before attempting to connect with IPv4. A well-behaved client cycles through all the addresses returned and ultimately, connects using IPv4. However, this takes both time and network resources to accomplish, and not all clients are well-behaved or bug-free.

To avoid undesirable tunneling and other potential problems, configure two host names in DNS. The existing host name can continue to be used for IPv4 connectivity to minimize disruption when connecting to unmodified AF_INET server applications. A new host name can also be defined, for which both IPv4 and IPv6 can be configured. When connecting by using the old host name, AF_INET6 clients connect by using IPv4. When connecting by using the new host name, AF_INET6 clients attempt to connect by using IPv6 and, if that fails, fall back and connect by using IPv4.

Using two host names allows the client to choose the network path that is taken. The client can route over IPv6 when the destination application is IPv6 enabled and a native IPv6 path exists, or take an IPv4 path.

The use of distinct host names for IPv4 and IPv4/IPv6 addresses is not strictly required. A single host name can be used to resolve to both IPv4 and IPv6 addresses. In addition, the use of distinct host names is necessary only during the initial transition phase when native IPv6 connectivity does not exist and applications have not yet been enabled for IPv6. After both of these occur, a single host name can be used.

Using source VIPA

Use a VIPA, either static or dynamic, as the source IP address on IPv6 hosts. When you use a VIPA, an IPv6 address can be resolved to a host name.

Procedure

- Define a VIPA to be used as the source IP address by using any of the following available configuration statements:
  - SOURCEVIPAINT parameter on the INTERFACE statement
  - TCPSTACKSOURCEVIPA parameter on the IPCONFIG6 statement
  - SRCIP statement

  For more information about choosing an appropriate method for specifying a source VIPA, see z/OS Communications Server: IP Configuration Guide.

Results

If you have also implemented the guidelines in “Updating DNS definitions” on page 63, an IPv6 address can be resolved to a host name.

Using dynamic or static routing to improve network selection

You can use the IPv6 OSPF or IPv6 RIP dynamic routing protocol provided by the OMPROUTE routing daemon to provide information about the IPv6 prefixes and hosts that can be accessed indirectly by way of adjacent routers. You can use IPv6 OSPF or IPv6 RIP, either alone or together with IPv6 router discovery, to provide complete routing information.

For routing considerations for interfaces that use the OSX CHPID type, see the information about OMPROUTE considerations for the intraensemble data network in z/OS Communications Server: IP Configuration Guide.

When both of the following statements are true, only default routes are available for accessing hosts that are not on directly attached links:

- Neither the IPv6 OSPF dynamic routing protocol nor the IPv6 RIP dynamic routing protocol of OMPROUTE is being used.
- Adjacent routers are not including indirect prefix routes (using the Route Information option as described in RFC 4191 *Default Router Preferences and More-Specific Routes*) in their router advertisement messages.

If the TCP/IP stack uses a non-optimal router when data is sent to one of these hosts, that router can send a redirect message that indicates a more optimal router for future use, as long as the more optimal router is on the same LAN as the original router.

When the TCP/IP stack is connected to multiple LANs, this processing might result in the following situations:

- A non-optimal router is used
- A router is used that cannot reach the final destination

For example, if the stack selects a router on one LAN, but the optimal router is on another LAN, the router on the first LAN cannot redirect the stack to the router that is on the second LAN. In this case, configure a static route so that the stack can initially select the optimal network path.

Guidelines: When you are defining static routes, use the following guidelines:

- Use subnet routes instead of host routes

  Remote IP addresses are difficult to predict. When using extensions to stateless address autoconfiguration, some clients can change their IP addresses on a routine basis, such as once an hour or once a day. In addition, these addresses can be created using cryptographic algorithms, making it difficult to impossible to predict which IP address a client might use. Defining static host routes to be used when communicating with such a client is equally as difficult or impossible.

  Instead of defining a host route, define subnet routes. The network prefixes used in generating IPv6 addresses are much more stable than the interface identifiers used by hosts, typically changing only when a site is renumbered.
- Use the link-local address of gateway router

  When you are defining the gateway router for a static route, use the link-local address for the router. Link-local addresses do not change as the result of site renumbering, which minimizes potential updates to the static routes. This is required in order to honor and process an ICMPv6 redirect message.
- Effects of site renumbering on static routes

  When a remote site is renumbered, new network prefixes are defined for the remote site and the old network prefixes are deprecated. After a time period, the old network prefixes are deleted.

  A static route to a remote subnet should be created when a prefix is defined and should remain as long as the prefix is either preferred or deprecated. Only when the remote prefix is deleted should the static route be deleted.

Connecting to non-local IPv4 locations

If native IPv6 connectivity does not exist between two IPv6 sites, IPv6 over IPv4 tunneling can be used to provide IPv6 connectivity to the two sites. z/OS Communications Server TCP/IP can make use of an IPv6 over IPv4 tunnel to send packets to a remote site, but cannot be used as a tunnel endpoint itself. Instead, an intermediate router which supports IPv6 over IPv4 tunneling must act as the tunnel endpoint.

See "Enabling IPv6 communication between IPv6 nodes or networks in an IPv4 environment" on page 41 for more information about IPv6 over IPv4 tunnels.

IPv6-only application access to IPv4-only application

When an IPv6-only application needs to communicate with an IPv4-only host or application, some form of IPv6-to-IPv4 translation or application-layer gateway must occur. If needed, an outboard protocol translator or application-layer gateway component must be used, as z/OS Communications Server TCP/IP does not include such support. There are various technologies which can be used, such as NAT-PT or SOCKS64. See "Application layer gateways and protocol translation" on page 43 for more information.

Chapter 6. API support

This topic describes API support and contains the following subtopics:

- "UNIX socket APIs" on page 68
- "Native TCP/IP socket APIs" on page 68

z/OS provides a versatile and diverse set of socket API libraries to support the various z/OS application environments. Figure 14 on page 67 illustrates the relationship of the various z/OS socket APIs and the level of IPv6 present for each API.

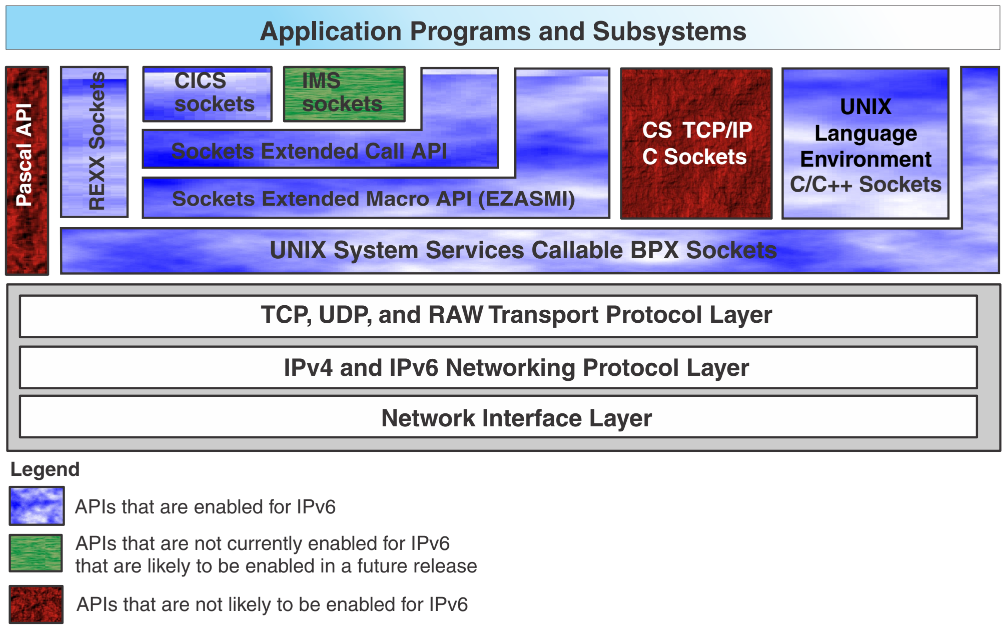

Application Programs and Subsystems
Pascal API
REXX Sockets
CICS sockets
IMS sockets
Sockets Extended Call API
Sockets Extended Macro API (EZASMI)
CS TCP/IP C Sockets
UNIX Language Environment C/C++ Sockets
UNIX System Services Callable BPX Sockets
TCP, UDP, and RAW Transport Protocol Layer
IPv4 and IPv6 Networking Protocol Layer
Network Interface Layer
Legend
APIs that are enabled for IPv6
APIs that are not currently enabled for IPv6 that are likely to be enabled in a future release
APIs that are not likely to be enabled for IPv6

Figure 14. z/OS socket APIs

The following environments are the two main socket API execution environments in z/OS:

- UNIX [implemented by UNIX System Services (Language Environment®)]
- Native TCP/IP (implemented by TCP/IP in z/OS Communications Server)

There are several higher level C/C++ APIs that rely on the TCP/IP sockets for communications over an IP network, including the following APIs:

- Resource Reservation Setup Protocol API (RAPI)
- Sun and NCS Remote Procedure Call (RPC)
- X Window System and Motif
- X/Open Transport Interface (XTI)

These APIs do not support IPv6 communications.

Guideline: CICS® programs written to use the IP CICS C Sockets API must use the TCP/IP C headers. Include the following definition to expose the required IPv6 structures, macros, and definitions in the header files:

```
#define __CICS_IPV6
```

See z/OS Communications Server: IP CICS Sockets Guide for information about using the IP CICS C Sockets API.

UNIX socket APIs

The UNIX socket APIs that support both IPv4 and IPv6 communications are z/OS UNIX Assembler Callable Services and z/OS C sockets.

z/OS UNIX Assembler Callable Services

z/OS UNIX Assembler Callable Services is a generalized call-based interface to z/OS UNIX IP sockets programming. This API supports both IPv4 and IPv6 communications. It includes support for the basic IPv6 API features and for a subset of the advanced IPv6 API features. For more information, see z/OS UNIX System Services Programming: Assembler Callable Services Reference.

z/OS C sockets

z/OS UNIX C sockets is used in the z/OS UNIX environment. Programmers use this API to create applications that conform to the POSIX or XPG4 standard (a UNIX specification). This API supports both IPv4 and IPv6 communications. It includes support for the basic IPv6 API features and for a subset of the advanced IPv6 API features. For more information about this API, see z/OS XL C/C++ Runtime Library Reference.

Native TCP/IP socket APIs

The following TCP/IP Services APIs are included in this library.

- Sockets extended macro API
- Sockets extended call instruction API
- REXX sockets
- CICS sockets
- IMS sockets
- Pascal API
- TCP/IP C/C++ Sockets

For more information about these APIs, excluding CICS, see z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference.

Sockets Extended macro API

The Sockets Extended macro API is a generalized assembler macro-based interface to IP socket programming. It includes support for IPv4 and for the basic IPv6 socket API functions.

Sockets Extended Call Instruction API

The Sockets Extended Call Instruction API is a generalized call-based interface to IP sockets programming. It includes support for IPv4 and for the basic IPv6 socket API functions.

REXX sockets

The REXX sockets programming interface implements facilities for IP socket communication directly from REXX programs by way of an address rxsocket function. It includes support for IPv4 and for the basic IPv6 socket API functions.

CICS sockets

Using the CICS socket interface, you can write CICS applications that act as clients or servers in a TCP/IP-based network. You can write applications in C language, using the C sockets programming interface, or in COBOL, PL/I, or assembler, using the Extended Sockets programming interface. This API supports TCP/IP communications over IPv4 and basic IPv6 socket API functions. For more information, see z/OS Communications Server: IP CICS Sockets Guide.

IMS sockets

The Information Management System (IMS) socket interface supports development of client/server applications in which one part of the application executes on a TCP/IP-connected host and the other part runs as an IMS application program. The programming interface used by both application parts is the socket programming interface. This API currently supports TCP/IP communications over IPv4 only, but will probably support IPv6 communications in a future release. For more information, see z/OS Communications Server: IP IMS Sockets Guide.

Pascal API

The Pascal socket application programming interface enables you to develop TCP/IP applications in the Pascal language. It supports only TCP/IP communications over IPv4. It is unlikely that this API will be enhanced to support IPv6 in the future. You are encouraged to migrate applications that use this API to one of the other socket APIs that are IPv6 enabled.

TCP/IP C/C++ sockets

The C/C++ socket interface supports IPv4 socket function calls that can be invoked from C/C++ programs. This API is similar to the UNIX C socket API that is the recommended socket API for C/C++ application development on z/OS. The TCP/IP C/C++ sockets API will not be enhanced for IPv6 support. Consider migrating existing applications that will be enabled for IPv6 to the UNIX C socket API.

Chapter 7. Basic socket API extensions for IPv6

IPv4 addresses are 32 bits long, but IPv6 interfaces are identified by 128-bit addresses. The socket interface makes the size of an IP address visible to an application; virtually all TCP/IP applications using sockets have knowledge of the size of an IP address. Those parts of the API that expose the addresses must be changed to accommodate the larger IPv6 address size. IPv6 also introduces new features, some of which must be made visible to applications by way of the API.

This topic describes the basic extensions to the socket interface and new features of IPv6 as described in the Internet Engineering Task Force (IETF) RFC 3493, *Basic Socket Interface Extensions for IPv6.*, and contains the following topics:

Note: All examples in this topic are shown using UNIX Language Environment C; see z/OS XL C/C++ Runtime Library Reference for details.

Design considerations

The two main programming tasks associated with IPv6 exploitation involve migrating existing application programs to support IPv6 and designing new programs for IPv6. In both cases, the changed or new code should be designed so that it is capable of using IPv4 or IPv6 addresses. Servers should be designed so that they can communicate with both IPv4 and IPv6 clients. Existing IPv4 client and server programs should continue to operate properly as long as only IPv4 connectivity is required between clients and servers.

The following topics describe key differences between IPv4 and IPv6.

Requirement: You must have a basic knowledge of IPv4 socket programming for clients and servers.

Protocol families

IPv4 socket applications use a AF_INET (equivalent to PF_INET) protocol family. For IPv6, a new protocol family of AF_INET6 (equivalent to PF_INET6) has been defined. The protocol family is the first parameter to the socket() function that is used to obtain a socket descriptor. For most applications, an AF_INET6 socket can be used to communicate with IPv4 and IPv6 clients.

Address families

Most socket functions require a socket descriptor and a generic socket address structure called a sockaddr. The exact format of the sockaddr structure depends on the address family. For IPv4 sockets, the sockaddr structure is sockaddr_in. For IPv6, the sockaddr structure sockaddr_in6 is used.

The following socket functions have a sockaddr structure as one of their parameters:

```
bind()
connect()
sendmsg()
sendto()
accept()
recvfrom()
recvmsg()
getpeername()
```

```
getsockname()
```

The sockaddr structure that is used in these functions must be the proper structure for the socket family.

For IPv4 (AF_INET), the sockaddr (sockaddr_in) contains the information shown in Table 8 on page 72.

Table 8. sockaddr format for AF_INET

Description	**Length**	**Contents**
sockaddr length	1 byte	Not used, should be set to 0
family	1 byte	AF_INET
port	2 bytes	TCP or UDP port number
IP address	4 bytes	IPv4 IP address
reserved bytes	8 bytes	Not used

For IPv6 (AF_INET6), the sockaddr (sockaddr_in6) contains additional information. Also, note that the IP address for IPv6 is 16 bytes long instead of 4 bytes long as in IPv4.

Table 9. sockaddr format for AF_INET6

Description	**Length**	**Contents**
sockaddr length	1 byte	Not used, should be set to 0
family	1 byte	AF_INET6
port	2 bytes	TCP or UDP port number (same as v4)
flowinfo	4 bytes	Flow information
IP address	16 bytes	IPv6 IP address
scope ID	4 bytes	Used to determine IP address scope

Special IP addresses

Like IPv4, IPv6 also defines loopback and wildcard (INADDR_ANY) addresses. The differences are shown in Table 10 on page 72.

Table 10. Special IP addresses

	IPv4	**IPv6**
Loopback address	127.0.0.1	::1 (15 bytes of zeros, 1 byte of 1)
Wildcard address	0.0.0.0	:: (16 bytes of zeros)
Multicast address	224.0.0.1 - 239.255.255.255	See "Multicast IPv6 addresses" on page 13

Name and address resolution functions

IPv6 introduces new APIs for the resolver function. These APIs allow applications to resolve host names to IP addresses and IP addresses to host names. The primary new APIs are getaddrinfo, getnameinfo, and freeaddrinfo. The APIs are designed to work with both IPv4 and IPv6 addressing. Consider use of these new APIs if an application is being designed for eventual use in an IPv6 environment.

How host name (getaddrinfo) or IP address (getnameinfo) resolution is done depends on the resolver specifications that are contained in the resolver setup files and TCPIP.DATA configuration files. These specifications determine whether the APIs query a name server first and then search the local host tables, whether the order is reversed, or even if one of the steps is eliminated. If the APIs have to search local host tables, the specifications also control which tables are accessed. For more information about resolver setup, see "Resolver configuration" on page 55.

Protocol-independent node name and service name translation

The getaddrinfo function is considered a replacement for the existing gethostbyname and getservbyname APIs. The getaddrinfo function takes an input host name, an input service name, or both, and returns one or more addrinfo structures upon successful resolution. The getaddrinfo function also accepts a host name or service name in numerical form as input, and returns the same value in presentation form by using the addrinfo structure. An addrinfo structure contains the following output information:

- A pointer to a sockaddr_in or sockaddr_in6 structure containing an IP address and service port. For IPv6 link-local addresses, the sockaddr_in6 structure might contain the zone index, if scope information was provided as part of the input host name. See "Scope information about getaddrinfo calls" on page 77 for details.
- Length of sockaddr structure and family type (AF_INET, AF_INET6) of the sockaddr structure
- Socktype and protocol values usable with this sockaddr structure
- Pointer to canonical name associated with the input host name (applicable only in the first addrinfo structure)
- Pointer to next addrinfo structure (set to 0 in the last element of the chain)

The storage for the addrinfo structures is allocated by the resolver from the application's address space, and the application should use the freeaddrinfo API to release the addrinfo structures when the information is no longer required. The application should not manipulate the chain of addrinfo structures returned by way of getaddrinfo, but rather the application should simply return the entire chain, as received, to the resolver by way of freeaddrinfo.

In addition to hostname or servicename, one of which must be present on a valid getaddrinfo invocation, the application can specify additional input to the resolver on the getaddrinfo invocation. This input is optional, and if specified, is passed by way of an input addrinfo structure. The input settings include the following possibilities:

- Family type of sockaddr structure required on output.
- Socktype and protocol values for which the returned IP address and port number must work. This would be used primarily for cases where a service name was being resolved, as might typically have been done previously by way of getservbyname.
- Various input flag settings include the following settings:
  - AI_ADDRCONFIG
  - AI_ALL
  - AI_CANONNAME
  - AI_NUMERICHOST
  - AI_NUMERICSERV
  - AI_PASSIVE
  - AI_V4MAPPED

If no specific input from the application is provided, the resolver assumes that any sockaddr type (that is, both IPv4 and IPv6 addresses) is acceptable as output. Thus, by default, the resolver searches for both IPv6 and IPv4 addresses by way of DNS or by way of local host files (such as /etc/hosts). This searching might not always be the best choice for the application that issues getaddrinfo. By using the input fields, an application that issues getaddrinfo() can influence the processing that the resolver function provides for that request in the following ways:

- The application can specify that the sockaddr returned by getaddrinfo should be of family type AF_INET, AF_INET6 or AF_UNSPEC (meaning either family type would be acceptable). For example, if AF_INET is specified, the resolver does not perform any searches for IPv6 addresses for *hostname*, because the output requested must be an IPv4 address.

- The application can specify that the following addresses are returned:
  - Both IPv6 and IPv4 addresses should be returned
  - IPv4 should be returned only if there are no IPv6 addresses resolved
  - Only IPv6 addresses should be returned
  - Only IPv4 addresses should be returned.

  This information, indicated by the input combination of family type and the AI_ALL and AI_V4MAPPED flags, to a large extent controls the types of searches performed by the Resolver during the course of the processing.
- The application can specify that IPv6 addresses should be returned only when the system has IPv6 interfaces defined and can specify that IPv4 addresses should be returned only when IPv4 interfaces are defined. This preference, indicated by way of the AI_ADDRCONFIG flag, allows the application to eliminate resolution searches looking for addresses that cannot be used by the application.
- The application can specify whether the sockaddr returned should contain an address for passive (that is, the INADDR_ANY address) or active (that is, the loopback address) socket activation. This choice is indicated by way of the AI_PASSIVE flag, and is applicable only in the absence of an input *hostname* value.
- The application can specify that only translation from presentation to numeric format should be performed for hostname, or service name, or both. This option is indicated by setting the AI_NUMERICHOST flag (for *hostname*) or the AI_NUMERICSERV (for *servicename*) flag, which indicates that the associated input value must be in numeric format or the getaddrinfo request should be failed.
- The application can specify that only a given socktype or protocol value should be used for looking up the port number associated with the input servicename, or can request that all valid socktype types and protocols (TCP and UDP) be used for the getservbyname processing. This preference is indicated by way of the socktype and protocol settings.

With such a flexible interface, the application programmer must decide what inputs are reasonable for the capabilities of the application being created or modified.

Table 11 on page 74 shows the two most likely application usages and the suggested getaddrinfo input settings that coincide with that functionality:

- IPv6-capable when the underlying system is IPv6 capable
- IPv4-capable only

Table 11. Getaddrinfo application capabilities 1

Application capabilities	**Sockaddr family to request**	**Additional flags to set**	**Expected outputs**
(IPv4 only) Application is pure IPv4 and cannot handle any IPv6 addresses.	AF_INET	AI_ADDRCONFIG	Getaddrinfo returns one or more addrinfo structures, each pointing to an IPv4 address saved in an AF_INET sockaddr. No addrinfo structures are returned if there are no IPv4 interfaces defined on the system. No searches of any kind are performed for IPv6 addresses as part of this request.

Table 11. Getaddrinfo application capabilities 1 (continued)

Application capabilities	**Sockaddr family to request**	**Additional flags to set**	**Expected outputs**
(IPv6 capable) Application wants all known addresses for hostname, in IPv6 format when the system supports IPv6, or in IPv4 format otherwise.	AF_UNSPEC	One of the following groups: • AI_ADDRCONFIG • AI_ADDRCONFIG and AI_V4MAPPED	Getaddrinfo returns one or more addrinfo structures, each pointing to a sockaddr structure. The sockaddrs consists of one of the following sets: • All AF_INET6 sockaddrs, containing IPv6 or mapped IPv4 addresses, if the system supports IPv6 processing (only when AI_V4MAPPED coded). • AF_INET6 sockaddrs, containing IPv6 addresses, and AF_INET sockaddrs, containing IPv4 addresses, if the system supports IPv6 processing (only when AI_V4MAPPED is NOT coded). • All AF_INET sockaddrs, containing IPv4 addresses, if the system does not support IPv6 processing. In all cases, the IPv6 addresses are returned only if there is an IPv6 interface defined on the system, and the IPv4 addresses are returned only if there is an IPv4 interface defined.

An application with no interest in using IPv6 wants to use the first entry in Table 11 on page 74; otherwise, if there is some interest in using IPv6 functionality, an application would achieve the greatest flexibility by using the second table entry. Using the IPv6 entry approach, the application places the burden of supplying a workable sockaddr structure on the Resolver logic. If IPv6 is supported on the system, the Resolver endeavors to return AF_INET6 sockaddrs to the application; otherwise, the Resolver returns AF_INET sockaddrs to the application. The choice of coding or not coding AI_V4MAPPED in this situation depends on the application's preference regarding receiving AF_INET6 sockaddrs: the more the application wants to deal exclusively with AF_INET6 sockaddrs, the more reason to code AI_V4MAPPED.

Table 11 on page 74 should be sufficient for most application usages. However, there are other likely application capability models possible, and Table 12 on page 75 provides some guidance on how to code the Getaddrinfo invocations for those applications.

Table 12. Getaddrinfo application capabilities 2

Application capabilities	**Sockaddr family to request**	**Additional flags to set**	**Expected outputs**
Application is pure IPv6 and cannot handle any mapped IPv4 addresses.	AF_INET6	AI_ADDRCONFIG	Getaddrinfo returns one or more addrinfo structures, each pointing to an IPv6 address saved in an AF_INET6 sockaddr. No addrinfo structure is returned if there are no IPv6 interfaces defined on the system. No searches of any kind are performed for IPv4 addresses as part of this request.

Table 12. Getaddrinfo application capabilities 2 (continued)

Application capabilities	Sockaddr family to request	Additional flags to set	Expected outputs
Application prefers IPv6 addresses, requires IPv6 address format, but can handle mapped IPv4 addresses if necessary.	AF_INET6	AI_ADDRCONFIG, AI_V4MAPPED	Getaddrinfo returns one or more addrinfo structures, each pointing to an AF_INET6 sockaddr. The addresses in the sockaddrs structure consist of one of the following sets: • All IPv6 addresses, if there is an IPv6 interface defined on the system and IPv6 addresses exist for hostname • All mapped IPv4 addresses, if there were no IPv6 addresses to be returned for hostname and there was an IPv4 interface defined for the system
Application wants all known addresses for hostname, in IPv6 format.	AF_INET6	AI_ADDRCONFIG, AI_V4MAPPED, AI_ALL	Getaddrinfo returns one or more addrinfo structures, each pointing to an AF_INET6 sockaddr. The addresses within the sockaddrs consist of all IPv6 addresses, if there is an IPv6 interface defined on the system, and mapped IPv4 addresses, if there is an IPv4 interface defined for the system, associated with hostname.
Application wants all known addresses for hostname, in native (IPv6 or IPv4) format.	AF_UNSPEC	One of the following flag groups: • AI_ADDRCONFIG and AI_ALL • AI_ADDRCONFIG	Getaddrinfo returns one or more addrinfo structures, each pointing to a sockaddr structure. The sockaddr structures are a mixture of AF_INET6 sockaddrs (each containing an IPv6 address) and AF_INET sockaddrs (each containing an IPv4 address). The IPv6 addresses are returned only if there is an IPv6 interface defined on the system, and the IPv4 addresses are returned only if there was an IPv4 interface defined for the system.
Application wants all known addresses for hostname, in IPv6 format when the system supports IPv6, or in IPv4 format otherwise.	AF_UNSPEC	One of the following flag groups: • AI_V4MAPPED and AI_ALL • AI_V4MAPPED	Getaddrinfo returns one or more addrinfo structures, each pointing to a sockaddr structure. The sockaddrs consists of one of the following sets: • All AF_INET6 sockaddrs, containing IPv6 or mapped IPv4 addresses, if the system supports IPv6 processing. • AF_INET6 sockaddrs, containing IPv6 addresses, and AF_INET sockaddrs, containing IPv4 addresses, if the system doesn't support IPv6 processing. The actual availability of IPv6 or IPv4 interfaces on the system is not taken into consideration.

Table 12. Getaddrinfo application capabilities 2 (continued)

Application capabilities	**Sockaddr family to request**	**Additional flags to set**	**Expected outputs**
Application wants all known addresses for hostname, regardless of system connectivity, in native format.	AF_UNSPEC	One of the following flag groups: • AI_ALL • NONE	Getaddrinfo returns one or more addrinfo structures, each pointing to a sockaddr structure. The sockaddr structures can be a mixture of AF_INET6 sockaddrs (each containing an IPv6 address) and AF_INET sockaddrs (each containing an IPv4 address), depending on the address resolution. The actual availability of IPv6 or IPv4 interfaces on the system is not taken into consideration. **Note:** These are the default settings independent of IPv6 enablement on the system.

Regardless of the application model in use, and because output from getaddrinfo can be a chain of addrinfo structures, the application should attempt to use each address, in the order received, to open a socket and connect or send a datagram to the target host name until it is successful, versus simply using the first address and stopping if a failure is encountered.

The application is now responsible for freeing the storage (addrinfo and sockaddr structures, and so on) associated with the new resolver APIs. The new freeaddrinfo API should be used to free this storage. If the application neglects to perform this step, the resolver cleans up the storage when the process ends, but storage constraints might occur before termination if a large number of getaddrinfo APIs are performed.

Scope information about getaddrinfo calls

The getaddrinfo process accepts scope information as part of the input host name. Scope information is defined as an interface name or the interface index that uniquely identifies a specific interface to be used with a link-local IPv6 address (see "Interface identification" on page 80 for information about interface indexes). An application might need to pass scope information to the resolver so that the resulting sockaddr_in6 structures have the appropriate zone index value set by the resolver. The zone index is determined using the if_nametoindex() function if the input scope information is an interface name, or it is determined by converting the input interface index value into binary form.

Scope information is provided in the format *hostname%scopeinformation*, where the scope information can be the interface name or an interface index. The combined *hostname%scopeinformation* cannot exceed 255 characters in length; if the information is longer, the request fails.

Rules: When getaddrinfo processes scope information the following rules apply:

- Scope information can be present only in the following cases:
  - The host name portion of the input is not null (for example, input that is not in the form *%scopeinformation*)
  - If a numeric form of host name is specified, the numeric form must represent an IPv6 address
- If scope information is specified as an interface name, the interface name must resolve to a zone index using the if_nametoindex() function.
- If scope information is specified as an interface index, the index must be valid for this system.

If any of these verification steps fail, the getaddrinfo request fails.

Zone indexes apply only to link-local IPv6 addresses in z/OS Communications Server. If the input host name specified by the application does not resolve to a link-local IPv6 address, any scope information provided as part of the host name is ignored.

See "Support for scope information" on page 51 for more general information about scope information in the z/OS Communications Server environment.

Socket address structure to host name and service name

The getnameinfo call is a replacement for the existing gethostbyaddr and getservbyport APIs. The getnameinfo call takes an input IP address, an input port number, or both, and returns (when resolution is successful) the hostname or the service location. These parameters are passed in a sockaddr structure that also contains the address family.

For input link-local IPv6 addresses, the zone index value in the sockaddr structure is also used as an input by getnameinfo processing. The zone index value in this instance is returned as scope information that is appended to the output host name, using the format *hostname%scopeinformation*. The form of the scope information can be the numeric form of the zone index value or the interface name associated with the zone index value, which is identified using the if_indextoname() function (see "Interface identification" on page 80 for details). The format of the scope information returned to the application as part of the hostname is determined by the flag, NI_NUMERICSCOPE, on the getnameinfo() call. The total length of the combined host name and scope information must be able to fit within the buffer passed by the application (up to a maximum buffer size of 255 characters in length), or the value is truncated to fit within the buffer.

In addition to IP address or port number, one of which must be present on a valid getnameinfo invocation, the application can specify more input to the Resolver on the getnameinfo invocation. This input is optional. The input settings include the following settings (various input flag settings can be specified):

NI_NOFQDN
Specifies that only the host name portion of the fully qualified domain name (FQDN) is returned for local hosts.

NI_NUMERICHOST
Specifies that the numeric form of the host name, its IP address, is returned instead of its name. No resolution takes place for the specified input if the NI_NUMERICHOST flag is on.

NI_NUMERICSERV
Specifies that the numeric form of the service name, the port number, is returned instead of the service name. No resolution takes place for the specified input if the NI_NUMERICSERV flag is on.

NI_NAMEREQD
Specifies that an error is returned if the host name cannot be located. (If NI_NAMEREQD is not specified, the numeric form of the host name, the IP address, is returned).

NI_DGRAM
Specifies that the service is a datagram service (SOCK_DGRAM). The default behavior assumes that the service is a stream service.

NI_NUMERICSCOPE
Specifies that the numeric form of the scope information, its interface index, appended to the host name, is returned instead of the interface name. If the input IP address was not a link-local address, or if the application did not request that the host name be returned as output, scope information is not returned, and the setting of NI_NUMERICSCOPE is ignored. If NI_NUMERICSCOPE is not specified, the default is to return the interface name when scope information is appended to the host name.

Address conversion functions

IP addresses often need to be given to a socket application in character (string) format. It is also common for socket applications to need to display IP addresses in string format. The following functions work for IPv4 and IPv6 addresses:

inet_ntop
Convert a binary IP address (either IPv4 or IPv6) into string format.

inet_pton
Convert an IP address in string format to binary format.

The functions inet_ntoa and inet_addr are still available, but they cannot be used for IPv6 addresses.

Table 13. Address conversion functions

Function	z/OS UNIX Assembler Callable services	C/C++ using Language Environment	IP CICS C sockets	REXX	Socket Extended macro/call (includes CICS EZASOKET)
inet_pton	No	Yes	Yes	No	No
inet_ntop	No	Yes	Yes	No	No
PTON	No	No	No	No	Yes
NTOP	No	No	No	No	Yes

Address testing macros

The macros listed in Table 14 on page 79 can be used to test for special IPv6 addresses.

Table 14. Address testing macros

Macros	Assembler Callable services	C/C++ using Language Environment	IP CICS C sockets	REXX	Socket Extended macro/call (includes CICS EZASOKET)
IN6_IS_ADDR_UNSPECIFIED	No	Yes	Yes	No	No
IN6_IS_ADDR_LOOPBACK	No	Yes	Yes	No	No
IN6_IS_ADDR_MULTICAST	No	Yes	Yes	No	No
IN6_IS_ADDR_LINKLOCAL	No	Yes	Yes	No	No
IN6_IS_ADDR_V4MAPPED	No	Yes	Yes	No	No
IN6_IS_ADDR_V4COMPAT	No	Yes	Yes	No	No
IN6_IS_ADDR_MC_NODELOCAL	No	Yes	Yes	No	No
IN6_IS_ADDR_MC_LINKLOCAL	No	Yes	Yes	No	No
IN6_IS_ADDR_MC_SITELOCAL	No	Yes	Yes	No	No
IN6_IS_ADDR_MC_ORGLOCAL	No	Yes	Yes	No	No
IN6_IS_ADDR_MC_GLOBAL	No	Yes	Yes	No	No

The macros function in the following ways:

- The first six macros return true if the address is of the specified type, or false otherwise.
- The last five macros test the scope of a multicast address and return true if the address is a multicast address of the specified scope, or false if the address is either not a multicast address or not of the specified scope.
- IN6_IS_ADDR_LINKLOCAL returns true only for IPv6 link-local unicast addresses, and therefore the IN6_IS_ADDR_LINKLOCAL macro returns false for the IPv6 loopback address (::1). This macro does not return true for IPv6 multicast addresses of link-local scope.

Interface identification

IPv6 interfaces can have many different IP addresses. IPv6 allows a socket application to specify an interface to use for sending data by specifying an interface index. Certain socket options allow specification of an interface index. Also, socket options for IPv6 multicast join group and IPv6 multicast leave group allow optional specification of an interface index.

The IPv6 resolver interface enables a socket application to specify interface index or interface name on getaddrinfo calls to initialize the zone index field in the sockaddr structure information for link-local IPv6 addresses. The getnameinfo calls return the interface index or interface name for input link-local IPv6 addresses when the sockaddr structure contains the zone index. See "Scope information about getaddrinfo calls" on page 77 for more information. Some z/OS applications use this resolver capability to enable users to include interface (or scope) information as part of host name or IPv6 address information passed to the resolver. See Table 15 on page 80 for a list of the applications that support for this function.

The function if_nameindex() allows socket applications to obtain a list of interface names and their corresponding indexes. The functions if_nametoindex() and if_indextoname() allow translation of an interface name to an interface index and translation of an interface index to an interface name. The function if_freenameindex() is used to free dynamic storage allocated by the if_nameindex() function.

For non-C/C++ (Language Environment applications), a new ioctl function code (SIOCGIFNAMEINDEX) is provided. Use Table 15 on page 80 to determine which APIs support this new ioctl.

Table 15. Function calls

Function/IOCTL	**z/OS UNIX Assembler Callable services**	**C/C++ using Language Environment**	**IP CICS C sockets**	**REXX**	**Socket Extended macro/call (includes CICS EZASOKET)**
if_nametoindex	No	Yes	Yes	No	No
if_indextoname	No	Yes	Yes	No	No
if_nameindex	No	Yes	Yes	No	No
SIOCGIFNAMEINDEX	Yes	No	No	Yes	Yes
if_freenameindex	No	Yes	Yes	No	No

Socket options to support IPv6

A group of socket options is defined to support IPv6. These options are defined with a level of IPPROTO_IPV6. The individual options begin with IPV6_ or with MCAST_.

Restriction: The options that begin with IPV6_ are allowed only on AF_INET6 sockets.

In most cases, an IPV6_*xxx* option can be set on an AF_INET6 socket that is using IPv4-mapped IPv6 addresses but have no effect. For example, the IPV6_UNICAST_HOPS socket option is used to set a hop limit value in the IPv6 header. Because IPv4 packets are used with IPv4-mapped IPv6 addresses, the hop limit value is not used.

Guideline: The Sockets Extended macro/call APIs do not use level as an input to getsockopt() and setsockopt(). However, other IPv6-enabled APIs do use level as input. For detailed information about setsockopt() and getsockopt() input and output, see the API-specific information.

Table 16. Socket options for getsockopt() and setsockopt()

Socket options getsockopt() setsockopt()	z/OS UNIX Assembler Callable services	C/C++ using Language Environment	IP CICS C sockets	REXX	Sockets Extended macro/call (includes CICS EZASOKET)
IPV6_ADDR_PREFERENCES	Yes	Yes	Yes	Yes	Yes
IPV6_UNICAST_HOPS	Yes	Yes	Yes	Yes	Yes
IPV6_MULTICAST_IF	Yes	Yes	Yes	Yes	Yes
IPV6_MULTICAST_LOOP	Yes	Yes	Yes	Yes	Yes
IPV6_MULTICAST_HOPS	Yes	Yes	Yes	Yes	Yes
IPV6_JOIN_GROUP	Yes	Yes	Yes	Yes	Yes
IPV6_LEAVE_GROUP	Yes	Yes	Yes	Yes	Yes
IPV6_V6ONLY	Yes	Yes	Yes	Yes	Yes
MCAST_BLOCK_SOURCE	Yes	Yes	Yes	Yes	Yes
MCAST_JOIN_GROUP	Yes	Yes	Yes	Yes	Yes
MCAST_JOIN_SOURCE_GROUP	Yes	Yes	Yes	Yes	Yes
MCAST_LEAVE_GROUP	Yes	Yes	Yes	Yes	Yes
MCAST_LEAVE_SOURCE_GROUP	Yes	Yes	Yes	Yes	Yes
MCAST_UNBLOCK_SOURCE	Yes	Yes	Yes	Yes	Yes

Option to control sending of unicast packets

Use the following option to control sending of unicast packets:

IPV6_UNICAST_HOPS

The IPv6 header contains a hop limit field that controls the number of hops over which a datagram can be sent before being discarded. This is similar to the TTL field in the IPv4 header. The IPV6_UNICAST_HOPS socket option can be used to set the default hop limit value for an outgoing unicast packet. The socket option value should be between 0 and 255 inclusive. A socket option value of -1 is used to clear the socket option. This causes the stack default to be used.

A getsockopt() with this option returns the value set by a setsockopt(). If a setsockopt() has not been performed, the stack's default value is returned.

The HOPLIMIT parameter on the IPCONFIG6 statement influences the default hop limit when this socket option is not set. An application must be APF-authorized or have superuser authority to set this option to a value greater than the value of HOPLIMIT on the IPCONFIG6 statement. See z/OS Communications Server: IP Configuration Guide for more information about the IPCONFIG6 statement.

Tip: This function is similar to the IPv4 socket option IP_TTL.

Options to control sending of multicast packets

These options allow an application to control certain features in the transmission of IPv6 multicast packets. These socket options do not have to be set to send multicast packets. Supplying a multicast address as the destination address is the only thing required to send an IPv6 multicast packet.

IPV6_MULTICAST_IF

This socket option allows an application to control the outgoing interface used for a multicast packet. The socket option value is the interface index of the interface to be used.

A getsockopt() with this option returns the value set by setsockopt(). If a setsockopt() has not been done, the value 0 is returned.

Tip: This function is similar to the IPv4 socket option IP_MULTICAST_IF.

IPV6_MULTICAST_HOPS

The IPv6 header contains a hop limit field that controls the number of hops over which a datagram can be sent before being discarded. This is similar to the TTL field in the IPv4 header. The IPV6_MULTICAST_HOPS socket option can be used to set the default hop limit value for an outgoing multicast packet. The socket option value should be in the range 0 – 255. A socket option value of -1 is used to clear the socket option. This causes the default value 1 to be used.

A getsockopt() with this option returns the value set by a setsockopt(). If a setsockopt() has not been done, the default value 1 is returned.

The default value is 1. An application must be APF-authorized or have superuser authority to set this option to a value greater than the value of HOPLIMIT on the IPCONFIG6 statement. See z/OS Communications Server: IP Configuration Guide for more information about the IPCONFIG6 statement.

Tip: This function is similar to the IPv4 socket option IP_MULTICAST_TTL.

IPV6_MULTICAST_LOOP

When a multicast packet is sent, if the sender belongs to the multicast group to which the packet was sent, then this option controls whether the sender receives a copy of the packet. If this option is enabled, then the sender receives a copy of the packet. The socket option value should be 1 to enable the option, or 0 to disable the option.

A getsockopt() with this option returns the value set by a setsockopt(). If a setsockopt() has not been done, the default value of 1 (enabled) is returned.

Tip: This function is similar to the IPv4 socket option IP_MULTICAST_LOOP.

Options to control receiving of multicast packets

Use the following options to control receiving of multicast packets:

IPV6_JOIN_GROUP

Enables an application to join a multicast group on a specific local interface. The socket option data specifies an IPv6 multicast address and an IPv6 interface index. IPv4-mapped IPv6 multicast addresses are not supported. If an interface index of 0 is specified, the stack selects a local interface. An application that wants to receive multicast packets destined for a multicast group needs to join that group. It is not necessary to join a multicast group to send multicast packets.

Restriction: Getsockopt() does not support this option.

Tip: This function is similar to the IPv4 socket option IP_ADD_MEMBERSHIP.

IPV6_LEAVE_GROUP

Enables an application to leave a multicast group it previously joined. The socket option data specifies an IPv6 multicast address and an IPv6 interface index. If an interface index of 0 is used to join a multicast group, an interface index of 0 must be used to leave the group.

Restriction: Getsockopt() does not support this option.

Tip: This function is similar to the IPv4 socket option IP_DROP_MEMBERSHIP.

MCAST_JOIN_GROUP

Enables an application to join a multicast group on a specific local interface. The socket option data specifies an IPv4 or IPv6 multicast address and an IPv4 or IPv6 interface index. IPv4-mapped IPv6 multicast addresses are not supported. If the interface index 0 is specified, the stack selects a local interface. An application that wants to receive multicast packets destined for a multicast group needs to join that group. An application does not need to join a multicast group to send multicast packets.

Restriction: Getsockopt() does not support this option.

Tip: This function is similar to the IPv4 socket option IP_ADD_MEMBERSHIP and the IPv6 socket option IPV6_JOIN_GROUP.

MCAST_BLOCK_SOURCE
Enables an application to exclude the reception of multicast packets from specified source IP addresses. This socket option is issued after an MCAST_JOIN_GROUP option has been issued.

Restriction: Getsockopt() does not support this option.

Tip: This function is similar to the IPv4 socket option IP_BLOCK_SOURCE.

MCAST_UNBLOCK_SOURCE
Enables an application to include the reception of multicast packets from previously excluded source IP addresses. This socket option is issued after the MCAST_JOIN_GROUP and the MCAST_BLOCK_SOURCE options have been issued.

Restriction: Getsockopt() does not support this option.

Tip: This function is similar to the IPv4 socket option IP_UNBLOCK_SOURCE.

MCAST_JOIN_SOURCE_GROUP
Enables an application to join a multicast group on a specific local interface and for a specific source address. The socket option data specifies an IPv4 or IPv6 multicast address, an IPv4 or IPv6 interface index, and a single IPv4 or IPv6 source address. IPv4-mapped IPv6 multicast addresses and IPv4-mapped IPv6 source addresses are not supported. If the interface index 0 is specified, the stack selects a local interface. To receive multicast packets that are destined for a multicast group and that are from a particular source IP address, an application needs to join that group for the source address. An application does not need to join a multicast group to send multicast packets. MCAST_JOIN_SOURCE_GROUP cannot be used with MCAST_JOIN_GROUP.

Restriction: Getsockopt() does not support this option.

Tip: This function is similar to the IPv4 socket option IP_ADD_SOURCE_MEMBERSHIP.

MCAST_LEAVE_GROUP
Enables an application to leave a multicast group that it previously joined or to leave all sources that joined for a multicast group. The socket option data specifies an IPv4 or IPv6 multicast address and an IPv4 or IPv6 interface index. If the interface index 0 was specified on the MCAST_LEAVE_GROUP option to join a multicast group, an interface index of 0 must be specified to leave the group.

Restriction: Getsockopt() does not support this option.

Tip: This function is similar to the IPv4 socket option IP_DROP_MEMBERSHIP.

MCAST_LEAVE_SOURCE_GROUP
Enables an application to leave a source multicast group that it previously joined. The socket option data specifies an IPv4 or IPv6 multicast address, an IPv4 or IPv6 interface index, and a single IPv4 or IPv6 source address. If the interface index 0 was specified on the MCAST_JOIN_SOURCE_GROUP option to join a multicast group, an interface index 0 must be specified to leave the group. MCAST_LEAVE_SOURCE_GROUP is used to leave the group that was joined by MCAST_JOIN_SOURCE_GROUP.

Restriction: Getsockopt() does not support this option.

Tip: This function is similar to the IPv4 socket option IP_DROP_SOURCE_MEMBERSHIP.

Socket option to control IPv4 and IPv6 communications

Use the following option to control IPv4 and IPv6 communications:

IPV6_V6ONLY
An AF_INET6 socket can be used for IPv6 communications, IPv4 communications, or a mix of IPv6 and IPv4 communications. The IPV6_V6ONLY socket option allows an application to limit an AF_INET6 socket to IPv6 communications only. A nonzero socket option value enables the option; a value of 0 disables the option.

A getsockopt() with this option returns the value set by a setsockopt(). If a setsockopt() has not been done, the default value of 0 (disabled) is returned.

If an application wants to enable this option, the setsockopt() must be set before binding the socket, connecting the socket, or sending data over the socket. This option cannot be changed (either enabled or disabled) after the socket has been bound. (An implicit bind is done for datagram sockets on connect or send operations if the socket is not already bound.)

Socket options for SOL_SOCKET, IPPROTO_TCP and IPPROTO_IP levels

Socket options at the SOL_SOCKET and IPPROTO_TCP levels are not dependent on the IP layer being used. They are supported for both AF_INET and AF_INET6 sockets.

Socket options at the IPPROTO_IP level support IPv4. They are not supported on AF_INET6 sockets.

Not all socket options at these levels are supported by all APIs. See API-specific information for specific socket options for support levels.

Chapter 8. Enabling an application for IPv6

This topic describes how to enable an application for IPv6 and contains the following topics:

- "Changes to enable IPv6 support" on page 85
- "Support for unmodified applications" on page 85

Changes to enable IPv6 support

Several coding changes are needed to enable an application for IPv6 communications. Chapter 7, " Basic socket API extensions for IPv6," on page 71 describes the changes to the basic Socket APIs that most applications use. Chapter 9, "Advanced socket APIs," on page 95 describes the changes to advanced functions (which are typically used by a small number of TCP/IP applications) of the socket APIs that facilitate IPv6 communications. The divisions in this topic describe some of the general considerations involved in enabling an application for IPv6. Note that while many of the examples and references in this topic assume the use of C/C++ sockets supported by the Language Environment, most of the concepts (unless explicitly noted) apply to the other Socket API libraries that support IPv6. For a more detailed description of the actual APIs, see Chapter 7, " Basic socket API extensions for IPv6," on page 71 and Chapter 9, "Advanced socket APIs," on page 95, and information for the specific API you are using.

Guideline: You should be familiar with IPv6 in general and IPv6 support on z/OS Communications Server.

Support for unmodified applications

During the transition period where networks, routers, and hosts are upgraded to support IPv6, it is expected that most IPv6-enabled hosts also continue to have IPv4 connectivity. This is accomplished with dual-mode stack support that allows a single TCP/IP protocol stack to support both IPv4 and IPv6 communications. TCP/IP on z/OS supports dual-mode stack operation. As a result, applications that are not IPv6 enabled continue to function over an IPv4 network, without any changes. However, at some point during the IPv6 deployment process, some IP hosts might have connectivity only to IPv6 networks or have a TCP/IP protocol stack that is capable of IPv6 communications only. You can enable IPv6-only hosts to communicate with IPv4-only applications as described in "Enabling IPv6 communication between IPv6 nodes or networks in an IPv4 environment" on page 41 and "Enabling end-to-end communication between IPv4 and IPv6 applications" on page 41. If you do not use these methods, an application needs to be enabled for IPv6 in order to allow for communications with IPv6-only hosts or applications.

Application awareness of whether system is IPv6 enabled

A z/OS system might or might not be enabled for IPv6 communications. Enabling a z/OS system for IPv6 support requires explicit configuration by the system administrator to allow AF_INET6 sockets to be created. As a result, an application cannot typically assume that IPv6 is enabled on the systems where the application is running. Some exceptions do exist. For example, applications can run on a limited number of systems that are known to be IPv6 enabled. However, in general, most applications that are being enhanced to support IPv6 must first perform a runtime test to determine whether IPv6 is enabled on the system where they are executing. If the system is not enabled for IPv6, the application should proceed with its existing IPv4 logic. If the system is enabled for IPv6, the application can now use AF_INET6 sockets and features to communicate with both IPv4 and IPv6 applications.

Determine if a system is enabled for IPv6 by attempting to create an AF_INET6 socket. If this operation is successful, the application can assume that IPv6 is enabled. If the operation fails (with return code EAFNOSUPPORT) the application should revert to its IPv4 logic and create an AF_INET socket.

Table 17. Using socket() to determine IPv6 enablement

Affected socket API call	**Required changes**
socket()	Specify AF_INET6 as the Address Family (or domain) parameter. This API call fails if the system is not enabled for IPv6.

The getaddrinfo() API is an alternative mechanism that can be used by TCP/IP client applications to determine whether IPv6 is enabled. This API is a replacement for the gethostbyname() API and is typically used by TCP/IP client programs to resolve a host name to an IP address. For example, a client application that receives the server application's host name or IP address (such as FTP) as input can invoke the getaddrinfo() function before opening up a socket with a selected set of options. This allows the application to receive a list of addrinfo structures (one for each IP address of the destination host) that contain the following information:

- The address family of the IP address (AF_INET or AF_INET6)
- A pointer to a socket address structure of the appropriate type (sockaddr_in or sockaddr_in6) that is fully initialized (including the IP address and Port fields)
- The length of the socket address structure

A client application can be coded with this information in a manner that allows it to be protocol-independent without having to perform specific runtime checks to determine whether IPv6 is enabled and without having to have dual-path logic (IPv4 versus IPv6). The following application is an example of this approach:

```
int
myconnect(char *hostname)
{
  struct addrinfo *res, *aip;
  struct addrinfo hints;
  char buf[INET6_ADDRSTRLEN];
  static char *servicename = "21";
  int sock = -1;
  int error;

  /* Initialize the hints structure for getaddrinfo() call.
     This application can deal with either IPv4 or IPv6 addresses.
     It relies on getaddrinfo to return the most appropriate IP address
     and socket address structure based on the current configuration */

  bzero(&hints, sizeof (hints));
  hints.ai_socktype = SOCK_STREAM; /* Interested in streams sockets
                                      only                          */
  /* Note that we are asking for all IP addresses to be returned (IPv4
     or IPv6) based on the system connectivity.  Also, note that we
     would prefer all addresses to be returned in sockaddr_in6 format
     if the system is enabled for IPv6.  In addition, we also specify
     a numeric port using AI_NUMERICSERV so that the returned socket
     address structures are primed with our port number.    */

  hints.ai_flags = AI_ALL | AI_V4MAPPED | AI_ADDRCONFIG |
                   AI_NUMERICSERV;
  hints.ai_family = AF_UNSPEC;
  error = getaddrinfo(hostname, servicename, &hints, &res);
  if (error != 0) {
    (void) fprintf(stderr,
    "getaddrinfo: %s for host %s service %s\n",
    gai_strerror(error), hostname, servicename);
  return (-1);
  }
for (aip = res; aip != NULL; aip = aip->ai_next) {
/*
* Loop through list of addresses returned, opening sockets
* and attempting to connect()until successful.  The
* The address type depends on what getaddrinfo()
* gave us.
*/
  sock = socket(aip->ai_family, aip->ai_socktype,
            aip->ai_protocol);
  if (sock == -1) {
    printf("Socket failed:
    freeaddrinfo(res);
    return (-1);
  }
  /* Connect to the host. */
  if (connect(sock, aip->ai_addr, aip->ai_addrlen) == -1) {
    printf("Connect failed, errno=%d, errno2=%08x\n",
    errno, __errno2());
    (void) close(sock);
    sock = -1;
    continue;
  }
  break;
}
  freeaddrinfo(res);
  return (sock);
}
```

Figure 15. Example of protocol-independent client application

When this example executes on a system where IPv6 is not enabled, only IPv4 addresses are returned in AF_INET format (in sockaddr_in structures). When this identical example executes on an IPv6-enabled system, both IPv4 and IPv6 addresses are returned, and the IPv4 addresses are returned in IPv4-mapped IPv6 address format (in sockaddr_in6 structures). Note that an AF_INET6 socket can be used for the connection even when the address returned by getaddrinfo() is an IPv4-mapped IPv6 address.

Socket address structure changes

As mentioned in Chapter 7, " Basic socket API extensions for IPv6," on page 71, the socket address structure (sockaddr) is larger for IPv6 and has a slightly different format. This structure is passed as input or output on several socket API calls. The type of structure passed must match the address family of the socket being used on the socket API call. As a result, application changes are necessary. Table 18 on page 88 describes the necessary changes:

Table 18. sockaddr structure changes

Affected Socket API calls	**Required changes**
Bind(), connect(), sendmsg(), sendto()	The length and type of sockaddr structure passed must match the address family of the socket being used (structure sockaddr_in or sockaddr_in6).
accept(), recvmsg(), recvfrom(), getpeername(), getsockname()	The sockaddr structure passed needs to be sufficiently large for the address family of the socket being used on these APIs. Note that the larger sockaddr_in6 structure can be passed even for AF_INET sockets. However, the application needs to be aware that the format of the sockaddr structure returned depends on the address family of the input socket.
z/OS UNIX System Services BPX1SRX (Send/Recv CSM buffers using sockets)	The length and type of sockaddr structure passed must match the address family of the socket being used (structure sockaddr_in or sockaddr_in6).

Address conversion functions

Because IPv6 and IPv4 addresses have a different format and size, changes are required when formatting these addresses for presentation purposes. Two utility functions have been introduced for a selected set of socket APIs to help applications perform this processing. A formatted IPv6 address uses significantly more space than a formatted IPv4 address (46 bytes versus 16 bytes) and this might affect the layout of any messages and displays that include an IP address.

Table 19. Address conversion function changes

Affected API call	**Required changes**
Translating an IP address from numeric form to presentation form using inet_ntoa()	Convert to use inet_ntop() function. This function can be used for both IPv4 and IPv6 addresses.
Translating a presentation form IP address to numeric form using inet_addr()	Convert to use inet_pton() function. This function can be used for both IPv4 and IPv6 addresses.

Resolver API processing

TCP/IP applications typically need to resolve a host name to an IP address and sometimes need to resolve an IP address to a host name. Applications perform this processing by invoking resolver APIs, such as gethostbyname() and gethostbyaddr(). A new set of resolver APIs was introduced to support IPv6. Applications that currently use resolver APIs need to be modified to use the new APIs in order to be enabled for IPv6. The older resolver APIs continue to be supported for IPv4 communications. For more information about resolver APIs, see "Name and address resolution functions" on page 72.

Table 20. Resolver API changes

Affected API call	Required changes
gethostbyname()	Use new getaddrinfo() API. This API can be used even if the system is not IPv6 enabled. Note that the freeaddrinfo() API needs to be issued to free up storage areas returned by the getaddrinfo() API.
gethostbyaddr()	Use the new getnameinfo() API. This API can also be used on a system that is not IPv6 enabled.

Special IPv6 addresses

IPv4 provides two IP addresses that have the following special meaning in the context of socket programs:

- The Loopback Address, typically 127.0.0.1, allows applications to connect() to or send datagrams to other applications on the same host.
- The INADDR_ANY address (0.0.0.0) allows TCP/IP server applications that specify it on a bind() call to accept incoming connections or datagrams across any network interface configured on the local host.

The concept of these special IPv4 addresses is also available in IPv6. The changes are described in Table 21 on page 89.

Table 21. Special IPv6 address changes

Socket API calls	Required changes
Binding a socket to the IPv4 wildcard address (INADDR_ANY - 0.0.0.0)	Specify the unspecified IPv6 address (in6addr_any), (::), in the sockaddr_in6 structure.
Using LOOPBACK (127.0.0.1) on bind(), connect(), sendto(), sendmsg()	Specify IPv6 Loopback address (::1) in the sockaddr_in6 structure.

See Chapter 7, “ Basic socket API extensions for IPv6,” on page 71 for details about any constant definitions available for these special IPv6 addresses and the socket API that you are using.

Passing ownership of sockets across applications using givesocket and takesocket APIs

If your application is using the givesocket() and takesocket() APIs to pass ownership of a socket from one program to another, some changes are necessary for IPv6 enablement. The givesocket() and takesocket() APIs now support an address family of AF_INET6 for the socket being given or taken. The address family specified by the program performing the takesocket() must match the address family specified by the program that performed the givesocket(). As a result, care should be taken in coordinating the updates for IPv6 support across the partner applications performing givesocket and takesocket processing.

Table 22. givesocket() and takesocket() changes

Affected API call	Required changes
givesocket()	Specify AF_INET6 (Decimal 19) as the domain when giving an AF_INET6 socket.
getclientid()	Specify AF_INET6 as the domain when dealing with an AF_INET6 socket.
takesocket()	Specify AF_INET6 as the domain when taking an AF_INET6 socket.

Using multicast and IPv6

IPv6 provides enhanced support for multicast applications, including a more granular scope for multicast addressing and socket options that enable an application to use this support. Table 23 on page 90 lists

IPv4 multicast setsockopt() and getsockopt() options, the equivalent IPv6 multicast options, and protocol-independent multicast options.

Table 23. Multicast options

Multicast function	IPv4	IPv6	Protocol-independent
Level of specified option on setsockopt()/ getsockopt()	IPPROTO_IP	IPPROTO_IPV6	IPPROTO_IP or IPPROTO_IPV6
Join a multicast group	IP_ADD_MEMBERSHIP	IPV6_JOIN_GROUP	MCAST_JOIN_GROUP
Leave a multicast group or leave all sources of that multicast group	IP_DROP_MEMBERSHIP	IPV6_LEAVE_GROUP	MCAST_LEAVE_GROUP
Select outbound interface for sending multicast datagrams	IP_MULTICAST_IF	IPV6_MULTICAST_IF	NA
Set maximum hop count	IP_MULTICAST_TTL	IPV6_MULTICAST_HOPS	NA
Enable multicast loopback	IP_MULTICAST_LOOP	IPV6_MULTICAST_LOOP	NA
Join a source multicast group	IP_ADD_SOURCE_MEMBERSHIP	NA	MCAST_JOIN_SOURCE_GROUP
Leave a source multicast group	IP_DROP_SOURCE_MEMBERSHIP	NA	MCAST_LEAVE_SOURCE_GROUP
Block data from a source to a multicast group	IP_BLOCK_SOURCE	NA	MCAST_BLOCK_SOURCE
Unblock a previously blocked source for a multicast group	IP_UNBLOCK_SOURCE	NA	MCAST_UNBLOCK_SOURCE

In addition to the changes in the setsockopt() and getsockopt() options, the input and output parameters specified for these options are also changed when compared to IPv4. For example, selecting an outgoing interface for sending multicast IPv6 datagram involves passing an interface index that identifies the interface versus passing the IP address of the interface. For a detailed description of the IPv6 multicast options see "Options to control sending of multicast packets" on page 81.

An important consideration in updating your multicast application for IPv6 is how these changes are provided to the other partner applications participating in these multicast operations. For example, if a partner application in the network that is receiving these multicast packets is not updated, then the application sending the multicast datagrams might need to send them twice, once to an IPv4 multicast address and once to an IPv6 multicast address. Also, in order to perform this type of processing the application needs to create two separate sockets, an AF_INET socket and a AF_INET6 socket. There is no support equivalent to IPv4-mapped IPv6 addresses that would allow an AF_INET6 socket to be used in sending IPv4 multicast packets. As an alternative solution, first enable all the receiver applications for IPv6 and then enable the sender applications.

IP addresses might not be permanent

Long-term use of an address is discouraged as IPv6 allows for IP addresses to be dynamically renumbered. Applications should rely on DNS resolvers to cache the appropriate IP addresses and should avoid having IP addresses in configuration files.

Including IP addresses in the data stream

Applications that include IP addresses in the data they transmit over TCP/IP require changes when enabling for IPv6, as the IPv6 addresses have a different format from IPv4 addresses. The following options can be considered in dealing with these changes:

- Determine whether IP addresses are needed in the data exchanged by the applications.
- Change the partner applications processing to always send IP addresses encoded using IPv6 format. In the case where IPv4 addresses are being used, they can be represented as IPv4-mapped IPv6 addresses.
- Include a version identifier that describes the format of the IP address being sent (IPv4 or IPv6).
- Modify applications to use host names instead of IP addresses in the data stream. This approach requires that the partner receiving the host name is able to resolve it to an IP address. Also note that a single IP host can have multiple IP addresses.
- In many cases, you might not be able to change all partner applications in your network at the same time. As a result, determining the type of IP address to send is a key consideration. Consider the following options when making this decision:
  - Determine the level of support when the connection is established by exchanging version or supported functions.
  - Encode the IPv6 addresses using new options. If the option is rejected by the peer, then it does not support IPv6.
  - Base the decision on the partner application's IP address. If the partner's source IP address is an IPv4 address, use only IPv4 addresses; otherwise, use an IPv6 address. This option can cause an IPv6-enabled partner application to be treated as an IPv4 partner if that application uses an IPv4-mapped IPv6 address to connect.

Example of an IPv4 TCP server program

The following example shows a simple IPv4 TCP server program written in C. The program opens a TCP socket, binds it to port 5000, and then performs a listen() followed by an accept() call. When a connection is accepted the server sends a `Hello` text string back to the client and closes the socket. This sample program is later shown with the changes required to make it IPv6 enabled.

```
/* simpleserver.c
   A very simple TCP socket server
 */
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(int argc,const char **argv)
{
  int serverPort = 5000;
  int rc;
  struct sockaddr_in serverSa;
  struct sockaddr_in clientSa;
  int clientSaSize;
  int on = 1;
  int c;
  int s = socket(PF_INET,SOCK_STREAM,0);
  rc = setsockopt(s,SOL_SOCKET,SO_REUSEADDR,&on,sizeof on);
  /* initialize the server's sockaddr */
  memset(&serverSa,0,sizeof(serverSa));
  serverSa.sin_family = AF_INET;
  serverSa.sin_addr.s_addr = htonl(INADDR_ANY);
  serverSa.sin_port = htons(serverPort);
  rc = bind(s,(struct sockaddr *)&serverSa,sizeof(serverSa));
  if (rc < 0)
  {
    perror("bind failed");
    exit(1);
  }
  rc = listen(s,10);
  if (rc < 0)
  {
    perror("listen failed");
    exit(1);
  }
  rc = accept(s,(struct sockaddr *)&clientSa,&clientSaSize);
  if (rc < 0)
  {
    perror("accept failed");
    exit(1);
  }
  printf("Client address is:
  c = rc;
  rc = write(c,"hello\n",6);
  close (s);
  close (c);
  return 0;
}
```

Example of the simple TCP server program enabled for IPv6

The simple TCP server program is shown with the changes that are required to allow it to accept connections from IPv6 clients.

```
/*
   A very simple TCP socket server for v4 or v6
  */
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
int main(int argc,const char **argv)
{
  int serverPort = 5000;
  int rc;
  union {
    struct sockaddr_in sin;
    struct sockaddr_in6 sin6;
  } serverSa;
  union {
    struct sockaddr_in sin;
    struct sockaddr_in6 sin6;
  } clientSa;

  int clientSaSize = sizeof(clientSa);
  int on = 1;
  int family;
  socklen_t serverSaSize;
  int c;
  char buf[INET6_ADDRSTRLEN];

  int s = socket(PF_INET6,SOCK_STREAM,0);
  if (s < 0)
  {
    fprintf(stderr, "IPv6 not active, falling back to IPv4...\n");
    s = socket(PF_INET,SOCK_STREAM,0);
    if (s < 0)
    {
      perror("socket failed");
      exit (1);
    }
    family = AF_INET;
    serverSaSize = sizeof(struct sockaddr_in);
  }
 else  /* got a v6 socket */
  {
    family = AF_INET6;
    serverSaSize = sizeof(struct sockaddr_in6);
  }
  printf("socket descriptor is
 rc = setsockopt(s,SOL_SOCKET,SO_REUSEADDR,&on,sizeof on);

  /* initialize the server's sockaddr */
  memset(&serverSa,0,sizeof(serverSa));
  switch(family)
  {
    case AF_INET:
      serverSa.sin.sin_family = AF_INET;
      serverSa.sin.sin_addr.s_addr = htonl(INADDR_ANY);
      serverSa.sin.sin_port = htons(serverPort);
      break;
    case AF_INET6:
      serverSa.sin6.sin6_family = AF_INET6;
      serverSa.sin6.sin6_addr = in6addr_any;
      serverSa.sin6.sin6_port = htons(serverPort);
  }
```

```
  rc = bind(s,(struct sockaddr *)&serverSa,serverSaSize);
  if (rc < 0)
  {
    perror("bind failed");
    exit(1);
  }
  rc = listen(s,10);
  if (rc < 0)
  {
    perror("listen failed");
    exit(1);
  }
  rc = accept(s,(struct sockaddr *)&clientSa,&clientSaSize);
  if (rc < 0)
  {
    perror("accept failed");
    exit(1);
  }
  c = rc;
  printf("Client address is: %s\n",
       inet_ntop(clientSa.sin.sin_family,
                 clientSa.sin.sin_family == AF_INET
                   ? &clientSa.sin.sin_addr
                   : &clientSa.sin6.sin6_addr,
                 buf, sizeof(buf)));

  if(clientSa.sin.sin_family == AF_INET6
    && ! IN6_IS_ADDR_V4MAPPED(&clientSa.sin6.sin6_addr))
    printf("Client is v6\n");
  else
    printf("Client is v4\n");

  rc = write(c,"hello\n",6);
  close (s);
  close (c);
  return 0;
}
```

Chapter 9. Advanced socket APIs

This topic describes the advanced socket APIs and includes the following topics:

- “Controlling the content of the IPv6 packet header” on page 95
- “Using ancillary data on sendmsg() and recvmsg()” on page 106
- “Interactions between socket options and ancillary data” on page 107
- “RAW sockets” on page 108

Before using advanced socket APIs in a multilevel security environment, see z/OS Communications Server: IP Configuration Guide. The advanced socket API for IPv6 support includes:

- IPv6 RAW socket support
- New socket options
- New ancillary data objects on sendmsg/recvmsg
- The ability to receive inbound packet information, including:
  - Arriving interface index
  - Destination IP address
  - Hop limit
  - Routing headers
  - Hop-by-hop options
  - Destination options
  - Traffic class by way of ancillary data
- The ability to set outgoing packet information, including:
  - Interface to use
  - Source IP address
  - Hop limit
  - Next hop address
  - Routing headers
  - Hop-by-hop options
  - Destination option
  - Traffic class (This can be set by socket options or ancillary data with some restrictions.)

z/OS UNIX C/C++ and z/OS UNIX Assembler Callable APIs support the advanced socket API for IPv6. The advanced socket API for IPv6 is not implemented in native TCP/IP socket APIs.

To obtain structure and length information that is related to the IPv6 advanced socket APIs, see http://tools.ietf.org/html/rfc3542.

Controlling the content of the IPv6 packet header

This topic contains information about socket options and how to control the content of the IPv6 packet header.

Socket options and ancillary data to support IPv6 (IPPROTO_IPV6 level)

An application can use socket options to enable or disable a function for a socket. An application can also provide a value to be used for a function with a socket option. After an option is enabled, it remains in effect for the socket until it is disabled.

An application can also use ancillary data on the sendmsg() API to enable a function or provide a value for the packet being sent by way of sendmsg(). The value of the ancillary data is in effect for that packet only. Note that the value of the ancillary data can override a socket option value. For a detailed explanation of ancillary data, see "Using ancillary data on sendmsg() and recvmsg()" on page 106.

An application can also receive ancillary data on the recvmsg() API. The returned ancillary data is enabled for any socket options that return data on recvmsg.

A group of advanced socket options and ancillary data is defined to support IPv6. They are defined with a level of IPPROTO_IPV6 or IPPROTO_ICMPV6. The individual options begin with IPV6_ and ICMP6_. These options are allowed on AF_INET6 sockets only. In most cases, these options can be set on an AF_INET6 socket that is using IPv4-mapped IPv6 addresses, but have no effect. For example, the IPV6_HOPLIMIT ancillary data option is used to set a hop limit value in the IPv6 header. Because IPv4 packets are used with IPv4-mapped IPv6 addresses, the hop limit value is not used. The following options are the only advanced socket options that have an effect on an AF_INET6 socket that is using IPv4–mapped IPv6 addresses:

- IPV6_PKTINFO
- IPV6_RECVPKTINFO
- IPV6_TCLASS
- IPV6_RECVTCLASS

Table 24. Sockets options at the IPPROTO_IPV6 level

Socket options getsockopt() setsockopt()	**z/OS UNIX Assembler Callable Services**	**C/C++ using Language Environment**	**REXX**	**Communications Server Sockets Extended macro/call**
IPV6_CHECKSUM	Y	Y	N	N
IPV6_DONTFRAG	Y	Y	N	N
IPV6_DSTOPTS	Y	Y	N	N
IPV6_HOPOPTS	Y	Y	N	N
IPV6_NEXTHOP	Y	Y	N	N
IPV6_PATHMTU[valid only on getsockopt()]	Y	Y	N	N
IPV6_PKTINFO	Y	Y	N	N
IPV6_RECVDSTOPTS	Y	Y	N	N
IPV6_RECVHOPLIMIT	Y	Y	N	N
IPV6_RECVHOPOPTS	Y	Y	N	N
IPV6_RECVPATHMTU	Y	Y	N	N
IPV6_RECVPKTINFO	Y	Y	N	N
IPV6_RECVRTHDR	Y	Y	N	N
IPV6_RECVTCLASS	Y	Y	N	N
IPV6_RTHDR	Y	Y	N	N
IPV6_RTHDRDSTOPTS	Y	Y	N	N
IPV6_TCLASS	Y	Y	N	N
IPV6_USE_MIN_MTU	Y using BPX1	Y	N	N

[1] This option is supported as ancillary data for UDP and RAW protocols. It is not possible to use ancillary data to transmit options for TCP because there is not a one-to-one mapping between send operations and the TCP segments being transmitted.

Table 25. Ancillary data on sendmsg() (Level = IPPROTO_IPV6)

Ancillary data on sendmsg()	Assembler Callable Services	C/C++ using Language Environment	REXX	Sockets Extended macro/call
IP_QOS_ CLASSIFICATION[1]	Y	Y	N	N
IPV6_DONTFRAG	Y	Y	N	N
IPV6_DSTOPTS	Y	Y	N	N
IPV6_HOPLIMIT[1]	Y	Y	N	N
IPV6_HOPOPTS	Y	Y	N	N
IPV6_NEXTHOP	Y	Y	N	N
IPV6_PKTINFO[1]	Y	Y	N	N
IPV6_RTHDR	Y	Y	N	N
IPV6_RTHDRDSTOPTS	Y	Y	N	N
IPV6_TCLASS	Y	Y	N	N
IPV6_USE_MIN_MTU	Y	Y	N	N

Table 26. Ancillary data on recvmsg() (Level = IPPROTO_IPV6)

Ancillary data on recvmsg()	Assembler Callable Services	C/C++ using Language Environment	REXX	Sockets Extended macro/call
IPV6_DSTOPTS	Y	Y	N	N
IPV6_HOPLIMIT	Y	Y	N	N
IPV6_HOPOPTS	Y	Y	N	N
IPV6_PATHMTU	Y	Y	N	N
IPV6_PKTINFO	Y	Y	N	N
IPV6_RTHDR	Y	Y	N	N
IPV6_TCLASS	Y	Y	N	N

Options for path MTU discovery

Use the following options for path MTU discovery:

IPV6_USE_MIN_MTU (used with TCP, UDP and RAW applications)

For IPv6, only the endpoint nodes can fragment a packet. Path MTU discovery determines the largest packet that can be sent to a destination without requiring fragmentation by an intermediate node (because that is not supported). In some cases, an application might want to avoid path MTU discovery. All nodes in an IPv6 network are required to support a minimum MTU of 1280 bytes. When an application enables this option, path MTU discovery is bypassed. If a direct route to the destination is not available, the minimum MTU size (1280 bytes) is used to send packets that otherwise might require fragmentation. If a direct route is available, the link's MTU size is used, because path MTU discovery is not needed when there are no intermediate nodes in the path.

For unicast destinations, this option is disabled by default, which avoids sending packets with the minimum MTU size. Instead, path MTU discovery information is used.

For multicast destinations, this option is enabled by default, which prevents path MTU discovery information from being used. If a direct route is not available, packets are sent with the minimum MTU size. If a direct route is available, packets are sent by using the MTU of the link because no intermediate nodes are in the path.

This option can be enabled or disabled for the following cases:

- A socket with a setsockopt()

- A single send operation with ancillary data on the sendmsg()

A value of -1 passed on the set socket option causes the default values for unicast and multicast destinations to be used.

A value of 0 disables this option for both unicast and multicast destinations. Path MTU discovery information is used to send packets greater than the minimum MTU size.

A value of 1 enables this option for unicast and multicast destinations. All packets are sent without using path MTU discovery information, using the minimum MTU size, unless a direct route is available to the destination.

A getsockopt() with this option returns the value set by a setsockopt(). If a setsockopt() has not been done, the default value of -1 (disabled for unicast, enabled for multicast) is returned.

IPV6_DONTFRAG (used with UDP and RAW applications)

The IPV6_DONTFRAG option enables the application to indicate that the packet should not be fragmented by the local z/OS host.

This option is useful for applications that want to discover the actual path MTU.

Guideline: When using the IPV6_DONTFRAG socket option, use the IPV6_RECVPATHMTU socket option also. Otherwise, packets are silently discarded without any notification to the application.

This option can be enabled or disabled for the following cases:

- A socket with a setsockopt()
- A single send operation with ancillary data on the sendmsg()

A value of 1 enables this option for unicast and multicast destinations.

A getsockopt() with this option returns the value set by a setsockopt(). If a setsockopt() has not been performed, then getsockopt() returns a value of 0.

If IPV6_DONTFRAG is specified along with IPV6_USE_MIN_MTU, the IPV6_DONTFRAG setting is ignored, resulting in selection of the minimum architected IPv6 MTU size (1280 bytes).

IPV6_RECVPATHMTU (used with UDP and RAW applications)

The IPV6_RECVPATHMTU option enables the application to receive notifications about changes to the path MTU. This option notifies the application about all path MTU changes for all destinations, not only the changes that this socket initiates.

When the IPV6_RECVPATHMTU socket option is enabled, the path MTU is returned as ancillary data on the recvmsg() API (for an empty message) whenever the path MTU changes. The path MTU can change if the application sends a packet with the IPV6_DONTFRAG option and the packet is larger than the current path MTU. The path MTU can also change if the stack receives a corresponding ICMPv6 `packet too big` error. The ancillary data that is returned has level IPPROTO_IPV6 and name IPV6_PATHMTU. For more information about ancillary data, see "Using ancillary data on sendmsg() and recvmsg()" on page 106.

This option can be enabled or disabled for a socket with a setsockopt().

A value of 1 enables this option.

A getsockopt() with this option returns the value set by a setsockopt(). If a setsockopt() has not been performed, then getsockopt() returns a value of 0.

IPV6_PATHMTU (used with UDP and RAW applications)

The IPV6_PATHMTU option enables the application to retrieve the current path MTU to a given destination for which it has done a connect().

This option is useful for applications also using IPV6_RECVPATHMTU that want to pick a good starting value.

This option is valid only on a getsockopt(). It returns the MTU that the stack uses on this connected socket.

Options to control the sending of packets

Some of these options add extension headers to outbound packets. z/OS TCP/IP allows the application to specify a maximum of 512 bytes of extension headers for an outbound packet.

Use the following options to control the sending of packets:

IPV6_PKTINFO (used with UDP and RAW applications)
The IPV6_PKTINFO option enables the application to provide the following pieces of information:

- The source IP address for an outgoing packet
- The outgoing interface for a packet

The option value contains a 16-byte IPv6 address and a 4-byte interface index. An application can provide a nonzero value for one or both pieces of information.

To perform this operation, an application must meet one of the following criteria:

- The application must be APF-authorized.
- The application must have superuser authority.
- The SERVAUTH resource EZB.SOCKOPT.*sysname*.*tcpname*.IPV6_PKTINFO must be defined and the application must at least have READ access to it.

This option can be enabled or disabled for the following cases:

- A socket with a setsockopt()
- A single send operation with ancillary data on the sendmsg()

To disable the option, specify both the IPv6 address and the interface index as 0 in the option value.

A getsockopt() with this option returns the value set by setsockopt(). If a setsockopt() has not been done, a value of 0 is returned.

See “Options for setting the source address” on page 108 for a discussion of the interaction of socket options and ancillary data for the setting of the source address. See “Options for specifying the outgoing interface” on page 108 for a discussion of the interaction of socket options and ancillary data for determining the outgoing interface.

IPV6_HOPLIMIT (used with UDP and RAW applications)
The IPv6 header contains a hop limit field that controls the number of hops over which a datagram can be sent before being discarded. This is similar to the TTL field in the IPv4 header. The IPV6_HOPLIMIT option can be used to set the hop limit value for an outgoing packet. The option value should be between 0 and 255 inclusive. A value of -1 causes the TCP/IP protocol stack default to be used.

To perform this operation, an application must meet one of the following criteria:

- The application must be APF-authorized.
- The application must have superuser authority.
- The SERVAUTH resource EZB.SOCKOPT.*sysname*.*tcpname*.IPV6_HOPLIMIT must be defined and the application must at least have READ access to it.

The IPV6_UNICAST_HOPS socket option and the IPV6_MULTICAST_HOPS socket option are available to set a hop limit value also. See “Hop limit options” on page 107 for information about the interaction of IPV6_UNICAST_HOPS, IPV6_MULTICAST_HOPS and IPV6_HOPLIMIT.

IPV6_NEXTHOP (used with UDP and RAW applications)
The IPV6_NEXTHOP option enables the application to specify the next hop address for an outgoing packet. The option value contains a sockaddr_in6 socket address structure and must contain an IPv6 address.

Restriction: This option does not support IPv4-mapped IPv6 addresses.

To perform this operation, an application must meet one of the following criteria:

- The application must be APF-authorized.

- The application must have superuser authority.
- The SERVAUTH resource EZB.SOCKOPT.*sysname*.*tcpname*.IPV6_NEXTHOP must be defined and the application must at least have READ access to it.

This option can be enabled or disabled for the following cases:

- A socket with a setsockopt()
- A single send operation with ancillary data on the sendmsg()

Restriction: IPV6_NEXTHOP is valid only for unicast destinations.

An option value with the *optlen* value of 0 disables IPV6_NEXTHOP. This option does not have any meaning for multicast destinations and is ignored for multicast.

A getsockopt() with this option returns the value set by a setsockopt(). If a setsockopt() has not been performed, then getsockopt() returns the value 0 in *optlen*.

See "Options for specifying the outgoing interface" on page 108 for information about the interaction of socket options and ancillary data for determining the outgoing interface.

Tips:

- If you use this socket option in a Common INET environment, establish affinity to the appropriate stack to ensure predictable results; some stacks might not have a route to the specified next hop address.
- If you specify a link-local address as the next hop address, specify the outgoing interface either on IPV6_PKTINFO or by using the scope portion of the socket address structure.

Rule: The next hop address cannot be a multicast address and must be a neighbor (for example, the stack must have a direct route to the next hop address).

IPV6_RTHDR (used with UDP and RAW applications)

The IPV6_RTHDR option enables the application to specify an IPv6 routing header (as an extension header) for an outgoing packet.

Restriction: Because the type 0 routing header is deprecated in z/OS Communications Server V1R11, no routing header type is currently supported. The IPV6_RTHDR option is accepted as a valid option, but all option type values are rejected as incorrect values.

To perform this operation, an application must meet one of the following criteria:

- The application must be APF-authorized.
- The application must have superuser authority.
- The SERVAUTH resource EZB.SOCKOPT.*sysname*.*tcpname*.IPV6_RTHDR must be defined and the application must at least have READ access to it.

This option can be enabled or disabled for the following cases:

- A socket with a setsockopt()
- A single send operation with ancillary data on the sendmsg()

A getsockopt() with this option returns the value set by a setsockopt(). If a setsockopt() has not been performed, then getsockopt() returns a value of 0 in *optlen*.

Tip: If you use this socket option in a Common INET environment, establish affinity to the appropriate stack to ensure predictable results; some stacks might not have a path to the first destination that is specified in the routing header.

A z/OS UNIX C/C++ application can use the following utilities to build routing headers:

- inet6_rth_space() - return number of bytes required for routing header
- inet6_rth_init() - initialize buffer data for routing header
- inet6_rth_add() - add one IPv6 address to the routing header

See z/OS XL C/C++ Runtime Library Reference for a description of these utilities.

A z/OS UNIX Assembler Callable Services application needs to build the routing headers explicitly. See z/OS UNIX System Services Programming: Assembler Callable Services Reference for information about z/OS UNIX Assembler Callable Services and the data structures defined in the BPXYSOCK macro.

IPV6_DSTOPTS (used with UDP and RAW applications)

The IPV6_DSTOPTS option enables the application to specify destination options that get examined by the host at the final destination.

The IPV6_DSTOPTS option can be used to set a destination options header (as an extension header) for an outgoing packet. The option value contains a destination options header.

To perform this operation, an application must meet one of the following criteria:

- The application must be APF-authorized.
- The application must have superuser authority.
- The SERVAUTH resource EZB.SOCKOPT.*sysname*.*tcpname*.IPV6_DSTOPTS must be defined and the application must at least have READ access to it.

This option can be enabled or disabled for the following cases:

- A socket with a setsockopt()
- A single send operation with ancillary data on the sendmsg()

A getsockopt() with this option returns the value set by a setsockopt(). If a setsockopt() has not been performed, then getsockopt() returns a value of 0 in optlen.

A z/OS UNIX C/C++ application can use the following utilities to build the following destination options headers:

- inet6_opt_init() - initialize buffer data for options header
- inet6_opt_append() - add one TLV option to the options header
- inet6_opt_finish() - finish adding TLV options to the option header
- inet6_opt_set_val() - add one component of the option content to the option

See z/OS XL C/C++ Runtime Library Reference for a description of these utilities.

A z/OS UNIX Assembler Callable Services application needs to build the options headers explicitly. See z/OS UNIX System Services Programming: Assembler Callable Services Reference for information about z/OS UNIX Assembler Callable Services and the data structures defined in the BPXYSOCK macro.

IPV6_RTHDRDSTOPTS (used with UDP and RAW applications)

The IPV6_RTHDRDSTOPTS option enables the application to specify destination options that get examined by every IP host that appears in the routing header.

The IPV6_RTHDRDSTOPTS option can be used to set a destination options header (as an extension header) for an outgoing packet. The option value contains a destination options header. This option is ignored if the application does not also use the IPV6_RTHDR option to specify a routing header.

To perform this operation, an application must meet one of the following criteria:

- The application must be APF-authorized.
- The application must have superuser authority.
- The SERVAUTH resource EZB.SOCKOPT.*sysname*.*tcpname*.IPV6_RTHDRDSTOPTS must be defined and the application must at least have READ access to it.

This option can be enabled or disabled for the following cases:

- A socket with a setsockopt()
- A single send operation with ancillary data on the sendmsg()

A getsockopt() with this option returns the value set by a setsockopt(). If a setsockopt() has not been performed, then getsockopt() returns a value of 0 in optlen.

A z/OS UNIX C/C++ application can use the following utilities to build destination options headers:

- inet6_opt_init() - initialize buffer data for options header
- inet6_opt_append() - add one TLV option to the options header
- inet6_opt_finish() - finish adding TLV options to the option header
- inet6_opt_set_val() - add one component of the option content to the option

See z/OS XL C/C++ Runtime Library Reference for a description of these utilities.

A z/OS UNIX Assembler Callable Services application needs to build the options headers explicitly. See z/OS UNIX System Services Programming: Assembler Callable Services Reference for information about z/OS UNIX Assembler Callable Services and the data structures defined in the BPXYSOCK macro.

IPV6_TCLASS (used with TCP, UDP and RAW applications)

The IPv6 header contains a traffic class field that can be used to identify and distinguish between different classes or priorities of IPv6 packets. This is similar to the type of service (ToS) field in the IPv4 header. The IPV6_TCLASS option can be used to set the traffic class value for an outgoing packet. However, if a QoS policy that specifies a traffic class for the packet is also in effect, then the stack ignores the value specified with the IPV6_TCLASS option and uses the value specified by the QoS policy.

To perform this operation, an application must meet one of the following criteria:

- The application must be APF-authorized.
- The application must have superuser authority.
- The SERVAUTH resource EZB.SOCKOPT.*sysname*.*tcpname*.IPV6_TCLASS must be defined and the application must at least have READ access to it.

This socket option is also valid for an AF_INET6 socket that is using IPv4-mapped IPv6 addresses.

This option can be enabled or disabled for a socket with a setsockopt(). For UDP and RAW, this option can be enabled or disabled for a single send operation with ancillary data on the sendmsg().

The option value is in the range 0 – 255. The value -1 causes the TCP/IP protocol stack to use the traffic class value that the policy specifies (if any) or the default value 0.

A getsockopt() with this option returns the value set by a setsockopt(). If a setsockopt() has not been performed, then the stack returns the traffic class value specified by policy (if any) or the default value 0.

Options that provide information about packets that have been received

To get information about packets that have been received, use the following options:

IPV6_RECVPKTINFO (used with UDP and RAW applications)

The IPV6_RECVPKTINFO socket option allows an application to receive the following pieces of information:

- The destination IP address from the IPv6 header
- The interface index for the interface over which the packet was received

When the IPV6_RECVPKTINFO socket option is enabled, the IP address and interface index are returned as ancillary data on the recvmsg() API. The ancillary data level is IPPROTO_IPV6. The option name is IPV6_PKTINFO. For a detailed explanation of ancillary data, see "Using ancillary data on sendmsg() and recvmsg()" on page 106.

Restriction: This option can be enabled or disabled only with a setsockopt(). IPV6_RECVPKTINFO is not valid as ancillary data on sendmsg(). A nonzero option value enables the option; a value of 0 disables the option.

A getsockopt() with this option returns the value set by a setsockopt(). If a setsockopt() has not been done, the default value of 0 (disabled) is returned.

IPV6_RECVHOPLIMIT (used with TCP, UDP and RAW applications)

The IPV6_RECVHOPLIMIT socket option allows an application to receive the value of the hop limit field from the IPv6 header. When the IPV6_RECVHOPLIMIT socket option is enabled, the hop limit is returned as ancillary data on the recvmsg() API. The ancillary data level is IPPROTO_IPV6. The option name is IPV6_HOPLIMIT. For a UDP or RAW application, if this option is enabled, the IPV6_HOPLIMIT ancillary data is returned with each recvmsg(). For a TCP application, if this option is enabled, IPV6_HOPLIMIT ancillary data is returned on recvmsg() only when the hop limit value being used has changed. For a detailed explanation of ancillary data, see "Using ancillary data on sendmsg() and recvmsg()" on page 106.

This option can be enabled or disabled only with a setsockopt(). IPV6_RECVHOPLIMIT is not valid as ancillary data on sendmsg(). A nonzero option value enables the option; a value of 0 disables the option.

A getsockopt() with this option returns the value set by a setsockopt(). If a setsockopt() has not been done, the default value of 0 (disabled) is returned.

IPV6_RECVRTHDR (used with UDP and RAW applications)

The IPV6_RECVRTHDR socket option enables the application to receive a routing header.

When the IPV6_RECVRTHDR socket option is enabled, the routing header is returned as ancillary data on the recvmsg() API. Each routing header is returned as one ancillary data object. The ancillary data level is IPPROTO_IPV6. The option name is IPV6_RTHDR. For a detailed explanation of ancillary data, see "Using ancillary data on sendmsg() and recvmsg()" on page 106.

This option can be enabled or disabled only with a setsockopt(). IPV6_RECVRTHDR is not valid as ancillary data on sendmsg(). A nonzero value enables the option; a value of 0 disables the option.

A getsockopt() with this option returns the value set by a setsockopt(). If a setsockopt() has not been performed, then getsockopt() returns a value of 0.

A z/OS UNIX C/C++ application can use the following utilities to process routing headers:

- inet6_rth_reverse() - reverse a routing header
- inet6_rth_segments() - return number of segments in a routing header
- inet6_rth_getaddr() - fetch one address from a routing header

See z/OS XL C/C++ Runtime Library Reference for a description of these utilities.

A z/OS UNIX Assembler Callable Services application needs to build the options headers explicitly. See z/OS UNIX System Services Programming: Assembler Callable Services Reference for information about z/OS UNIX Assembler Callable Services and the data structures defined in the BPXYSOCK macro.

IPV6_RECVHOPOPTS (used with UDP and RAW applications)

The IPV6_RECVHOPOPTS socket option enables the application to receive hop-by-hop options.

When the IPV6_RECVHOPOPTS socket option is enabled, the hop-by-hop options are returned as ancillary data on the recvmsg() API. The ancillary data level is IPPROTO_IPV6. The option name is IPV6_HOPOPTS. For a detailed explanation of ancillary data, see "Using ancillary data on sendmsg() and recvmsg()" on page 106.

This option can be enabled or disabled only with a setsockopt(). IPV6_RECVHOPOPTS is not valid as ancillary data on sendmsg(). A nonzero value enables the option; a value of 0 disables the option.

A getsockopt() with this option returns the value set by a setsockopt(). If a setsockopt() has not been performed, then getsockopt() returns a value of 0.

A z/OS UNIX C/C++ application can use the following utilities to process hop-by-hop options headers:

- inet6_opt_next() - extract the next option from the options header
- inet6_opt_find() - extract an option of a specified type from the header

- inet6_opt_get_val() - retrieve one component of the option content

See z/OS XL C/C++ Runtime Library Reference for a description of these utilities.

A z/OS UNIX Assembler Callable Services application needs to build the options headers explicitly. See z/OS UNIX System Services Programming: Assembler Callable Services Reference for information about z/OS UNIX Assembler Callable Services and the data structures defined in the BPXYSOCK macro.

IPV6_RECVDSTOPTS (used with UDP and RAW applications)

The IPV6_RECVDSTOPTS socket option enables the application to receive destination options.

When the IPV6_RECVDSTOPTS socket option is enabled, the destination options are returned as ancillary data on the recvmsg() API. The application can receive two destination options headers if a received packet contains a routing header and one destination options header before the routing header and one destination options header after the routing header. Each destination options header is returned as one ancillary data object. The ancillary data level is IPPROTO_IPV6. The option name is IPV6_DSTOPTS. For more information about ancillary data, see "Using ancillary data on sendmsg() and recvmsg()" on page 106.

This option can be enabled or disabled only with a setsockopt(). IPV6_RECVDSTOPTS is not valid as ancillary data on sendmsg(). A nonzero value enables the option; a value of 0 disables the option.

A getsockopt() with this option returns the value set by a setsockopt(). If a setsockopt() has not been performed, then getsockopt() returns a value of 0.

A z/OS UNIX C/C++ application can use the following utilities to process destination options headers:

- inet6_opt_next() - extract the next option from the options header
- inet6_opt_find() - extract an option of a specified type from the header
- inet6_opt_get_val() - retrieve one component of the option content

See z/OS XL C/C++ Runtime Library Reference for a description of these utilities.

A z/OS UNIX Assembler Callable Services application needs to build the options headers explicitly. See z/OS UNIX System Services Programming: Assembler Callable Services Reference for information about z/OS UNIX Assembler Callable Services and the data structures defined in the BPXYSOCK macro.

IPV6_RECVTCLASS (used with TCP, UDP and RAW applications)

The IPV6_RECVTCLASS socket option enables the application to receive the value of the traffic class field from the IPv6 header.

When the IPV6_RECVTCLASS socket option is enabled, the traffic class is returned as ancillary data on the recvmsg() API. The ancillary data level is IPPROTO_IPV6. The option name is IPV6_TCLASS. For a UDP, or RAW application, if this option is enabled, the IPv6_TCLASS ancillary data is returned with each recvmsg(). For a TCP application, if this option is enabled, IPV6_TCLASS ancillary data is returned on recvmsg() only when the traffic class value being used has changed. For a detailed explanation of ancillary data, see "Using ancillary data on sendmsg() and recvmsg()" on page 106 .

This socket option is also valid for an AF_INET6 socket that is using IPv4-mapped IPv6 addresses.

This option can be enabled or disabled only with a setsockopt(). IPV6_RECVTCLASS is not valid as ancillary data on sendmsg(). A nonzero value enables the option; a value of 0 disables the option.

A getsockopt() with this option returns the value set by a setsockopt(). If a setsockopt() has not been performed, then getsockopt() returns a value of 0.

Option to provide checksum processing for RAW applications

Use the following option to provide checksum processing for RAW applications:

IPV6_CHECKSUM (used with RAW applications)

The IPV6_CHECKSUM socket option can be used by a RAW application to enable checksum processing to be done by the TCP/IP protocol stack for packets on a socket. When enabled, the checksum is computed and stored for outbound packets; the checksum is verified for inbound packets. Note that this socket option is not applicable for ICMPv6 RAW sockets because the TCP/IP protocol stack always provides checksum processing for them.

This option can be enabled or disabled only with a setsockopt(). IPV6_CHECKSUM is not valid as ancillary data on sendmsg(). The option value provides the offset into the user data where the checksum field begins. The option value should be an even number in the range 0 - 65534. The value -1 causes the option to be disabled.

A getsockopt() with this option returns the value set by a setsockopt(). If a setsockopt() has not been done, the value -1 (disabled) is returned.

Option to provide QoS classification data

Use the following option to provide QoS classification data:

IP_QOS_CLASSIFICATION (used with TCP applications)

This option enables the application to provide QoS classification data. It is a z/OS Communications Server-specific ancillary data type, and is not associated with the IPv6 Advanced Socket API. It can be specified as ancillary data on sendmsg() for AF_INET and AF_INET6 sockets. For AF_INET sockets the level specified should be IPPROTO_IP; for AF_INET6 sockets the level specified should be IPPROTO_IPV6. For a detailed description of the function, see the programming interfaces in z/OS Communications Server: IP Programmer's Guide and Reference for providing classification data to be used in differentiated services policies.

Socket option to support ICMPv6 (IPPROTO_ICMPV6 level)

Table 27. Sockets options at the IPPROTO_ICMPV6 level

Socket options getsockopt() setsockopt()	Assembler Callable Services	C/C++ using Language Environment	REXX	Sockets Extended macro/ call
ICMP6_FILTER	N	Y	N	N

Use the following socket option to support ICMPv6 (IPPROTO_ICMPV6 level):

ICMP6_FILTER (used with RAW applications)

The ICMP6_FILTER socket option can be used by a RAW application to filter out ICMPv6 message types that it does not need to receive. There are many more ICMPv6 message types than ICMPv4 message types. ICMPv6 provides function comparable to ICMPv4 plus IGMPv4 and ARPv4 functionality. An application might be interested in receiving only a subset of the messages received for ICMPv6.

This option is enabled or disabled with a setsockopt(). The option value provides a 256-bit array of message types that should be filtered. To disable the option, the setsockopt() should be issued with an option length of 0. This causes the TCP/IP protocol stack's default filter to be in effect.

A getsockopt() with this option returns the value set by a setsockopt(). If a setsockopt() has not been done, the TCP/IP protocol stack's default filter is returned. For more information about default filtering, see "ICMP considerations" on page 110.

Table 28 on page 105 lists the macros that are provided in the Language Environment C/C++ environment to manipulate the filter value.

Table 28. Macros used to manipulate filter value

Macro	Description
void ICMP6_FILTER_SETPASSALL(struct icmp6_filter *);	Specifies that all ICMPv6 messages are passed to the application.

Table 28. Macros used to manipulate filter value (continued)

Macro	**Description**
void ICMP6_FILTER_SETBLOCKALL(struct icmp6_filter *);	Specifies that all ICMPv6 messages are blocked from being passed to the application.
void ICMP6_FILTER_SETPASS(int, struct icmp6_filter *);	ICMPv6 messages of type specified in int should be passed to the application.
void ICMP6_FILTER_SETBLOCK(int, struct icmp6_filter *);	ICMPv6 messages of type specified in int should not be passed to the application.
void ICMP6_FILTER_WILLPASS(int, const struct icmp6_filter *);	Returns true if the message type specified in int is passed to the application by the filter pointed to by the second argument.
void ICMP6_FILTER_WILLBLOCK(int, const struct icmp6_filter *);	Returns true if the message type specified in int is not passed to the application by the filter pointed to by the second argument.

Using ancillary data on sendmsg() and recvmsg()

The sendmsg() API is similar to other socket APIs, such as send() and write() that allow an application to send data, but also provides the capability of specifying ancillary data. Ancillary data allows applications to pass additional option data to the TCP/IP protocol stack along with the normal data that is sent to the IP network.

The recvmsg() API is similar to other socket APIs, such as recv() and read(), that allow an application to receive data, but also provides the capability of receiving ancillary data. Ancillary data allows the TCP/IP protocol stack to return additional option data to the application along with the normal data from the IP network.

These sendmsg() and recvmsg() API extensions are available only to applications using the following socket API libraries:

- z/OS IBM C/C++ sockets with the z/OS Language Environment. For more information about these APIs, see z/OS XL C/C++ Runtime Library Reference.
- z/OS UNIX Assembler Callable services socket APIs. For more information about these APIs, see z/OS UNIX System Services Programming: Assembler Callable Services Reference.

For the sendmsg() and recvmsg() APIs most parameters are passed in a message header input parameter. The mapping for the message header is defined in socket.h for C/C++ and in the BPXYMSGH macro for users of the z/OS UNIX Assembler Callable services. For simplicity, only the C/C++ version of the data structures is shown in the following code example:

```
struct msghdr {
    void             *msg_name;           /* optional address          */
    size_t           msg_namelen;         /* size of address           */
    struct           iovec *msg_iov;      /* scatter/gather array      */
    int              msg_iovlen;          /* # elements in msg_iov     */
    void             *msg_control;        /* ancillary data            */
    size_t           msg_controllen;      /* ancillary data length     */
    int              msg_flags;           /* flags on received msg     */
 };
```

Notes:

- The msg_name and msg_namelen parameters are used to specify the destination sockaddr on a sendmsg(). On a recvmsg() the msg_name and msg_namelen parameters are used to return the remote sockaddr to the application.

- Data to be sent using sendmsg() needs to be described in the msg_iov structure. On recvmsg() the received data is described in the msg_iov structure.
- The address of the ancillary data is passed in the msg_control field.
- The length of the ancillary data is passed in msg_controllen. Note that if multiple ancillary data sections are being passed, this length should reflect the total length of ancillary data sections.
- msg_flags is not applicable for sendmsg().

The msg_control parameter points to the ancillary data. This msg_control pointer points to the following structure (C/C++ example shown) that describes the ancillary data (also defined in socket.h and BPXYMSGH):

```
struct cmsghdr  {
      size_t   cmsg_len;       /* data byte count includes hdr */
      int      cmsg_level;     /* originating protocol         */
      int      cmsg_type;      /* protocol-specific type       */
     /* followed by u_char    cmsg_data[]; */
   };
```

Guidelines:

- The cmsg_len should be set to the length of the cmsghdr plus the length of all ancillary data that follows immediately after the cmsghdr. This is represented by the commented out cmsg_data field.
- The cmsg_level should be set to the option level (for example, IPPROTO_IPV6).
- The cmsg_type should be set to the option name (for example, IPV6_USE_MIN_MTU).

Interactions between socket options and ancillary data

This topic describes interactions between socket options and ancillary data, including hop limits.

Hop limit options

The IPv6 header contains a hop limit field that controls the number of hops over which a datagram can be sent before being discarded. This is similar to the TTL field in the IPv4 header. An application can influence the value of the hop limit field using the following options:

- IPV6_UNICAST_HOPS socket option (hop limit value to be used for unicast packets on a socket)
- IPV6_MULTICAST_HOPS socket option (hop limit value to be used for multicast packets on a socket)
- IPV6_HOPLIMIT ancillary data option on sendmsg() (hop limit value to be used for single packet)

The hop limit value can also be influenced by a router advertised hop limit, as well as the globally configured HOPLIMIT parameter value on the IPCONFIG6 statement.

For a unicast packet, the following precedence order is used to determine a packet's hop limit value:

1. If IPV6_HOPLIMIT ancillary data is specified on sendmsg(), use its value.
2. If the IPV6_UNICAST_HOPS socket option is set, use its value.
3. If a router advertised hop limit is known, use its value.
4. If there is a globally configured IPv6 hop limit, use its value.
5. Use the IPv6 default unicast hop limit, 255.

For a multicast packet, the following precedence order is used to determine the packet's hop limit value:

1. If IPV6_HOPLIMIT ancillary data is specified on sendmsg(), use its value.
2. If the IPV6_MULTICAST_HOPS socket option is set, use its value.
3. Use the IPv6 default multicast hop limit, 1.

Options for setting the source address

A UDP or RAW application can influence the setting of the source address with the bind() IPv6 address or with the IPV6_PKTINFO option.

The following precedence order is used to determine the source IP address for a packet:

1. If IPV6_PKTINFO ancillary data is specified on sendmsg() with a nonzero source IP address, use its value. If the IPV6_PKTINFO ancillary data is specified with a length of 0 or with a zero source IP address, go to step 3.
2. If the IPV6_PKTINFO socket option is set and contains a nonzero source IP address, use its value.
3. If the application bound the socket to a specific address, use the Bind address.
4. The TCP/IP protocol stack selects a source address.

Options for specifying the outgoing interface

A UDP or RAW application can influence the outgoing interface for a packet with the IPV6_PKTINFO option, the IPV6_NEXTHOP option, or the IPV6_MULTICAST_IF option. The scope ID field in the send operation's destination sockaddr can also affect the outgoing interface. The options field contains an interface index. The scope ID field contains a zone index.

When UDP and RAW applications respond to a peer, the applications use the sockaddr_in6 structure that they received, and they should not set the scope ID field to zero. When sending an unsolicited packet (for example, not responding to one that was received), the scope ID field should be zero. UDP and RAW applications should use the IPV6_PKTINFO, IPV6_NEXTHOP, or IPV6_MULTICAST_IF options to select the outgoing interfaces. Alternatively, if the sockaddr_in6 structure is created by the resolver using a getaddrinfo call, UDP and RAW applications can specify scope information in the getaddrinfo call; the scope ID field will be set appropriately by the resolver. See "Scope information about getaddrinfo calls" on page 77 for further information.

The following precedence order is used to determine the outgoing interface for a packet:

1. If the send operation specifies a destination sockaddr structure with a scope ID, then the scope ID is used if valid (note that a scope ID should be provided with a link-local address only).
2. If IPV6_PKTINFO ancillary data is specified on sendmsg() with a nonzero interface index, use its value. If the IPV6_PKTINFO ancillary data is specified with a length of 0 or with an interface index of 0, then skip to rule 4.
3. If the IPV6_PKTINFO socket option is set and contains a nonzero interface index, use its value.
4. If this is a multicast packet and the IPV6_MULTICAST_IF socket option is set, use its value.
5. If IPV6_NEXTHOP ancillary data is specified on sendmsg() with a nonzero value, use the stack routing table to determine the interface to the next hop address. If the IPV6_NEXTHOP ancillary data is specified with a length of 0, go to step 7.
6. If the IPV6_NEXTHOP socket option is set and contains a nonzero value, use the stack routing table to determine the interface to the next hop address.
7. The TCP/IP protocol stack uses the routing table to determine the interface to the destination IP address.

An application should provide outgoing interface information using only one method, or the application must ensure that the various specifications all indicate the same outgoing interface. If conflicting outgoing interface specifications are provided, the packet is discarded by the stack. For example, if scope information for the resolved destination host name specifies interface-1 and IPV6_PKTINFO ancillary data specifies interface-2, then the packet is discarded.

RAW sockets

Consider the following factors for RAW sockets use:

- An application (for example, PING) can send and receive ICMPv6 messages.

- An application can send and receive datagrams with an IP protocol that the TCP/IP stack does not support.

The external behavior of IPv6 RAW sockets differs significantly from the external behavior of IPv4 RAW sockets, specifically with regards to the following items:

- RAW protocol values allowed
- Application visibility of IP headers
- ICMP considerations
- Checksum of data

RAW protocol values

Protocol values 0, 41, 43, 44, 50, 51, 59 and 60 are not allowed because they conflict with the following IPv6 extension header types:

- IPPROTO_HOPOPTS (0)
- IPPROTO_IPV6 (41)
- IPPROTO_ROUTING (43)
- IPPROTO_FRAGMENT (44)
- IPPROTO_ESP (50)
- IPPROTO_AH (51)
- IPPROTO_NONE (59)
- IPPROTO_DSTOPTS (60)

Of the RAW protocol values listed, only the following values correspond to well-known IPv4 RAW protocols:

- IPPROTO_ESP (50)
- IPPROTO_AH (51)

Application visibility of IP headers

Applications do not see IP headers of incoming datagrams and cannot provide IP headers with outgoing datagrams.

IPv6 RAW applications can get or set selected IP header information for incoming and outgoing datagrams by way of socket options and ancillary data as follows:

- Applications can set the IPV6_RECVHOPLIMIT socket option in order to get the hop limit for incoming datagrams in ancillary data. By default, this socket option is set to off.
- Applications can set the IPV6_RECVPKTINFO socket option in order to get the destination IP address and interface identifier for incoming datagrams in ancillary data. By default, this socket option is set to off.
- Applications can set the IPV6_RECVRTHDR socket option in order to get the routing header for incoming datagrams in ancillary data. By default, this socket option is set to off.
- Applications can set the IPV6_RECVHOPOPTS socket option in order to get the hop-by-hop options for incoming datagrams in ancillary data. By default, this socket option is set to off.
- Applications can set the IPV6_RECVDSTOPTS socket option in order to get the destination options for incoming datagrams in ancillary data. By default, this socket option is set to off.
- Applications can set the IPV6_RECVTCLASS socket option in order to get the traffic class for incoming datagrams in ancillary data. By default, this socket option is set to off.
- Applications can set the IPV6_UNICAST_HOPS socket option in order to set the hop limit for outgoing unicast datagrams. By default, this socket option is set to off and the configured maximum hop limit or the default hop limit is used.

- Applications can set the IPV6_MULTICAST_HOPS socket option in order to set the hop limit for outgoing multicast datagrams. By default, this socket option is set to off and a hop limit of 1 is used.
- Applications can use the IPV6_HOPLIMIT ancillary data option to set the hop limit for an outgoing datagram.
- Applications can use the IPV6_PKTINFO socket option and ancillary data option to set the source address and interface identifier for outgoing datagrams. By default, the socket option is set to off.
- Applications can use the IPV6_NEXTHOP socket option and ancillary data option to set the next hop address for outgoing datagrams. By default, the socket option is set to off.
- Applications can use the IPV6_RTHDR socket option and ancillary data option to set the routing header for outgoing datagrams. By default, the socket option is set to off.
- Applications can use the IPV6_HOPOPTS socket option and ancillary data option to set the hop-by-hop options for outgoing datagrams. By default, the socket option is set to off.
- Applications can use the IPV6_DSTOPTS socket option and ancillary data option to set the destination options (that get examined by the host at the final destination) for outgoing datagrams. By default, the socket option is set to off.
- Applications can use the IPV6_RTHDRDSTOPTS socket option and ancillary data option to set the destination options (that get examined by every host that appears in the routing header) for outgoing datagrams. By default, the socket option is set to off.
- Applications can use the IPV6_TCLASS socket option and ancillary data option to set the traffic class for outgoing datagrams. By default, the socket option is set to off.

ICMP considerations

IPv6 RAW ICMPv6 applications can set the ICMP6_FILTER socket option to specify which ICMPv6 message types the socket receives. By default, the following message types are blocked (are not received):

- ICMP_ECHO
- ICMP_TSTAMP
- ICMP_IREQ
- ICMP_MASKREQ
- ICMP6_ECHO_REQUEST
- MLD_LISTENER_QUERY
- MLD_LISTENER_REPORT
- MLD_LISTENER_REDUCTION
- ND_ROUTER_SOLICIT
- ND_ROUTER_ADVERT
- ND_NEIGHBOR_SOLICIT
- ND_NEIGHBOR_ADVERT
- ND_REDIRECT

Checksum of data

IPv6 RAW applications can set the IPV6_CHECKSUM socket option in order to have TCP/IP calculate checksums for outgoing datagrams and verify checksums for incoming datagrams. By default, this socket option is set to off.

Chapter 10. Advanced concepts and topics

This topic explains some of the advanced concepts and ideas for IPv6 implementation and includes the following topics:

Tunneling

When IPv6 or IPv6/IPv4 systems are separated from other similar systems that they want to communicate with by IPv4 networks, then IPv6 packets must be tunneled through the IPv4 network. IPv6 packets are tunneled over IPv4 very simply: the IPv6 packet is encapsulated in an IPv4 datagram, or in other words, a complete IPv4 header is added to the IPv6 packet. The presence of the IPv6 packet within the IPv4 datagram is indicated by a protocol value of 41 in the IPv4 header.

Restriction: z/OS Communications Server cannot function as an endpoint for this type of tunnel.

While there are many tunneling protocols that can be used, all share the following common features and processing characteristics:

- The source tunnel endpoint determines that an IPv6 packet needs to be tunneled over an IPv4 network. This depends on the tunneling protocol that is used. After this decision is made, the source tunnel endpoint adds an IPv4 header to the IPv6 packet. The protocol value in the IPv4 header is set to 41. This indicates that this is an IPv6 over IPv4 tunnel packet. The source and destination addresses in the IPv4 header are set based on the tunneling protocol that is used.
- At the destination tunnel endpoint, the IPv4 layer receives the IPv4 packet (or packets, if the IPv4 datagram was fragmented). The IPv4 layer processes the datagram in the normal way, reassembling fragments if necessary, and records the protocol value of 41 in the IPv4 header. IPv4 security checks are made, and the IPv4 header is removed, leaving the original IPv6 packet. The IPv6 packet is processed as normal.

Figure 16 on page 112 shows a subset of the available tunneling protocols, with descriptions of the more prevalent protocols. Others exist or are in the process of being defined. Select one that is appropriate for your environment.

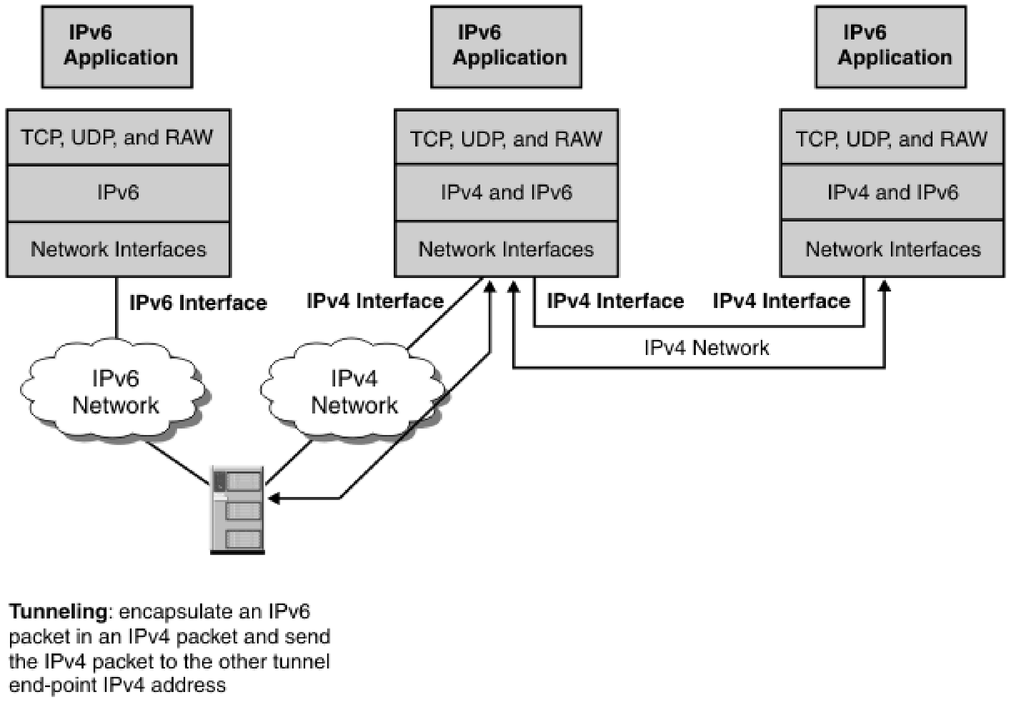


Figure 16. Tunneling

Configured tunnels

Configured tunneling refers to IPv6 over IPv4 tunneling, where the IPv4 tunnel endpoint address is determined by configuration information for the encapsulating node. The tunnels can be unidirectional or bidirectional. Bidirectional configured tunnels act similarly as virtual point-to-point links. For each tunnel, the encapsulating node must store the tunnel endpoint address. When an IPv6 packet is transmitted over a tunnel, the tunnel endpoint address configured for that tunnel is used as the destination address for the encapsulating IPv4 header.

Routing information for the encapsulating node usually determines which packets to tunnel. This is typically done by way of a routing table, which directs packets based on their destination address using the prefix mask and match technique.

Configured tunnels can be host-host, host-router, or router-router. Host-host tunnels allow two IPv6/IPv4 nodes to send IPv6 packets directly to one another without going through an intermediate IPv6 router. This can be useful if the applications need to take advantage of IPv6 features that are not available in IPv4.

An IPv6/IPv4 host that is connected to datalinks with no IPv6 routers can use a configured tunnel to reach an IPv6 router. This tunnel allows the host to communicate with the rest of the IPv6 Internet. If the IPv4 address of an IPv6/IPv4 router bordering the IPv6 backbone is known, this can be used as the tunnel endpoint address, and can be used as an IPv6 default route. This default route is used only if a more specific route is not known.

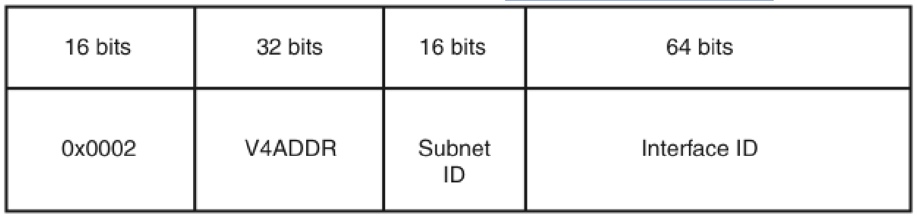
Configured tunnels can also be used between routers, allowing isolated IPv6 networks to be connected by way of an IPv4 backbone. This connectivity can be accomplished by arranging tunnels directly with each IPv6 site to which connectivity is needed, but more typically it is done by arranging a tunnel into a larger IPv6 routing infrastructure that can guarantee connectivity to all IPv6 user site networks. One example of this type of IPv6 routing infrastructure is the 6bone.

When using configured tunnels, a peering relationship must be established between the two IPv6 sites. This requires establishing a technical relationship with the peer and working through the various low-level details of how to configure tunnels between the two sites, including answering questions such as what peering protocol is used (presumably, an IPv6-capable version of BGP4).

6to4 addresses

The IANA has permanently assigned one 13-bit IPv6 Top Level Aggregator (TLA) identifier under the IPv6 Format Prefix 001 for the 6to4 scheme. Its numeric value is 0x2002; that is, it is 2002::/16 when expressed as an IPv6 address prefix.

The format for a 6to4 address is shown in Figure 17 on page 113:



16 bits	32 bits	16 bits	64 bits
0x0002	V4ADDR	Subnet ID	Interface ID

Figure 17. 6to4 address format

Thus, this prefix has the same format as normal /48 prefixes assigned according to other aggregatable global unicast addresses. It can be abbreviated as 2002:V4ADDR::/48. Within the subscriber site it can be used like any other valid IPv6 prefix, for example, for automated address assignment and discovery for native IPv6 routing, or for the 6over4 mechanism.

6to4 provides a mechanism to allow isolated IPv6 domains, attached to a wide area network with no native IPv6 support, to communicate with other such IPv6 domains with minimal configuration. The idea is to embed IPv4 tunnel addresses into the IPv6 prefixes so that any domain border router can automatically discover tunnel endpoints for outbound IPv6 traffic.

The 6to4 transition mechanism advertises a site's IPv4 tunnel endpoint (to be used for a dynamic tunnel) in a special external routing prefix for that site. When one site tries to reach another site, it discovers the 6to4 tunnel endpoint from a DNS name to address lookup and use a dynamically built tunnel from site to site for communication. The tunnels are transient in that there is no state maintained for them, lasting only as long as a specified transaction uses the path.

A 6to4 site identifies one or more routers to run as a dual-mode stack and to act as a 6to4 router. A globally routable IPv4 address is assigned to the 6to4 router. The 6to4 prefix, which has the 6to4 router's IPv4 address embedded within it, is then advertised by way of the Neighbor Discovery protocol to the 6to4 site, and this prefix is used by hosts within the site to generate a global IPv6 address.

When one IPv6-enabled host at a 6to4 site tries to access an IPv6-enabled host by domain name at another 6to4 site, the DNS returns the IPv6 IP address for that host. The requesting host sends a packet to its nearest router, eventually reaching a site's 6to4 router. When the site's 6to4 router receives the packet and sees that it must send the packet to another site, and the next hop destination prefix is a 2002:://16 prefix, the IPv6 packet is encapsulated as described in "Tunneling" on page 111. The source IPv4 address is the one in the requesting site's 6to4 prefix (which is the IPv4 address of an outgoing interface for one of the site's 6to4 routers) and the destination IPv4 address is the one in the next hop destination 6to4 prefix of the IPv6 packet. When the destination site's 6to4 router receives the IPv4 packet, the IPv4 header is removed, leaving the original IPv6 packet for local forwarding.

6over4 tunnels

The Interface Identifier of an IPv4 interface using 6over4 is the 32-bit IPv4 address of that interface, padded to the left with 0s and is 64 bits in length. Note that the Universal/Local bit is 0, indicating that the Interface Identifier is not globally unique. When the host has more than one IPv4 address in use on the physical interface concerned, an administrative choice of one of these IPv4 addresses is made.

The IPv6 Link-local address for an IPv4 virtual interface is formed by appending the Interface Identifier, as defined above, to the prefix FE80::/64.

3 bits	45 bits	16 bits	32 bits	32 bits
001	Network	Subnet	0..........0	IPv4 address

Figure 18. 6over4 address format

Global unicast addresses are generated by adding a 64-bit prefix to the 6over4 Interface Identifier. These prefixes can be learned in any of the normal ways, for example, as part of stateless address autoconfiguration or by way of manual configuration.

6over4 is a transition mechanism which allows isolated IPv6 hosts, located on a physical link which has no directly connected IPv6 router, to use an IPv4 multicast domain as their virtual local link. A 6over4 host uses an IPv4 address for the interface in the creation of the IPv6 interface ID, placing the 32-bit IPv4 address in the low-order bits and padding to the left with 0's for a total of 64 bits. The IPv6 prefix used is the normal IPv6 prefix, and can be manually configured or dynamically learned by way of Stateless Address Autoconfiguration.

Because 6over4 creates a virtual link using IPv4 multicast, at least one IPv6 router using the same method must be connected to the same IPv4 multicast domain if IPv6 routing to other links is required.

When encapsulating the IPv6 packet, the source IP address for the IPv4 packet is an IPv4 address from the sending interface of the 6over4 host. The destination IPv4 address is the low-order 32 bits of the IPv6 address of the next-hop for the packet. Note that the final destination of the packet does not need to be a 6over4 host, although it might be one.

Application migration and coexistence overview

Many IPv6 stacks support both IPv4 and IPv6 interfaces and are capable of receiving and sending native IPv4 and IPv6 packets over the corresponding interfaces. This type of TCP/IP stack is generally referred to as a dual-mode stack IP node. This does not mean that there are two separate TCP/IP stacks running on this type of node. It means that the TCP/IP stack has built-in support for both IPv4 and IPv6. In this topic, the term dual-mode stack or IP node is a TCP/IP stack that supports both IPv4 and IPv6 protocols.

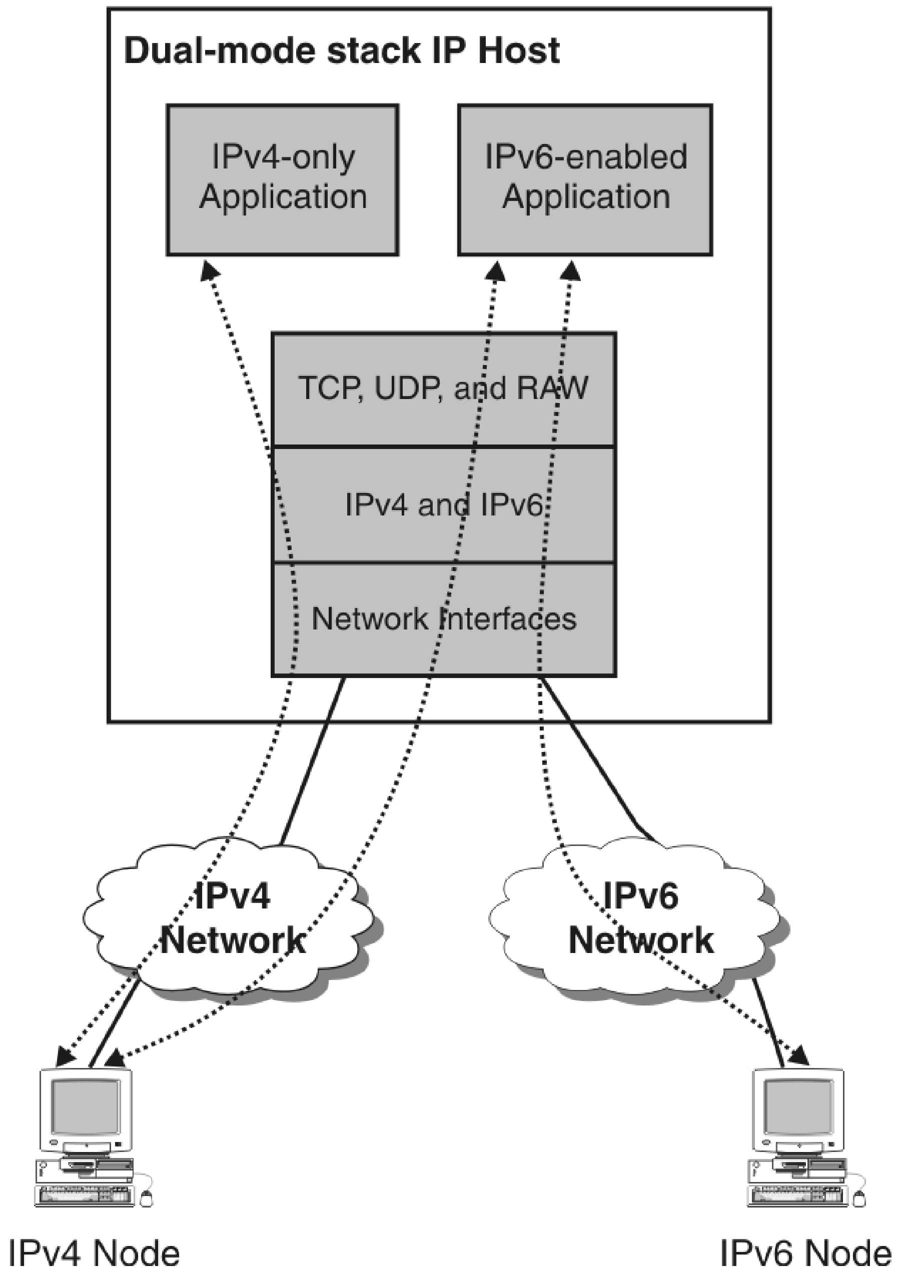


Figure 19. Dual-mode stack IP host

For a multihomed dual-mode IP host, it is a likely configuration that the host has both IPv4 and IPv6 interfaces over which requests for host-resident applications are received or sent. Older AF_INET applications can communicate using IPv4 addresses only. IPv6-enabled applications that use AF_INET6 sockets can communicate using both IPv4 and IPv6 addresses (on a dual-mode host). AF_INET and AF_INET6 applications are able to communicate with one another, but only using IPv4 addresses.

If the socket libraries on the IPv6-enabled host are updated to support IPv6 sockets (AF_INET6), applications can be IPv6 enabled. When an application on a dual-mode stack host is IPv6 enabled, the application is able to communicate with both IPv4 and IPv6 partners. This is true for both clients and server on a dual-mode stack host.

Table 29. Application communication on a dual-mode host

	IPv4-only	**IPv6-enabled**
IPv4-only partner	Yes	Yes

Table 29. Application communication on a dual-mode host (continued)

	IPv4-only	IPv6-enabled
IPv6-only partner		Yes

IPv6-enabling both sockets libraries and applications on dual-mode hosts therefore becomes a migration concern. As soon as IPv6-only hosts are being deployed in a network, applications on those IPv6-only nodes cannot communicate with the IPv4-only applications on the dual-mode hosts, unless one of multiple migration technologies are implemented either on intermediate nodes in the network or directly on the dual-mode hosts.

Application migration approaches

The ultimate and preferred migration approach for applications that reside on a dual-mode TCP/IP host is to IPv6-enable the applications by migrating them from AF_INET sockets to AF_INET6 sockets.

There are multiple reasons why this approach is not always applicable, such as the following reasons:

- No access to the source code (vendor product, or source no longer available).
- The sockets API implementation does not yet (or never does) support IPv6.
- Resource availability or prioritization dictates a phased IPv6-enabling where not all applications can be available in an IPv6-enabled version at the same point in time where the stack is IPv6-capable.

For those applications that are not or cannot be IPv6 enabled, an alternative migration strategy is needed. The IETF has identified multiple approaches as summarized in draft RFC, *An Overview of the Introduction of IPv6 in the Internet*.

Some of the technologies that are defined by the IETF are supposed to be implemented on intermediate nodes that route traffic between IPv4 and IPv6 network segments. Other technologies are intended for implementation on the dual-mode IP nodes themselves.

Translation mechanisms

This topic provides an introduction to a few transition mechanisms that can be used when migrating to an IPv6 network.

The key to successful adoption and deployment of IPv6 is the transition from the installed IPv4 base. The goal of all transition strategies is to facilitate the partial and incremental upgrade of hosts, servers, routers, and network infrastructure. There are many possible approaches, and some of the more likely approaches are described below. The transition strategy a company chooses to take varies based on the particular needs of that company.

Several migration issues must be addressed when the backbone routing protocol is IPv4. First, a mechanism is needed to allow communication between islands of IPv6 networks that are interconnected only using the IPv4 backbone. Tunneling of IPv6 packets over the IPv4 network can be used to connect the clouds. Second, end-to-end communication between IPv4 and IPv6 applications must be enabled. Several approaches to accomplish this exist; Application Layer Gateways, NAT-PT, and Bump-in-the-Stack are all possibilities. During the migration phase, it is likely that a combination of one, multiple, or all of these transition mechanisms can be used.

Application Layer Gateways (ALGs) enable IPv6-only applications to communicate with an IPv4-only peer. Using an ALG, the client connects to the ALG by using its native protocol (IPv4 or IPv6) and the ALG connects to the server by using the other protocol (IPv6 or IPv4).

SOCKS gateway

A SOCKS gateway is a method of providing an ALG. The SOCKS64 implementation works as a SOCKS server that relays communication between IPv4 and IPv6 flows. Servers do not require any changes, but client applications (or the stack where the client applications reside) need to be socks-enabled to be able to reach out through a SOCKS64 server to an IPv6-only partner.

Proxy

Protocol translation involves converting IPv4 packets into IPv6 packets and IPv6 packets into IPv4 packets. This translation typically involves some form of network address translation (NAT) in addition to the protocol translation (PT) function. It might run in a specialized node between an IPv4 network and an IPv6 network, or it might run in the host that owns the IPv4 application.

Protocol Translation is useful when devices need to communicate but are not using the same protocol, allowing IPv6-only devices to communicate with IPv4-only devices. However, the following issues make a less-than ideal solution:

- Protocol translation is not foolproof. It is difficult to determine how long to keep the mappings between the real IPv6 address and the locally mapped IPv4 address available. An address can be reused before all servers have stopped accessing the address.
- Some applications might use the remote IP address as a means of performing a security check. Unless AH or an IPSec tunnel is used, then this method is not foolproof, but it is still done. If the IPv4 address is a locally mapped address, any checks such as this are broken.
- Displays and traces of the remote IP address are meaningless. Today, many applications generate messages, traces, and so on containing the IP address of the remote client.
- All DNS queries for the IPv4-mapped address must flow through the node that performed the NAT function. The DNS resolver or name server at this node, as well as the TCP/IP stack, must maintain a mapping between the IPv4 address and IPv6 address.
- Not all IPv6 protocols have IPv4 equivalents and not all IPv4 protocols have IPv6 equivalents. It might not be possible to translate the contents of an IPv4 packet into an equivalent IPv6 packet or the contents of an IPv6 packet into an equivalent IPv4 packet.

Stateless IP/ICMP Translation Algorithm

This algorithm translates between IPv4 and IPv6 packet headers (including ICMP headers) in separate translator boxes in the network without requiring any per-connection state in those boxes. Stateless IP/ICMP Translation Algorithm (SIIT) can be used as part of a solution that allows IPv6 hosts, which do not have permanently assigned IPv4 addresses, to communicate with IPv4-only hosts.

Network address translation - protocol translation

Protocol translation can occur at a specialized node that resides between IPv4 and IPv6 networks. This node is typically referred to as a Network address translation - protocol translation (NAT-PT) device because it must translate between the IPv4 and IPv6 addresses, as well as between the IPv4 and IPv6 protocols.

An NAT-PT node plays a similar role to an ALG. Both nodes allow IPv4-only applications to communicate with IPv6-only peers, and both reside in similar places in the network. However, each takes a different approach to accomplish a similar goal.

SOCKS64 is a proxy solution and requires client applications to be updated to use SOCKS64. NAT-PT is not a proxy and requires no changes to either the client or server. Based solely on this, NAT-PT might appear to be a superior solution. However, because of the limitations of NAT-PT and familiarity with SOCKS, it is more likely that SOCKS64 is used to allow IPv4-only applications to communicate with IPv6-only peers.

Appendix A. IPv6 support tables

This appendix contains the IPv6 support tables and includes the following topics:

- "Supported IPv6 standards" on page 119
- "Application support of scope information specified on host name or IP address" on page 120
- "z/OS-specific features" on page 121
- "Applications not enabled for IPv6" on page 124

Supported IPv6 standards

Table 30 on page 119 lists the supported IPv6 standards. RFCs are not implemented in their entirety.

Table 30. Supported IPv6 standards

Standard	RFC or Internet Draft
DNS Extensions to support IP version 6	1886
Path MTU discovery	1981
RIPng for IPv6	2080
An IPv6 Aggregatable Global Unicast Address Format	2374
FTP Extensions for IPv6 and NATs	2428
Internet Protocol, Version 6 (IPv6) Specification	2460
Neighbor discovery for IP Version 6 (IPv6)	2461
IPv6 Stateless Address Autoconfiguration	2462
Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification	2463
Transmission of IPv6 Packets over Ethernet Networks	2464
Multicast Listener Discovery (MLD) for IPv6	2710
IPv6 Router Alert Option	2711
OSPF for IPv6	2740
DNS Extensions to Support IPv6 Address Aggregation and Renumbering	2874
Default Address Selection for Internet Protocol Version 6 (IPv6)	3484
Basic Socket Interface Extensions for IPv6	3493
Internet Protocol Version 6 (IPv6) Addressing Architecture	3513
Advanced Sockets Application Programming Interface (API) for IPv6	3542
Multicast Listener Discovery Version 2 (MLDv2) for IPv6	3810
Socket Interface Extensions for Multicast Source Filters	3678

Table 30. Supported IPv6 standards (continued)

Standard	RFC or Internet Draft
IPv6 Scoped Address Architecture	4007
IPv6 Socket for Source Address Selection	5014

Application support of scope information specified on host name or IP address

Table 31 on page 120 lists the applications that accept scope information (for example, interface name or interface index) as part of a user-specified or user-configured host name or IPv6 address. The topic of scope information is described in more detail in "Support for scope information" on page 51 .

Table 31. Application support for scope information

Application	Support level
FTP client	1. Scope information can be specified on host name or IPv6 address provided as command input. 2. Scope information can be specified on host name or IPv6 address provided as input on the OPEN subcommand. 3. Scope information can be specified on host names or IPv6 addresses in NETRC configuration information
FTP server	Scope information can appear in SMF records or in banner lines.
MVRSHD	Scope information can be specified on host names coded in *userid*.RHOSTS.DATA configuration file
Ping	1. Scope information can be specified on host name or IPv6 address representing the destination host. 2. Scope information cannot be specified as part of the source IP address operand. 3. Scope information cannot be specified as part of the interface operand
REXEC/OREXEC	Scope information can be specified on host name or IPv6 address provided on command input.
RSH/ORSH	Scope information can be specified on host name or IPv6 address provided on command input.
Syslogd	1. Scope information can appear as part of host name information generated as syslog output. 2. Scope information cannot be specified as part of selector host name information

Table 31. Application support for scope information (continued)

Application	Support level
Traceroute	1. Scope information can be specified on host name or IPv6 address representing the destination host. 2. Scope information cannot be specified as part of the source IP address operand. 3. Scope information cannot be specified as part of the interface operand

z/OS-specific features

These tables summarize z/OS TCP/IP features and the level of support that is provided in an IPv6 network. In the future, more features are projected for IPv6 support in subsequent releases of the z/OS Communications Server.

Table 32 on page 121 lists the link-layer device support.

Table 32. Link-layer device support

Link-layer device support	IPv4 support	IPv6 support	Comments
OSA-Express in QDIO mode	Y	Y	Fast and Gigabit Ethernet support for IPv6 traffic is configured by way of an INTERFACE statement of type IPAQENET6.
CTC	Y	N	None
LCS	Y	N	None
CLAW	Y	N	None
CDLC (3745/3746)	Y	N	None
SNALINK LU0 and LU6.2	Y	N	None
X.25 NPSI	Y	N	None
NSC HyperChannel	Y	N	None
MPC Point-Point	Y	Y	Support is configured by way of an INTERFACE statement of type MPCPTP6.
ATM	Y	N	None
HiperSockets	Y	Y	Support is configured by way of an INTERFACE statement of type IPAQIDIO6 or dynamically configured by way of the IPCONFIG6 DYNAMICXCF statement.
XCF	Y	Y	Support is configured by way of an INTERFACE statement of type MPCPTP6 or dynamically configured by way of the IPCONFIG6 DYNAMICXCF statement.

Table 33 on page 122 lists virtual IP Addressing support.

Table 33. Virtual IP Addressing support

Virtual IP Addressing support	IPv4 support	IPv6 support	Comments
Virtual Device/Interface Configuration for static VIPA	Y	Y	None

All sysplex functions support IPv6 except for the function that is listed in Table 34 on page 122.

Table 34. Sysplex support

Sysplex support	IPv4 support	IPv6 support	Comments
Sysplex distributor integration with Cisco MNLB	Y	N	None

Table 35 on page 122 lists IP routing functions.

Table 35. IP routing functions

IP routing functions	IPv4 support	IPv6 support	Comments
Dynamic Routing - OSPF	Y	Y	None
Dynamic Routing - RIP	Y	Y	None
Multipath Routing Groups	Y	Y	None
Policy-based Routing	Y	Y	None
Static Route Configuration by way of BEGINROUTES statement	Y	Y	None

Table 36 on page 122 lists miscellaneous IP/IF-layer functions.

Table 36. Miscellaneous IP/IF-layer functions

Miscellaneous IP/IF-layer functions	IPv4 support	IPv6 support	Comments
Path MTU Discovery	Y	Y	None
Configurable Device or Interface Recovery Interval	Y	Y	None
Link-Layer Address Resolution	Y	Y	None
ARP/Neighbor Cache PURGE Capability	Y	Y	None
Datagram Forwarding Enable/ Disable	Y	Y	None
HiperSockets accelerator	Y	N	None
QDIO accelerator	Y	N	None
Checksum offload	Y	Y	Based on OSA-Express support
Segmentation offload	Y	Y	Based on OSA-Express support

Table 36. Miscellaneous IP/IF-layer functions (continued)

Miscellaneous IP/IF-layer functions	**IPv4 support**	**IPv6 support**	**Comments**
QDIO inbound workload queueing	Y	Y	Based on OSA-Express support

Table 37 on page 123 lists transport-layer functions.

Table 37. Transport-layer functions

Transport-layer functions	**IPv4 support**	**IPv6 support**	**Comments**
Fast Response Cache Accelerator	Y	Y	None
Enterprise Extender	Y	Y	IPv6 Enterprise Extender support requires a virtual IP address that is configured by way of an INTERFACE statement of type VIRTUAL6 and IUTSAMEH configured by way of an INTERFACE statement of type MPCPTP6 or dynamically configured by way of IPCONFIG6 DYNAMICXCF.
Server-BIND Control	Y	Y	None
UDP Checksum Disablement Option	Y	N	None

Table 38 on page 123 lists network management and accounting functions.

Table 38. Network management and accounting functions

Network management and accounting Functions	**IPv4 support**	**IPv6 support**	**Comments**
SNMP	Y	Y	None
SNMP agent	Y	Y	None
TCP/IP subagent	Y	Y	No IPv6 UDP support
Network SLAPM2 subagent	Y	Y	None
Distributed Protocol Interface	Y	Y	None
OMPROUTE subagent	Y	N	None
Trap forwarder daemon	Y	Y	None
Policy-Based Networking	Y	Y	None
SMF	Y	Y	None
TN3270 subagent	Y	Y	None

Table 39 on page 124 lists security functions.

Table 39. Security functions

Security functions	**IPv4 support**	**IPv6 support**	**Comments**
IPSec	Y	Y	None
IP filtering	Y	Y	None
IKE daemon	Y	Y	None
NAT traversal	Y	N	None
Network Access Control	Y	Y	None
Stack and Port Access Control	Y	Y	None
Application Transparent TLS	Y	Y	None
Intrusion Detection Services	Y	Y	None

Applications not enabled for IPv6

Some applications are not enabled for IPv6. These applications are listed in Table 40 on page 124, Table 41 on page 124, and Table 42 on page 125.

Table 40. Server applications not enabled for IPv6

Server applications	**IPv4 support**	**IPv6 support**
SMTPPROC/NJE server	Y	N
Rlogind server	Y	N
MVS Miscellaneous server	Y	N
Popper	Y	N
MVS LPD server	Y	N
TIMED server	Y	N
NCS LLBD and GLBD servers	Y	N
ONC/RPC MVS portmapper	Y	N
ONC/RPC UNIX portmapper	Y	N
NPF	Y	N
RSVP daemon	Y	N

Table 41. Client applications not enabled for IPv6

Client applications	**IPv4 support**	**IPv6 support**
TSO TELNET client	Y	N
TSO LPR client	Y	N

Table 42. Command-type applications not enabled for IPv6

Command-type applications	**IPv4 support**	**IPv6 support**
TSO DIG	Y	N
TSO LPRM	Y	N
TSO NSLOOKUP	Y	N
TSO RPCINFO	Y	N
UNIX **`dig`**	Y	N
UNIX **`host`**	Y	N
UNIX **`hostname`**	Y	N
UNIX **`rpcinfo`**	Y	N

Appendix B. Related protocol specifications

This appendix lists the related protocol specifications (RFCs) for TCP/IP. The Internet Protocol suite is still evolving through requests for comments (RFC). New protocols are being designed and implemented by researchers and are brought to the attention of the Internet community in the form of RFCs. Some of these protocols are so useful that they become recommended protocols. That is, all future implementations for TCP/IP are recommended to implement these particular functions or protocols. These become the *de facto* standards, on which the TCP/IP protocol suite is built.

RFCs are available at http://www.rfc-editor.org/rfc.html.

Draft RFCs that have been implemented in this and previous Communications Server releases are listed at the end of this topic.

Many features of TCP/IP Services are based on the following RFCs:

RFC	**Title and Author**
RFC 652	*Telnet output carriage-return disposition option* D. Crocker
RFC 653	*Telnet output horizontal tabstops option* D. Crocker
RFC 654	*Telnet output horizontal tab disposition option* D. Crocker
RFC 655	*Telnet output formfeed disposition option* D. Crocker
RFC 657	*Telnet output vertical tab disposition option* D. Crocker
RFC 658	*Telnet output linefeed disposition* D. Crocker
RFC 698	*Telnet extended ASCII option* T. Mock
RFC 726	*Remote Controlled Transmission and Echoing Telnet option* J. Postel, D. Crocker
RFC 727	*Telnet logout option* M.R. Crispin
RFC 732	*Telnet Data Entry Terminal option* J.D. Day
RFC 733	*Standard for the format of ARPA network text messages* D. Crocker, J. Vittal, K.T. Pogran, D.A. Henderson
RFC 734	*SUPDUP Protocol* M.R. Crispin
RFC 735	*Revised Telnet byte macro option* D. Crocker, R.H. Gumpertz
RFC 736	*Telnet SUPDUP option* M.R. Crispin
RFC 749	*Telnet SUPDUP—Output option* B. Greenberg
RFC 765	*File Transfer Protocol specification* J. Postel

RFC 768
User Datagram Protocol J. Postel

RFC 779
Telnet send-location option E. Killian

RFC 791
Internet Protocol J. Postel

RFC 792
Internet Control Message Protocol J. Postel

RFC 793
Transmission Control Protocol J. Postel

RFC 820
Assigned numbers J. Postel

RFC 823
DARPA Internet gateway R. Hinden, A. Sheltzer

RFC 826
Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware D. Plummer

RFC 854
Telnet Protocol Specification J. Postel, J. Reynolds

RFC 855
Telnet Option Specification J. Postel, J. Reynolds

RFC 856
Telnet Binary Transmission J. Postel, J. Reynolds

RFC 857
Telnet Echo Option J. Postel, J. Reynolds

RFC 858
Telnet Suppress Go Ahead Option J. Postel, J. Reynolds

RFC 859
Telnet Status Option J. Postel, J. Reynolds

RFC 860
Telnet Timing Mark Option J. Postel, J. Reynolds

RFC 861
Telnet Extended Options: List Option J. Postel, J. Reynolds

RFC 862
Echo Protocol J. Postel

RFC 863
Discard Protocol J. Postel

RFC 864
Character Generator Protocol J. Postel

RFC 865
Quote of the Day Protocol J. Postel

RFC 868
Time Protocol J. Postel, K. Harrenstien

RFC 877
Standard for the transmission of IP datagrams over public data networks J.T. Korb

RFC 883
Domain names: Implementation specification P.V. Mockapetris

RFC 884
Telnet terminal type option M. Solomon, E. Wimmers

RFC 885
Telnet end of record option J. Postel

RFC 894
Standard for the transmission of IP datagrams over Ethernet networks C. Hornig

RFC 896
Congestion control in IP/TCP internetworks J. Nagle

RFC 903
Reverse Address Resolution Protocol R. Finlayson, T. Mann, J. Mogul, M. Theimer

RFC 904
Exterior Gateway Protocol formal specification D. Mills

RFC 919
Broadcasting Internet Datagrams J. Mogul

RFC 922
Broadcasting Internet datagrams in the presence of subnets J. Mogul

RFC 927
TACACS user identification Telnet option B.A. Anderson

RFC 933
Output marking Telnet option S. Silverman

RFC 946
Telnet terminal location number option R. Nedved

RFC 950
Internet Standard Subnetting Procedure J. Mogul, J. Postel

RFC 952
DoD Internet host table specification K. Harrenstien, M. Stahl, E. Feinler

RFC 959
File Transfer Protocol J. Postel, J.K. Reynolds

RFC 961
Official ARPA-Internet protocols J.K. Reynolds, J. Postel

RFC 974
Mail routing and the domain system C. Partridge

RFC 1001
Protocol standard for a NetBIOS service on a TCP/UDP transport: Concepts and methods NetBios Working Group in the Defense Advanced Research Projects Agency, Internet Activities Board, End-to-End Services Task Force

RFC 1002
Protocol Standard for a NetBIOS service on a TCP/UDP transport: Detailed specifications NetBios Working Group in the Defense Advanced Research Projects Agency, Internet Activities Board, End-to-End Services Task Force

RFC 1006
ISO transport services on top of the TCP: Version 3 M.T. Rose, D.E. Cass

RFC 1009
Requirements for Internet gateways R. Braden, J. Postel

RFC 1011
Official Internet protocols J. Reynolds, J. Postel

RFC 1013
X Window System Protocol, version 11: Alpha update April 1987 R. Scheifler

RFC 1014
XDR: External Data Representation standard Sun Microsystems

RFC 1027
Using ARP to implement transparent subnet gateways S. Carl-Mitchell, J. Quarterman

RFC 1032
Domain administrators guide M. Stahl

RFC 1033
Domain administrators operations guide M. Lottor

RFC 1034
Domain names—concepts and facilities P.V. Mockapetris

RFC 1035
Domain names—implementation and specification P.V. Mockapetris

RFC 1038
Draft revised IP security option M. St. Johns

RFC 1041
Telnet 3270 regime option Y. Rekhter

RFC 1042
Standard for the transmission of IP datagrams over IEEE 802 networks J. Postel, J. Reynolds

RFC 1043
Telnet Data Entry Terminal option: DODIIS implementation A. Yasuda, T. Thompson

RFC 1044
Internet Protocol on Network System's HYPERchannel: Protocol specification K. Hardwick, J. Lekashman

RFC 1053
Telnet X.3 PAD option S. Levy, T. Jacobson

RFC 1055
Nonstandard for transmission of IP datagrams over serial lines: SLIP J. Romkey

RFC 1057
RPC: Remote Procedure Call Protocol Specification: Version 2 Sun Microsystems

RFC 1058
Routing Information Protocol C. Hedrick

RFC 1060
Assigned numbers J. Reynolds, J. Postel

RFC 1067
Simple Network Management Protocol J.D. Case, M. Fedor, M.L. Schoffstall, J. Davin

RFC 1071
Computing the Internet checksum R.T. Braden, D.A. Borman, C. Partridge

RFC 1072
TCP extensions for long-delay paths V. Jacobson, R.T. Braden

RFC 1073
Telnet window size option D. Waitzman

RFC 1079
Telnet terminal speed option C. Hedrick

RFC 1085
ISO presentation services on top of TCP/IP based internets M.T. Rose

RFC 1091
Telnet terminal-type option J. VanBokkelen

RFC 1094
NFS: Network File System Protocol specification Sun Microsystems

RFC 1096
Telnet X display location option G. Marcy

RFC 1101
DNS encoding of network names and other types P. Mockapetris

RFC 1112
Host extensions for IP multicasting S.E. Deering

RFC 1113
Privacy enhancement for Internet electronic mail: Part I — message encipherment and authentication procedures J. Linn

RFC 1118
Hitchhikers Guide to the Internet E. Krol

RFC 1122
Requirements for Internet Hosts—Communication Layers R. Braden, Ed.

RFC 1123
Requirements for Internet Hosts—Application and Support R. Braden, Ed.

RFC 1146
TCP alternate checksum options J. Zweig, C. Partridge

RFC 1155
Structure and identification of management information for TCP/IP-based internets M. Rose, K. McCloghrie

RFC 1156
Management Information Base for network management of TCP/IP-based internets K. McCloghrie, M. Rose

RFC 1157
Simple Network Management Protocol (SNMP) J. Case, M. Fedor, M. Schoffstall, J. Davin

RFC 1158
Management Information Base for network management of TCP/IP-based internets: MIB-II M. Rose

RFC 1166
Internet numbers S. Kirkpatrick, M.K. Stahl, M. Recker

RFC 1179
Line printer daemon protocol L. McLaughlin

RFC 1180
TCP/IP tutorial T. Socolofsky, C. Kale

RFC 1183
New DNS RR Definitions C.F. Everhart, L.A. Mamakos, R. Ullmann, P.V. Mockapetris

RFC 1184
Telnet Linemode Option D. Borman

RFC 1186
MD4 Message Digest Algorithm R.L. Rivest

RFC 1187
Bulk Table Retrieval with the SNMP M. Rose, K. McCloghrie, J. Davin

RFC 1188
Proposed Standard for the Transmission of IP Datagrams over FDDI Networks D. Katz

RFC 1190
Experimental Internet Stream Protocol: Version 2 (ST-II) C. Topolcic

RFC 1191
Path MTU discovery J. Mogul, S. Deering

RFC 1198
FYI on the X window system R. Scheifler

RFC 1207
FYI on Questions and Answers: Answers to commonly asked "experienced Internet user" questions G. Malkin, A. Marine, J. Reynolds

RFC 1208
Glossary of networking terms O. Jacobsen, D. Lynch

RFC 1213
Management Information Base for Network Management of TCP/IP-based internets: MIB-II K. McCloghrie, M.T. Rose

RFC 1215
Convention for defining traps for use with the SNMP M. Rose

RFC 1227
SNMP MUX protocol and MIB M.T. Rose

RFC 1228
SNMP-DPI: Simple Network Management Protocol Distributed Program Interface G. Carpenter, B. Wijnen

RFC 1229
Extensions to the generic-interface MIB K. McCloghrie

RFC 1230
IEEE 802.4 Token Bus MIB K. McCloghrie, R. Fox

RFC 1231
IEEE 802.5 Token Ring MIB K. McCloghrie, R. Fox, E. Decker

RFC 1236
IP to X.121 address mapping for DDN L. Morales, P. Hasse

RFC 1256
ICMP Router Discovery Messages S. Deering, Ed.

RFC 1267
Border Gateway Protocol 3 (BGP-3) K. Lougheed, Y. Rekhter

RFC 1268
Application of the Border Gateway Protocol in the Internet Y. Rekhter, P. Gross

RFC 1269
Definitions of Managed Objects for the Border Gateway Protocol: Version 3 S. Willis, J. Burruss

RFC 1270
SNMP Communications Services F. Kastenholz, ed.

RFC 1285
FDDI Management Information Base J. Case

RFC 1315
Management Information Base for Frame Relay DTEs C. Brown, F. Baker, C. Carvalho

RFC 1321
The MD5 Message-Digest Algorithm R. Rivest

RFC 1323
TCP Extensions for High Performance V. Jacobson, R. Braden, D. Borman

RFC 1325
FYI on Questions and Answers: Answers to Commonly Asked "New Internet User" Questions G. Malkin, A. Marine

RFC 1327
Mapping between X.400 (1988)/ISO 10021 and RFC 822 S. Hardcastle-Kille

RFC 1340
Assigned Numbers J. Reynolds, J. Postel

RFC 1344
Implications of MIME for Internet Mail Gateways N. Bornstein

RFC 1349
Type of Service in the Internet Protocol Suite P. Almquist

RFC 1351
SNMP Administrative Model J. Davin, J. Galvin, K. McCloghrie

RFC 1352
SNMP Security Protocols J. Galvin, K. McCloghrie, J. Davin

RFC 1353
Definitions of Managed Objects for Administration of SNMP Parties K. McCloghrie, J. Davin, J. Galvin

RFC 1354
IP Forwarding Table MIB F. Baker

RFC 1356
Multiprotocol Interconnect on X.25 and ISDN in the Packet Mode A. Malis, D. Robinson, R. Ullmann

RFC 1358
Charter of the Internet Architecture Board (IAB) L. Chapin

RFC 1363
A Proposed Flow Specification C. Partridge

RFC 1368
Definition of Managed Objects for IEEE 802.3 Repeater Devices D. McMaster, K. McCloghrie

RFC 1372
Telnet Remote Flow Control Option C. L. Hedrick, D. Borman

RFC 1374
IP and ARP on HIPPI J. Renwick, A. Nicholson

RFC 1381
SNMP MIB Extension for X.25 LAPB D. Throop, F. Baker

RFC 1382
SNMP MIB Extension for the X.25 Packet Layer D. Throop

RFC 1387
RIP Version 2 Protocol Analysis G. Malkin

RFC 1388
RIP Version 2 Carrying Additional Information G. Malkin

RFC 1389
RIP Version 2 MIB Extensions G. Malkin, F. Baker

RFC 1390
Transmission of IP and ARP over FDDI Networks D. Katz

RFC 1393
Traceroute Using an IP Option G. Malkin

RFC 1398
Definitions of Managed Objects for the Ethernet-Like Interface Types F. Kastenholz

RFC 1408
Telnet Environment Option D. Borman, Ed.

RFC 1413
Identification Protocol M. St. Johns

RFC 1416
Telnet Authentication Option D. Borman, ed.

RFC 1420
SNMP over IPX S. Bostock

RFC 1428
Transition of Internet Mail from Just-Send-8 to 8bit-SMTP/MIME G. Vaudreuil

RFC 1442
Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2) J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1443
Textual Conventions for version 2 of the Simple Network Management Protocol (SNMPv2) J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1445
Administrative Model for version 2 of the Simple Network Management Protocol (SNMPv2) J. Galvin, K. McCloghrie

RFC 1447
Party MIB for version 2 of the Simple Network Management Protocol (SNMPv2) K. McCloghrie, J. Galvin

RFC 1448
Protocol Operations for version 2 of the Simple Network Management Protocol (SNMPv2) J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1464
Using the Domain Name System to Store Arbitrary String Attributes R. Rosenbaum

RFC 1469
IP Multicast over Token-Ring Local Area Networks T. Pusateri

RFC 1483
Multiprotocol Encapsulation over ATM Adaptation Layer 5 Juha Heinanen

RFC 1514
Host Resources MIB P. Grillo, S. Waldbusser

RFC 1516
Definitions of Managed Objects for IEEE 802.3 Repeater Devices D. McMaster, K. McCloghrie

RFC 1521
MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies N. Borenstein, N. Freed

RFC 1535
A Security Problem and Proposed Correction With Widely Deployed DNS Software E. Gavron

RFC 1536
Common DNS Implementation Errors and Suggested Fixes A. Kumar, J. Postel, C. Neuman, P. Danzig, S. Miller

RFC 1537
Common DNS Data File Configuration Errors P. Beertema

RFC 1540
Internet Official Protocol Standards J. Postel

RFC 1571
Telnet Environment Option Interoperability Issues D. Borman

RFC 1572
Telnet Environment Option S. Alexander

RFC 1573
Evolution of the Interfaces Group of MIB-II K. McCloghrie, F. Kastenholz

RFC 1577
Classical IP and ARP over ATM M. Laubach

RFC 1583
OSPF Version 2 J. Moy

RFC 1591
Domain Name System Structure and Delegation J. Postel

RFC 1592
Simple Network Management Protocol Distributed Protocol Interface Version 2.0 B. Wijnen, G. Carpenter, K. Curran, A. Sehgal, G. Waters

RFC 1594
FYI on Questions and Answers— Answers to Commonly Asked "New Internet User" Questions A. Marine, J. Reynolds, G. Malkin

RFC 1644
T/TCP — TCP Extensions for Transactions Functional Specification R. Braden

RFC 1646
TN3270 Extensions for LUname and Printer Selection C. Graves, T. Butts, M. Angel

RFC 1647
TN3270 Enhancements B. Kelly

RFC 1652
SMTP Service Extension for 8bit-MIMEtransport J. Klensin, N. Freed, M. Rose, E. Stefferud, D. Crocker

RFC 1664
Using the Internet DNS to Distribute RFC1327 Mail Address Mapping Tables C. Allochio, A. Bonito, B. Cole, S. Giordano, R. Hagens

RFC 1693
An Extension to TCP: Partial Order Service T. Connolly, P. Amer, P. Conrad

RFC 1695
Definitions of Managed Objects for ATM Management Version 8.0 using SMIv2 M. Ahmed, K. Tesink

RFC 1701
Generic Routing Encapsulation (GRE) S. Hanks, T. Li, D. Farinacci, P. Traina

RFC 1702
Generic Routing Encapsulation over IPv4 networks S. Hanks, T. Li, D. Farinacci, P. Traina

RFC 1706
DNS NSAP Resource Records B. Manning, R. Colella

RFC 1712
DNS Encoding of Geographical Location C. Farrell, M. Schulze, S. Pleitner D. Baldoni

RFC 1713
Tools for DNS debugging A. Romao

RFC 1723
RIP Version 2—Carrying Additional Information G. Malkin

RFC 1752
The Recommendation for the IP Next Generation Protocol S. Bradner, A. Mankin

RFC 1766
Tags for the Identification of Languages H. Alvestrand

RFC 1771
A Border Gateway Protocol 4 (BGP-4) Y. Rekhter, T. Li

RFC 1794
DNS Support for Load Balancing T. Brisco

RFC 1819
Internet Stream Protocol Version 2 (ST2) Protocol Specification—Version ST2+ L. Delgrossi, L. Berger Eds.

RFC 1826
IP Authentication Header R. Atkinson

RFC 1828
IP Authentication using Keyed MD5 P. Metzger, W. Simpson

RFC 1829
The ESP DES-CBC Transform P. Karn, P. Metzger, W. Simpson

RFC 1830
SMTP Service Extensions for Transmission of Large and Binary MIME Messages G. Vaudreuil

RFC 1831
RPC: Remote Procedure Call Protocol Specification Version 2 R. Srinivasan

RFC 1832
XDR: External Data Representation Standard R. Srinivasan

RFC 1833
Binding Protocols for ONC RPC Version 2 R. Srinivasan

RFC 1850
OSPF Version 2 Management Information Base F. Baker, R. Coltun

RFC 1854
SMTP Service Extension for Command Pipelining N. Freed

RFC 1869
SMTP Service Extensions J. Klensin, N. Freed, M. Rose, E. Stefferud, D. Crocker

RFC 1870
SMTP Service Extension for Message Size Declaration J. Klensin, N. Freed, K. Moore

RFC 1876
A Means for Expressing Location Information in the Domain Name System C. Davis, P. Vixie, T. Goodwin, I. Dickinson

RFC 1883
Internet Protocol, Version 6 (IPv6) Specification S. Deering, R. Hinden

RFC 1884
IP Version 6 Addressing Architecture R. Hinden, S. Deering, Eds.

RFC 1886
DNS Extensions to support IP version 6 S. Thomson, C. Huitema

RFC 1888
OSI NSAPs and IPv6 J. Bound, B. Carpenter, D. Harrington, J. Houldsworth, A. Lloyd

RFC 1891
SMTP Service Extension for Delivery Status Notifications K. Moore

RFC 1892
The Multipart/Report Content Type for the Reporting of Mail System Administrative Messages G. Vaudreuil

RFC 1894
*An Extensible Message Format for Delivery Status Notifications*K. Moore, G. Vaudreuil

RFC 1901
Introduction to Community-based SNMPv2 J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1902
Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2) J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1903
Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2) J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1904
Conformance Statements for Version 2 of the Simple Network Management Protocol (SNMPv2) J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1905
Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2) J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1906
Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2) J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1907
Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2) J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1908
Coexistence between Version 1 and Version 2 of the Internet-standard Network Management Framework J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1912
Common DNS Operational and Configuration Errors D. Barr

RFC 1918
: *Address Allocation for Private Internets* Y. Rekhter, B. Moskowitz, D. Karrenberg, G.J. de Groot, E. Lear

RFC 1928
: *SOCKS Protocol Version 5* M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, L. Jones

RFC 1930
: *Guidelines for creation, selection, and registration of an Autonomous System (AS)* J. Hawkinson, T. Bates

RFC 1939
: *Post Office Protocol-Version 3* J. Myers, M. Rose

RFC 1981
: *Path MTU Discovery for IP version 6* J. McCann, S. Deering, J. Mogul

RFC 1982
: *Serial Number Arithmetic* R. Elz, R. Bush

RFC 1985
: *SMTP Service Extension for Remote Message Queue Starting* J. De Winter

RFC 1995
: *Incremental Zone Transfer in DNS* M. Ohta

RFC 1996
: *A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)* P. Vixie

RFC 2010
: *Operational Criteria for Root Name Servers* B. Manning, P. Vixie

RFC 2011
: *SNMPv2 Management Information Base for the Internet Protocol using SMIv2* K. McCloghrie, Ed.

RFC 2012
: *SNMPv2 Management Information Base for the Transmission Control Protocol using SMIv2* K. McCloghrie, Ed.

RFC 2013
: *SNMPv2 Management Information Base for the User Datagram Protocol using SMIv2* K. McCloghrie, Ed.

RFC 2018
: *TCP Selective Acknowledgement Options* M. Mathis, J. Mahdavi, S. Floyd, A. Romanow

RFC 2026
: *The Internet Standards Process — Revision 3* S. Bradner

RFC 2030
: *Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI* D. Mills

RFC 2033
: *Local Mail Transfer Protocol* J. Myers

RFC 2034
: *SMTP Service Extension for Returning Enhanced Error Codes*N. Freed

RFC 2040
: *The RC5, RC5–CBC, RC-5–CBC-Pad, and RC5–CTS Algorithms*R. Baldwin, R. Rivest

RFC 2045
: *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies* N. Freed, N. Borenstein

RFC 2052
: *A DNS RR for specifying the location of services (DNS SRV)* A. Gulbrandsen, P. Vixie

RFC 2065
: *Domain Name System Security Extensions* D. Eastlake 3rd, C. Kaufman

RFC 2066
: *TELNET CHARSET Option* R. Gellens

RFC 2080
RIPng for IPv6 G. Malkin, R. Minnear

RFC 2096
IP Forwarding Table MIB F. Baker

RFC 2104
HMAC: Keyed-Hashing for Message Authentication H. Krawczyk, M. Bellare, R. Canetti

RFC 2119
Keywords for use in RFCs to Indicate Requirement Levels S. Bradner

RFC 2133
Basic Socket Interface Extensions for IPv6 R. Gilligan, S. Thomson, J. Bound, W. Stevens

RFC 2136
Dynamic Updates in the Domain Name System (DNS UPDATE) P. Vixie, Ed., S. Thomson, Y. Rekhter, J. Bound

RFC 2137
Secure Domain Name System Dynamic Update D. Eastlake 3rd

RFC 2163
Using the Internet DNS to Distribute MIXER Conformant Global Address Mapping (MCGAM) C. Allocchio

RFC 2168
Resolution of Uniform Resource Identifiers using the Domain Name System R. Daniel, M. Mealling

RFC 2178
OSPF Version 2 J. Moy

RFC 2181
Clarifications to the DNS Specification R. Elz, R. Bush

RFC 2205
Resource ReSerVation Protocol (RSVP)—Version 1 Functional Specification R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, S. Jamin

RFC 2210
The Use of RSVP with IETF Integrated Services J. Wroclawski

RFC 2211
Specification of the Controlled-Load Network Element Service J. Wroclawski

RFC 2212
Specification of Guaranteed Quality of Service S. Shenker, C. Partridge, R. Guerin

RFC 2215
General Characterization Parameters for Integrated Service Network Elements S. Shenker, J. Wroclawski

RFC 2217
Telnet Com Port Control Option G. Clarke

RFC 2219
Use of DNS Aliases for Network Services M. Hamilton, R. Wright

RFC 2228
FTP Security Extensions M. Horowitz, S. Lunt

RFC 2230
Key Exchange Delegation Record for the DNS R. Atkinson

RFC 2233
The Interfaces Group MIB using SMIv2 K. McCloghrie, F. Kastenholz

RFC 2240
A Legal Basis for Domain Name Allocation O. Vaughn

RFC 2246
The TLS Protocol Version 1.0 T. Dierks, C. Allen

RFC 2251
Lightweight Directory Access Protocol (v3) M. Wahl, T. Howes, S. Kille

RFC 2253
Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names M. Wahl, S. Kille, T. Howes

RFC 2254
The String Representation of LDAP Search Filters T. Howes

RFC 2261
An Architecture for Describing SNMP Management Frameworks D. Harrington, R. Presuhn, B. Wijnen

RFC 2262
Message Processing and Dispatching for the Simple Network Management Protocol (SNMP) J. Case, D. Harrington, R. Presuhn, B. Wijnen

RFC 2271
An Architecture for Describing SNMP Management Frameworks D. Harrington, R. Presuhn, B. Wijnen

RFC 2273
SNMPv3 Applications D. Levi, P. Meyer, B. Stewartz

RFC 2274
User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3) U. Blumenthal, B. Wijnen

RFC 2275
View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP) B. Wijnen, R. Presuhn, K. McCloghrie

RFC 2279
UTF-8, a transformation format of ISO 10646 F. Yergeau

RFC 2292
Advanced Sockets API for IPv6 W. Stevens, M. Thomas

RFC 2308
Negative Caching of DNS Queries (DNS NCACHE) M. Andrews

RFC 2317
Classless IN-ADDR.ARPA delegation H. Eidnes, G. de Groot, P. Vixie

RFC 2320
Definitions of Managed Objects for Classical IP and ARP Over ATM Using SMIv2 (IPOA-MIB) M. Greene, J. Luciani, K. White, T. Kuo

RFC 2328
OSPF Version 2 J. Moy

RFC 2345
Domain Names and Company Name Retrieval J. Klensin, T. Wolf, G. Oglesby

RFC 2352
A Convention for Using Legal Names as Domain Names O. Vaughn

RFC 2355
TN3270 Enhancements B. Kelly

RFC 2358
Definitions of Managed Objects for the Ethernet-like Interface Types J. Flick, J. Johnson

RFC 2373
IP Version 6 Addressing Architecture R. Hinden, S. Deering

RFC 2374
An IPv6 Aggregatable Global Unicast Address Format R. Hinden, M. O'Dell, S. Deering

RFC 2375
IPv6 Multicast Address Assignments R. Hinden, S. Deering

RFC 2385
Protection of BGP Sessions via the TCP MD5 Signature Option A. Hefferman

RFC 2389
Feature negotiation mechanism for the File Transfer Protocol P. Hethmon, R. Elz

RFC 2401
Security Architecture for Internet Protocol S. Kent, R. Atkinson

RFC 2402
IP Authentication Header S. Kent, R. Atkinson

RFC 2403
The Use of HMAC-MD5–96 within ESP and AH C. Madson, R. Glenn

RFC 2404
The Use of HMAC-SHA–1–96 within ESP and AH C. Madson, R. Glenn

RFC 2405
The ESP DES-CBC Cipher Algorithm With Explicit IV C. Madson, N. Doraswamy

RFC 2406
IP Encapsulating Security Payload (ESP) S. Kent, R. Atkinson

RFC 2407
*The Internet IP Security Domain of Interpretation for ISAKMP*D. Piper

RFC 2408
Internet Security Association and Key Management Protocol (ISAKMP) D. Maughan, M. Schertler, M. Schneider, J. Turner

RFC 2409
The Internet Key Exchange (IKE) D. Harkins, D. Carrel

RFC 2410
The NULL Encryption Algorithm and Its Use With IPsec R. Glenn, S. Kent,

RFC 2428
FTP Extensions for IPv6 and NATs M. Allman, S. Ostermann, C. Metz

RFC 2445
Internet Calendaring and Scheduling Core Object Specification (iCalendar) F. Dawson, D. Stenerson

RFC 2459
Internet X.509 Public Key Infrastructure Certificate and CRL Profile R. Housley, W. Ford, W. Polk, D. Solo

RFC 2460
Internet Protocol, Version 6 (IPv6) Specification S. Deering, R. Hinden

RFC 2461
Neighbor Discovery for IP Version 6 (IPv6) T. Narten, E. Nordmark, W. Simpson

RFC 2462
IPv6 Stateless Address Autoconfiguration S. Thomson, T. Narten

RFC 2463
Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification A. Conta, S. Deering

RFC 2464
Transmission of IPv6 Packets over Ethernet Networks M. Crawford

RFC 2466
Management Information Base for IP Version 6: ICMPv6 Group D. Haskin, S. Onishi

RFC 2476
Message Submission R. Gellens, J. Klensin

RFC 2487
SMTP Service Extension for Secure SMTP over TLS P. Hoffman

RFC 2505
Anti-Spam Recommendations for SMTP MTAs G. Lindberg

RFC 2523
Photuris: Extended Schemes and Attributes P. Karn, W. Simpson

RFC 2535
Domain Name System Security Extensions D. Eastlake 3rd

RFC 2538
Storing Certificates in the Domain Name System (DNS) D. Eastlake 3rd, O. Gudmundsson

RFC 2539
Storage of Diffie-Hellman Keys in the Domain Name System (DNS) D. Eastlake 3rd

RFC 2540
Detached Domain Name System (DNS) Information D. Eastlake 3rd

RFC 2554
SMTP Service Extension for Authentication J. Myers

RFC 2570
Introduction to Version 3 of the Internet-standard Network Management Framework J. Case, R. Mundy, D. Partain, B. Stewart

RFC 2571
An Architecture for Describing SNMP Management Frameworks B. Wijnen, D. Harrington, R. Presuhn

RFC 2572
Message Processing and Dispatching for the Simple Network Management Protocol (SNMP) J. Case, D. Harrington, R. Presuhn, B. Wijnen

RFC 2573
SNMP Applications D. Levi, P. Meyer, B. Stewart

RFC 2574
User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3) U. Blumenthal, B. Wijnen

RFC 2575
View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP) B. Wijnen, R. Presuhn, K. McCloghrie

RFC 2576
Co-Existence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework R. Frye, D. Levi, S. Routhier, B. Wijnen

RFC 2578
Structure of Management Information Version 2 (SMIv2) K. McCloghrie, D. Perkins, J. Schoenwaelder

RFC 2579
Textual Conventions for SMIv2 K. McCloghrie, D. Perkins, J. Schoenwaelder

RFC 2580
Conformance Statements for SMIv2 K. McCloghrie, D. Perkins, J. Schoenwaelder

RFC 2581
TCP Congestion Control M. Allman, V. Paxson, W. Stevens

RFC 2583
Guidelines for Next Hop Client (NHC) Developers R. Carlson, L. Winkler

RFC 2591
Definitions of Managed Objects for Scheduling Management Operations D. Levi, J. Schoenwaelder

RFC 2625
IP and ARP over Fibre Channel M. Rajagopal, R. Bhagwat, W. Rickard

RFC 2635
Don't SPEW A Set of Guidelines for Mass Unsolicited Mailings and Postings (spam)* S. Hambridge, A. Lunde

RFC 2637
Point-to-Point Tunneling Protocol K. Hamzeh, G. Pall, W. Verthein, J. Taarud, W. Little, G. Zorn

RFC 2640
Internationalization of the File Transfer Protocol B. Curtin

RFC 2665
Definitions of Managed Objects for the Ethernet-like Interface Types J. Flick, J. Johnson

RFC 2671
Extension Mechanisms for DNS (EDNS0) P. Vixie

RFC 2672
Non-Terminal DNS Name Redirection M. Crawford

RFC 2675
IPv6 Jumbograms D. Borman, S. Deering, R. Hinden

RFC 2710
Multicast Listener Discovery (MLD) for IPv6 S. Deering, W. Fenner, B. Haberman

RFC 2711
IPv6 Router Alert Option C. Partridge, A. Jackson

RFC 2740
OSPF for IPv6 R. Coltun, D. Ferguson, J. Moy

RFC 2753
A Framework for Policy-based Admission Control R. Yavatkar, D. Pendarakis, R. Guerin

RFC 2782
A DNS RR for specifying the location of services (DNS SRV) A. Gubrandsen, P. Vixix, L. Esibov

RFC 2821
Simple Mail Transfer Protocol J. Klensin, Ed.

RFC 2822
Internet Message Format P. Resnick, Ed.

RFC 2840
TELNET KERMIT OPTION J. Altman, F. da Cruz

RFC 2845
Secret Key Transaction Authentication for DNS (TSIG) P. Vixie, O. Gudmundsson, D. Eastlake 3rd, B. Wellington

RFC 2851
Textual Conventions for Internet Network Addresses M. Daniele, B. Haberman, S. Routhier, J. Schoenwaelder

RFC 2852
Deliver By SMTP Service Extension D. Newman

RFC 2874
DNS Extensions to Support IPv6 Address Aggregation and Renumbering M. Crawford, C. Huitema

RFC 2915
The Naming Authority Pointer (NAPTR) DNS Resource Record M. Mealling, R. Daniel

RFC 2920
SMTP Service Extension for Command Pipelining N. Freed

RFC 2930
Secret Key Establishment for DNS (TKEY RR) D. Eastlake, 3rd

RFC 2941
Telnet Authentication Option T. Ts'o, ed., J. Altman

RFC 2942
Telnet Authentication: Kerberos Version 5 T. Ts'o

RFC 2946
Telnet Data Encryption Option T. Ts'o

RFC 2952
Telnet Encryption: DES 64 bit Cipher Feedback T. Ts'o

RFC 2953
Telnet Encryption: DES 64 bit Output Feedback T. Ts'o

RFC 2992
Analysis of an Equal-Cost Multi-Path Algorithm C. Hopps

RFC 3019
IP Version 6 Management Information Base for The Multicast Listener Discovery Protocol B. Haberman, R. Worzella

RFC 3060
Policy Core Information Model—Version 1 Specification B. Moore, E. Ellesson, J. Strassner, A. Westerinen

RFC 3152
Delegation of IP6.ARPA R. Bush

RFC 3164
The BSD Syslog Protocol C. Lonvick

RFC 3207
SMTP Service Extension for Secure SMTP over Transport Layer Security P. Hoffman

RFC 3226
DNSSEC and IPv6 A6 aware server/resolver message size requirements O. Gudmundsson

RFC 3291
Textual Conventions for Internet Network Addresses M. Daniele, B. Haberman, S. Routhier, J. Schoenwaelder

RFC 3363
Representing Internet Protocol version 6 (IPv6) Addresses in the Domain Name System R. Bush, A. Durand, B. Fink, O. Gudmundsson, T. Hain

RFC 3376
Internet Group Management Protocol, Version 3 B. Cain, S. Deering, I. Kouvelas, B. Fenner, A. Thyagarajan

RFC 3390
Increasing TCP's Initial Window M. Allman, S. Floyd, C. Partridge

RFC 3410
Introduction and Applicability Statements for Internet-Standard Management Framework J. Case, R. Mundy, D. Partain, B. Stewart

RFC 3411
An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks D. Harrington, R. Presuhn, B. Wijnen

RFC 3412
Message Processing and Dispatching for the Simple Network Management Protocol (SNMP) J. Case, D. Harrington, R. Presuhn, B. Wijnen

RFC 3413
Simple Network Management Protocol (SNMP) Applications D. Levi, P. Meyer, B. Stewart

RFC 3414
User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3) U. Blumenthal, B. Wijnen

RFC 3415
View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP) B. Wijnen, R. Presuhn, K. McCloghrie

RFC 3416
Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP) R. Presuhn, J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 3417
: *Transport Mappings for the Simple Network Management Protocol (SNMP)* R. Presuhn, J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 3418
: *Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)* R. Presuhn, J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 3419
: *Textual Conventions for Transport Addresses* M. Daniele, J. Schoenwaelder

RFC 3484
: *Default Address Selection for Internet Protocol version 6 (IPv6)* R. Draves

RFC 3493
: *Basic Socket Interface Extensions for IPv6* R. Gilligan, S. Thomson, J. Bound, J. McCann, W. Stevens

RFC 3513
: *Internet Protocol Version 6 (IPv6) Addressing Architecture* R. Hinden, S. Deering

RFC 3526
: *More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)* T. Kivinen, M. Kojo

RFC 3542
: *Advanced Sockets Application Programming Interface (API) for IPv6* W. Richard Stevens, M. Thomas, E. Nordmark, T. Jinmei

RFC 3566
: *The AES-XCBC-MAC-96 Algorithm and Its Use With IPsec* S. Frankel, H. Herbert

RFC 3569
: *An Overview of Source-Specific Multicast (SSM)* S. Bhattacharyya, Ed.

RFC 3584
: *Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework* R. Frye, D. Levi, S. Routhier, B. Wijnen

RFC 3602
: *The AES-CBC Cipher Algorithm and Its Use with IPsec* S. Frankel, R. Glenn, S. Kelly

RFC 3629
: *UTF-8, a transformation format of ISO 10646* R. Kermode, C. Vicisano

RFC 3658
: *Delegation Signer (DS) Resource Record (RR)* O. Gudmundsson

RFC 3678
: *Socket Interface Extensions for Multicast Source Filters* D. Thaler, B. Fenner, B. Quinn

RFC 3715
: *IPsec-Network Address Translation (NAT) Compatibility Requirements* B. Aboba, W. Dixon

RFC 3810
: *Multicast Listener Discovery Version 2 (MLDv2) for IPv6* R. Vida, Ed., L. Costa, Ed.

RFC 3826
: *The Advanced Encryption Standard (AES) Cipher Algorithm in the SNMP User-based Security Model* U. Blumenthal, F. Maino, K McCloghrie.

RFC 3947
: *Negotiation of NAT-Traversal in the IKE* T. Kivinen, B. Swander, A. Huttunen, V. Volpe

RFC 3948
: *UDP Encapsulation of IPsec ESP Packets* A. Huttunen, B. Swander, V. Volpe, L. DiBurro, M. Stenberg

RFC 4001
: *Textual Conventions for Internet Network Addresses* M. Daniele, B. Haberman, S. Routhier, J. Schoenwaelder

RFC 4007
: *IPv6 Scoped Address Architecture* S. Deering, B. Haberman, T. Jinmei, E. Nordmark, B. Zill

RFC 4022
Management Information Base for the Transmission Control Protocol (TCP) R. Raghunarayan

RFC 4106
The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP) J. Viega, D. McGrew

RFC 4109
Algorithms for Internet Key Exchange version 1 (IKEv1) P. Hoffman

RFC 4113
Management Information Base for the User Datagram Protocol (UDP) B. Fenner, J. Flick

RFC 4191
Default Router Preferences and More-Specific Routes R. Draves, D. Thaler

RFC 4217
Securing FTP with TLS P. Ford-Hutchinson

RFC 4292
IP Forwarding Table MIB B. Haberman

RFC 4293
Management Information Base for the Internet Protocol (IP) S. Routhier

RFC 4301
Security Architecture for the Internet Protocol S. Kent, K. Seo

RFC 4302
IP Authentication Header S. Kent

RFC 4303
IP Encapsulating Security Payload (ESP) S. Kent

RFC 4304
Extended Sequence Number (ESN) Addendum to IPsec Domain of Interpretation (DOI) for Internet Security Association and Key Management Protocol (ISAKMP) S. Kent

RFC 4307
Cryptographic Algorithms for Use in the Internet Key Exchange Version 2 (IKEv2) J. Schiller

RFC 4308
Cryptographic Suites for IPsec P. Hoffman

RFC 4434
The AES-XCBC-PRF-128 Algorithm for the Internet Key Exchange Protocol P. Hoffman

RFC 4443
Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification A. Conta, S. Deering

RFC 4552
Authentication/Confidentiality for OSPFv3 M. Gupta, N. Melam

RFC 4678
Server/Application State Protocol v1 A. Bivens

RFC 4753
ECP Groups for IKE and IKEv2 D. Fu, J. Solinas

RFC 4754
IKE and IKEv2 Authentication Using the Elliptic Curve Digital Signature Algorithm (ECDSA) D. Fu, J. Solinas

RFC 4809
Requirements for an IPsec Certificate Management Profile C. Bonatti, Ed., S. Turner, Ed., G. Lebovitz, Ed.

RFC 4835
Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH) V. Manral

RFC 4862
IPv6 Stateless Address Autoconfiguration S. Thomson, T. Narten, T. Jinmei

RFC 4868
Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec S. Kelly, S. Frankel

RFC 4869
Suite B Cryptographic Suites for IPsec L. Law, J. Solinas

RFC 4941
Privacy Extensions for Stateless Address Autoconfiguration in IPv6 T. Narten, R. Draves, S. Krishnan

RFC 4945
The Internet IP Security PKI Profile of IKEv1/ISAKMP, IKEv2, and PKIX B. Korver

RFC 5014
IPv6 Socket API for Source Address Selection E. Nordmark, S. Chakrabarti, J. Laganier

RFC 5095
Deprecation of Type 0 Routing Headers in IPv6 J. Abley, P. Savola, G. Neville-Neil

RFC 5175
IPv6 Router Advertisement Flags Option B. Haberman, Ed., R. Hinden

RFC 5282
Using Authenticated Encryption Algorithms with the Encrypted Payload of the Internet Key Exchange version 2 (IKEv2) Protocol D. Black, D. McGrew

RFC 5996
Internet Key Exchange Protocol Version 2 (IKEv2) C. Kaufman, P. Hoffman, Y. Nir, P. Eronen

RFC 8446
The Transport Layer Security (TLS) Protocol Version 1.3 E. Rescorla

Internet drafts

Internet drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Other groups can also distribute working documents as Internet drafts. You can see Internet drafts at http://www.ietf.org/ID.html.

Appendix C. Accessibility

Publications for this product are offered in Adobe Portable Document Format (PDF) and should be compliant with accessibility standards. If you experience difficulties when using PDF files, you can view the information through the z/OS Internet Library website http://www.ibm.com/systems/z/os/zos/library/bkserv/ or IBM Knowledge Center http://www.ibm.com/support/knowledgecenter/. If you continue to experience problems, send a message to Contact z/OS web page (www.ibm.com/systems/z/os/zos/webqs.html) or write to:

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. See z/OS TSO/E Primer, z/OS TSO/E User's Guide, and z/OS ISPF User's Guide Vol I for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Notices

Terms and conditions for product documentation

Applicability

Personal use

Commercial use

Rights

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at ibm.com®/privacy and IBM's Online Privacy Statement at ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, HCD, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Trademarks

Bibliography

This bibliography contains descriptions of the documents in the z/OS Communications Server library.

z/OS Communications Server documentation is available online at the z/OS Internet Library web page at http://www.ibm.com/systems/z/os/zos/library/bkserv/.

z/OS Communications Server library updates

Updates to documents are also available on RETAIN and in information APARs (info APARs). Go to http://www.software.ibm.com/support to view information APARs.

- z/OS Communications Server V2R1 New Function APAR Summary
- z/OS Communications Server V2R2 New Function APAR Summary
- z/OS Communications Server V2R3 New Function APAR Summary

z/OS Communications Server information

z/OS Communications Server product information is grouped by task in the following tables.

Planning

Title	Number	Description
z/OS Communications Server: New Function Summary	GC27-3664	This document is intended to help you plan for new IP or SNA functions, whether you are migrating from a previous version or installing z/OS for the first time. It summarizes what is new in the release and identifies the suggested and required modifications needed to use the enhanced functions.
z/OS Communications Server: IPv6 Network and Application Design Guide	SC27-3663	This document is a high-level introduction to IPv6. It describes concepts of z/OS Communications Server's support of IPv6, coexistence with IPv4, and migration issues.

Resource definition, configuration, and tuning

Title	Number	Description
z/OS Communications Server: IP Configuration Guide	SC27-3650	This document describes the major concepts involved in understanding and configuring an IP network. Familiarity with the z/OS operating system, IP protocols, z/OS UNIX System Services, and IBM Time Sharing Option (TSO) is recommended. Use this document with the z/OS Communications Server: IP Configuration Reference.

Title	Number	Description
z/OS Communications Server: IP Configuration Reference	SC27-3651	This document presents information for people who want to administer and maintain IP. Use this document with the z/OS Communications Server: IP Configuration Guide. The information in this document includes: • TCP/IP configuration data sets • Configuration statements • Translation tables • Protocol number and port assignments
z/OS Communications Server: SNA Network Implementation Guide	SC27-3672	This document presents the major concepts involved in implementing an SNA network. Use this document with the z/OS Communications Server: SNA Resource Definition Reference.
z/OS Communications Server: SNA Resource Definition Reference	SC27-3675	This document describes each SNA definition statement, start option, and macroinstruction for user tables. It also describes NCP definition statements that affect SNA. Use this document with the z/OS Communications Server: SNA Network Implementation Guide.
z/OS Communications Server: SNA Resource Definition Samples	SC27-3676	This document contains sample definitions to help you implement SNA functions in your networks, and includes sample major node definitions.
z/OS Communications Server: IP Network Print Facility	SC27-3658	This document is for systems programmers and network administrators who need to prepare their network to route SNA, JES2, or JES3 printer output to remote printers using TCP/IP Services.

Operation

Title	Number	Description
z/OS Communications Server: IP User's Guide and Commands	SC27-3662	This document describes how to use TCP/IP applications. It contains requests with which a user can log on to a remote host using Telnet, transfer data sets using FTP, send electronic mail, print on remote printers, and authenticate network users.
z/OS Communications Server: IP System Administrator's Commands	SC27-3661	This document describes the functions and commands helpful in configuring or monitoring your system. It contains system administrator's commands, such as TSO NETSTAT, PING, TRACERTE and their UNIX counterparts. It also includes TSO and MVS commands commonly used during the IP configuration process.
z/OS Communications Server: SNA Operation	SC27-3673	This document serves as a reference for programmers and operators requiring detailed information about specific operator commands.
z/OS Communications Server: Quick Reference	SC27-3665	This document contains essential information about SNA and IP commands.

Customization

Title	Number	Description
z/OS Communications Server: SNA Customization	SC27-3666	This document enables you to customize SNA, and includes the following information: • Communication network management (CNM) routing table • Logon-interpret routine requirements • Logon manager installation-wide exit routine for the CLU search exit • TSO/SNA installation-wide exit routines • SNA installation-wide exit routines

Writing application programs

Title	Number	Description
z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference	SC27-3660	This document describes the syntax and semantics of program source code necessary to write your own application programming interface (API) into TCP/IP. You can use this interface as the communication base for writing your own client or server application. You can also use this document to adapt your existing applications to communicate with each other using sockets over TCP/IP.
z/OS Communications Server: IP CICS Sockets Guide	SC27-3649	This document is for programmers who want to set up, write application programs for, and diagnose problems with the socket interface for CICS using z/OS TCP/IP.
z/OS Communications Server: IP IMS Sockets Guide	SC27-3653	This document is for programmers who want application programs that use the IMS TCP/IP application development services provided by the TCP/IP Services of IBM.
z/OS Communications Server: IP Programmer's Guide and Reference	SC27-3659	This document describes the syntax and semantics of a set of high-level application functions that you can use to program your own applications in a TCP/IP environment. These functions provide support for application facilities, such as user authentication, distributed databases, distributed processing, network management, and device sharing. Familiarity with the z/OS operating system, TCP/IP protocols, and IBM Time Sharing Option (TSO) is recommended.
z/OS Communications Server: SNA Programming	SC27-3674	This document describes how to use SNA macroinstructions to send data to and receive data from (1) a terminal in either the same or a different domain, or (2) another application program in either the same or a different domain.
z/OS Communications Server: SNA Programmer's LU 6.2 Guide	SC27-3669	This document describes how to use the SNA LU 6.2 application programming interface for host application programs. This document applies to programs that use only LU 6.2 sessions or that use LU 6.2 sessions along with other session types. (Only LU 6.2 sessions are covered in this document.)

Title	Number	Description
z/OS Communications Server: SNA Programmer's LU 6.2 Reference	SC27-3670	This document provides reference material for the SNA LU 6.2 programming interface for host application programs.
z/OS Communications Server: CSM Guide	SC27-3647	This document describes how applications use the communications storage manager.
z/OS Communications Server: CMIP Services and Topology Agent Guide	SC27-3646	This document describes the Common Management Information Protocol (CMIP) programming interface for application programmers to use in coding CMIP application programs. The document provides guide and reference information about CMIP services and the SNA topology agent.

Diagnosis

Title	Number	Description
z/OS Communications Server: IP Diagnosis Guide	GC27-3652	This document explains how to diagnose TCP/IP problems and how to determine whether a specific problem is in the TCP/IP product code. It explains how to gather information for and describe problems to the IBM Software Support Center.
z/OS Communications Server: ACF/TAP Trace Analysis Handbook	GC27-3645	This document explains how to gather the trace data that is collected and stored in the host processor. It also explains how to use the Advanced Communications Function/Trace Analysis Program (ACF/TAP) service aid to produce reports for analyzing the trace data information.
z/OS Communications Server: SNA Diagnosis Vol 1, Techniques and Procedures and z/OS Communications Server: SNA Diagnosis Vol 2, FFST Dumps and the VIT	GC27-3667 GC27-3668	These documents help you identify an SNA problem, classify it, and collect information about it before you call the IBM Support Center. The information collected includes traces, dumps, and other problem documentation.
z/OS Communications Server: SNA Data Areas Volume 1 and z/OS Communications Server: SNA Data Areas Volume 2	GC31-6852 GC31-6853	These documents describe SNA data areas and can be used to read an SNA dump. They are intended for IBM programming service representatives and customer personnel who are diagnosing problems with SNA.

Messages and codes

Title	Number	Description
z/OS Communications Server: SNA Messages	SC27-3671	This document describes the ELM, IKT, IST, IUT, IVT, and USS messages. Other information in this document includes: • Command and RU types in SNA messages • Node and ID types in SNA messages • Supplemental message-related information

Title	Number	Description
z/OS Communications Server: IP Messages Volume 1 (EZA)	SC27-3654	This volume contains TCP/IP messages beginning with EZA.
z/OS Communications Server: IP Messages Volume 2 (EZB, EZD)	SC27-3655	This volume contains TCP/IP messages beginning with EZB or EZD.
z/OS Communications Server: IP Messages Volume 3 (EZY)	SC27-3656	This volume contains TCP/IP messages beginning with EZY.
z/OS Communications Server: IP Messages Volume 4 (EZZ, SNM)	SC27-3657	This volume contains TCP/IP messages beginning with EZZ and SNM.
z/OS Communications Server: IP and SNA Codes	SC27-3648	This document describes codes and other information that appear in z/OS Communications Server messages.

Index

Communicating your comments to IBM

If you especially like or dislike anything about this document, you can send us comments electronically by using one of the following methods:

Internet email:
comsvrcf@us.ibm.com

World Wide Web:
http://www.ibm.com/systems/z/os/zos/webqs.html

If you would like a reply, be sure to include your name, address, and telephone number. Make sure to include the following information in your comment or note:

- Title and order number of this document
- Page number or topic related to your comment

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this document. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

SC27-3663-40