z/OS Communications Server

IPv6 Network and Application Design Guide

Version 1 Release 4

SC31-8885-00

z/OS Communications Server

IPv6 Network and Application Design Guide

Version 1 Release 4

SC31-8885-00

Note

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 137.

First Edition (September 2002)

This edition applies to Version 1 Release 4 of z/OS (5694-A01) and Version 1 Release 4 of z/OS.e (5655-G52) and to all subsequent releases and modifications until otherwise indicated in new editions.

Publications are not stocked at the address given below. If you want more IBM® publications, ask your IBM representative or write to the IBM branch office serving your locality.

A form for your comments is provided at the back of this document. If the form has been removed, you may address comments to:

IBM Corporation
Software Reengineering
Department G7IA/ Bldg 503
Research Triangle Park, NC 27709-9990
U.S.A.

If you prefer to send comments electronically, use one of the following methods:

Fax (USA and Canada):
1-800-254-0206

Internet e-mail:
usib2hpd@vnet.ibm.com

World Wide Web:
http://www.ibm.com/servers/eserver/zseries/zos/webqs.html

IBMLink™:
CIBMORCF at RALVM17

IBM Mail Exchange:
tkinlaw@us.ibm.com

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

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 1 Release 4.

This document supports z/OS.e™.

Who should use this document

The reader of this document should be familiar with the IPv6 protocol.

Parts 1, 2, and 4 of this document are intended for programmers and system administrators who are familiar with TCP/IP, MVS™, and z/OS UNIX®.

Part 3 is intended for application programmers.

Where to find more information

This section contains:
- Pointers to information available on the Internet
- Information about licensed documentation
- Information about LookAt, the online message tool
- A set of tables that describes the documents in the z/OS Communications Server (z/OS CS) library, along with related publications

Where to find related information on the Internet

z/OS
– http://www.ibm.com/servers/eserver/zseries/zos/

z/OS Internet Library
– http://www.ibm.com/servers/eserver/zseries/zos/bkserv/

IBM Communications Server product
– http://www.software.ibm.com/network/commserver/

IBM Communications Server product support
– http://www.software.ibm.com/network/commserver/support/

IBM Systems Center publications
– http://www.redbooks.ibm.com/

IBM Systems Center flashes
– http://www-1.ibm.com/support/techdocs/atsmastr.nsf

RFCs
– http://www.ietf.org/rfc.html

RFC drafts
– http://www.ietf.org/ID.html

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

DNS web sites

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

USENET news groups:
comp.protocols.dns.bind

For BIND mailing lists, see:

- http://www.isc.org/ml-archives/
  - 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 (Note: This list may 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.

For definitions of the terms and abbreviations used in this document, you can view or download the latest *IBM Glossary of Computing Terms* at the following Web address:

http://www.ibm.com/ibm/terminology

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

Accessing z/OS licensed documents on the Internet

z/OS licensed documentation is available on the Internet in PDF format at the IBM Resource Link™ Web site at:

`http://www.ibm.com/servers/resourcelink`

Licensed documents are available only to customers with a z/OS license. Access to these documents requires an IBM Resource Link user ID and password, and a key code. With your z/OS order you received a Memo to Licensees, (GI10-0671), that includes this key code.

To obtain your IBM Resource Link user ID and password, log on to:

`http://www.ibm.com/servers/resourcelink`

To register for access to the z/OS licensed documents:

1. Sign in to Resource Link using your Resource Link user ID and password.
2. Select **User Profiles** located on the left-hand navigation bar.

Note: You cannot access the z/OS licensed documents unless you have registered for access to them and received an e-mail confirmation informing you that your request has been processed.

Printed licensed documents are not available from IBM.

You can use the PDF format on either **z/OS Licensed Product Library CD-ROM** or IBM Resource Link to print licensed documents.

Using LookAt to look up message explanations

LookAt is an online facility that allows you to look up explanations for most messages you encounter, as well as for some system abends and codes. Using LookAt to find information is faster than a conventional search because in most cases LookAt goes directly to the message explanation.

You can access LookAt from the Internet at:

`http://www.ibm.com/eserver/zseries/zos/bkserv/lookat/`

or from anywhere in z/OS where you can access a TSO/E command line (for example, TSO/E prompt, ISPF, z/OS UNIX System Services running OMVS). You can also download code from the *z/OS Collection* (SK3T-4269) and the LookAt Web site that will allow you to access LookAt from a handheld computer (Palm Pilot VIIx suggested).

To use LookAt as a TSO/E command, you must have LookAt installed on your host system. You can obtain the LookAt code for TSO/E from a disk on your *z/OS Collection* (SK3T-4269) or from the **News** section on the LookAt Web site.

Some messages have information in more than one document. For those messages, LookAt displays a list of documents in which the message appears.

Using LookAt to look up message explanations

LookAt is an online facility that allows you to look up explanations for most messages you encounter, as well as for some system abends and codes. Using LookAt to find information is faster than a conventional search because in most cases LookAt goes directly to the message explanation.

You can access LookAt from the Internet at:

`http://www.ibm.com/eserver/zseries/zos/bkserv/lookat/`

or from anywhere in z/OS where you can access a TSO/E command line (for example, TSO/E prompt, ISPF, z/OS UNIX System Services running OMVS). You can also download code from the *z/OS Collection* (SK3T-4269) and the LookAt Web site that will allow you to access LookAt from a handheld computer (Palm Pilot VIIx suggested).

To use LookAt as a TSO/E command, you must have LookAt installed on your host system. You can obtain the LookAt code for TSO/E from a disk on your *z/OS Collection* (SK3T-4269) or from the **News** section on the LookAt Web site.

Some messages have information in more than one document. For those messages, LookAt displays a list of documents in which the message appears.

How to contact IBM service

For immediate assistance, visit this Web site:
http://www.software.ibm.com/network/commserver/support/

Most problems can be resolved at this Web site, where you can submit questions and problem reports electronically, as well as 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 (1-800-237-5511). You will receive a return call within 8 business hours (Monday – Friday, 8:00 a.m. – 5:00 p.m., local customer time).

Outside of 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 147.

z/OS Communications Server information

This section contains descriptions of the documents in the z/OS Communications Server library.

z/OS Communications Server publications are available:

- Online at the z/OS Internet Library web page at http://www.ibm.com/servers/eserver/zseries/zos/bkserv
- In softcopy on CD-ROM collections.

Softcopy information

Softcopy publications are available in the following collections:

Titles	Order Number	Description
z/OS V1R4 Collection	SK3T-4269	This is the CD collection shipped with the z/OS product. It includes the libraries for z/OS V1R4, in both BookManager® and PDF formats.
z/OS Software Products Collection	SK3T-4270	This CD includes, in both BookManager and PDF formats, the libraries of z/OS software products that run on z/OS but are not elements and features, as well as the *Getting Started with Parallel Sysplex®* bookshelf.
z/OS V1R4 and Software Products DVD Collection	SK3T-4271	This collection includes the libraries of z/OS (the element and feature libraries) and the libraries for z/OS software products in both BookManager and PDF format. This collection combines SK3T-4269 and SK3T-4270.
z/OS Licensed Product Library	SK3T-4307	This CD includes the licensed documents in both BookManager and PDF format.
System Center Publication IBM S/390® Redbooks™ Collection	SK2T-2177	This collection contains over 300 ITSO redbooks that apply to the S/390 platform and to host networking arranged into subject bookshelves.

z/OS Communications Server library

z/OS V1R4 Communications Server documents are available on the CD-ROM accompanying z/OS (SK3T-4269 or SK3T-4307). Unlicensed documents can be viewed at the z/OS Internet library site.

Updates to documents are available on RETAIN® and in information APARs (info APARs). See “Information APARs” on page 131 for a list of the documents and the info APARs associated with them.

- Info APARs for OS/390® documents are in the document called *OS/390 DOC APAR and PTF ++HOLD Documentation* which can be found at http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/IDDOCMST/CCONTENTS.
- Info APARs for z/OS documents are in the document called *z/OS and z/OS.e DOC APAR and PTF ++HOLD Documentation* which can be found at http://publibz.boulder.ibm.com:80/cgi-bin/bookmgr_OS390/BOOKS/ZIDOCMST/CCONTENTS.

Planning and migration:

Title	Number	Description
z/OS Communications Server: SNA Migration	GC31-8774	This document is intended to help you plan for SNA, whether you are migrating from a previous version or installing SNA for the first time. This document also identifies the optional and required modifications needed to enable you to use the enhanced functions provided with SNA.
z/OS Communications Server: IP Migration	GC31-8773	This document is intended to help you plan for TCP/IP Services, whether you are migrating from a previous version or installing IP for the first time. This document also identifies the optional and required modifications needed to enable you to use the enhanced functions provided with TCP/IP Services.
z/OS Communications Server: IPv6 Network and Application Design Guide	SC31-8885	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	SC31-8775	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 in conjunction with the *z/OS Communications Server: IP Configuration Reference*.
z/OS Communications Server: IP Configuration Reference	SC31-8776	This document presents information for people who want to administer and maintain IP. Use this document in conjunction with the *z/OS Communications Server: IP Configuration Guide*. The information in this document includes: • TCP/IP configuration data sets • Configuration statements • Translation tables • SMF records • Protocol number and port assignments
z/OS Communications Server: SNA Network Implementation Guide	SC31-8777	This document presents the major concepts involved in implementing an SNA network. Use this document in conjunction with the *z/OS Communications Server: SNA Resource Definition Reference*.
z/OS Communications Server: SNA Resource Definition Reference	SC31-8778	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 in conjunction with the *z/OS Communications Server: SNA Network Implementation Guide*.
z/OS Communications Server: SNA Resource Definition Samples	SC31-8836	This document contains sample definitions to help you implement SNA functions in your networks, and includes sample major node definitions.
z/OS Communications Server: AnyNet SNA over TCP/IP	SC31-8832	This guide provides information to help you install, configure, use, and diagnose SNA over TCP/IP.
z/OS Communications Server: AnyNet Sockets over SNA	SC31-8831	This guide provides information to help you install, configure, use, and diagnose sockets over SNA. It also provides information to help you prepare application programs to use sockets over SNA.

Title	Number	Description
z/OS Communications Server: IP Network Print Facility	SC31-8833	This document is for system 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	SC31-8780	This document describes how to use TCP/IP applications. It contains requests that allow a user to log on to a remote host using Telnet, transfer data sets using FTP, send and receive electronic mail, print on remote printers, and authenticate network users.
z/OS Communications Server: IP System Administrator's Commands	SC31-8781	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	SC31-8779	This document serves as a reference for programmers and operators requiring detailed information about specific operator commands.
z/OS Communications Server: Quick Reference	SX75-0124	This document contains essential information about SNA and IP commands.

Customization:

Title	Number	Description
z/OS Communications Server: SNA Customization	LY43-0092	This document enables you to customize SNA, and includes the following: • 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 Application Programming Interface Guide	SC31-8788	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	SC31-8807	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.

Title	Number	Description
z/OS Communications Server: IP IMS Sockets Guide	SC31-8830	This document is for programmers who want application programs that use the IMS™ TCP/IP application development services provided by IBM's TCP/IP Services.
z/OS Communications Server: IP Programmer's Reference	SC31-8787	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	SC31-8829	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	SC31-8811	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.)
z/OS Communications Server: SNA Programmer's LU 6.2 Reference	SC31-8810	This document provides reference material for the SNA LU 6.2 programming interface for host application programs.
z/OS Communications Server: CSM Guide	SC31-8808	This document describes how applications use the communications storage manager.
z/OS Communications Server: CMIP Services and Topology Agent Guide	SC31-8828	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	GC31-8782	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: SNA Diagnosis Vol 1, Techniques and Procedures and *z/OS Communications Server: SNA Diagnosis Vol 2, FFST Dumps and the VIT*	LY43-0088 LY43-0089	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*	LY43-0090 LY43-0091	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	SC31-8790	This document describes the ELM, IKT, IST, ISU, 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
z/OS Communications Server: IP Messages Volume 1 (EZA)	SC31-8783	This volume contains TCP/IP messages beginning with EZA.
z/OS Communications Server: IP Messages Volume 2 (EZB)	SC31-8784	This volume contains TCP/IP messages beginning with EZB.
z/OS Communications Server: IP Messages Volume 3 (EZY)	SC31-8785	This volume contains TCP/IP messages beginning with EZY.
z/OS Communications Server: IP Messages Volume 4 (EZZ-SNM)	SC31-8786	This volume contains TCP/IP messages beginning with EZZ and SNM.
z/OS Communications Server: IP and SNA Codes	SC31-8791	This document describes codes and other information that appear in z/OS Communications Server messages.

APPC Application Suite:

Title	Number	Description
z/OS Communications Server: APPC Application Suite User's Guide	SC31-8809	This documents the end-user interface (concepts, commands, and messages) for the AFTP, ANAME, and APING facilities of the APPC application suite. Although its primary audience is the end user, administrators and application programmers may also find it useful.
z/OS Communications Server: APPC Application Suite Administration	SC31-8835	This document contains the information that administrators need to configure the APPC application suite and to manage the APING, ANAME, AFTP, and A3270 servers.
z/OS Communications Server: APPC Application Suite Programming	SC31-8834	This document provides the information application programmers need to add the functions of the AFTP and ANAME APIs to their application programs.

Redbooks

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

Title	Number
TCP/IP Tutorial and Technical Overview	GG24–3376
SNA and TCP/IP Integration	SG24–5291
IBM Communications Server for OS/390 V2R10 TCP/IP Implementation Guide: Volume 1: Configuration and Routing	SG24–5227
IBM Communications Server for OS/390 V2R10 TCP/IP Implementation Guide: Volume 2: UNIX Applications	SG24–5228
IBM Communications Server for OS/390 V2R7 TCP/IP Implementation Guide: Volume 3: MVS Applications	SG24–5229
Secureway Communications Server for OS/390 V2R8 TCP/IP: Guide to Enhancements	SG24–5631
TCP/IP in a Sysplex	SG24–5235
Managing OS/390 TCP/IP with SNMP	SG24–5866

Title	Number
Security in OS/390–based TCP/IP Networks	SG24–5383
IP Network Design Guide	SG24–2580
Migrating Subarea Networks to an IP Infrastructure	SG24–5957
IBM Communication Controller Migration Guide	SG24–6298

Related information

For information about z/OS products, refer to *z/OS Information Roadmap* (SA22-7500). The Roadmap describes what level of documents are supplied with each release of z/OS Communications Server, as well as describing each z/OS publication.

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 table below lists documents that may be helpful to readers.

Title	Number
z/OS Security Server Firewall Technologies	SC24-5922
S/390: OSA-Express Customer's Guide and Reference	SA22-7403
z/OS JES2 Initialization and Tuning Guide	SA22-7532
z/OS MVS Diagnosis: Procedures	GA22-7587
z/OS MVS Diagnosis: Reference	GA22-7588
z/OS MVS Diagnosis: Tools and Service Aids	GA22-7589
z/OS Security Server LDAP Client Programming	SC24-5924
z/OS Security Server LDAP Server Administration and Use	SC24-5923
Understanding LDAP	SG24-4986
z/OS UNIX System Services Programming: Assembler Callable Services Reference	SA22-7803
z/OS UNIX System Services Command Reference	SA22-7802
z/OS UNIX System Services User's Guide	SA22-7801
z/OS UNIX System Services Planning	GA22-7800
z/OS MVS Using the Subsystem Interface	SA22-7642
z/OS C/C++ Run-Time Library Reference	SA22-7821
z/OS Program Directory	GI10-0670
DNS and BIND, Fourth Edition, O'Reilly and Associates, 2001	ISBN 0-596-00158-4
Routing in the Internet , Christian Huitema (Prentice Hall PTR, 1995)	ISBN 0-13-132192-7
sendmail, Bryan Costales and Eric Allman, O'Reilly and Associates, 1997	ISBN 156592–222–0
TCP/IP Tutorial and Technical Overview	GG24-3376
TCP/IP Illustrated, Volume I: The Protocols, W. Richard Stevens, Addison-Wesley Publishing, 1994	ISBN 0-201-63346-9
TCP/IP Illustrated, Volume II: The Implementation, Gary R. Wright and W. Richard Stevens, Addison-Wesley Publishing, 1995	ISBN 0-201-63354-X
TCP/IP Illustrated, Volume III, W. Richard Stevens, Addison-Wesley Publishing, 1995	ISBN 0-201-63495-3
z/OS System Secure Sockets Layer Programming	SC24-5901

Determining if a publication is current

As needed, IBM updates its publications with new and changed information. For a given publication, updates to the hardcopy and associated BookManager softcopy are usually available at the same time. Sometimes, however, the updates to hardcopy and softcopy are available at different times. The following information describes how to determine if you are looking at the most current copy of a publication:

- At the end of a publication's order number there is a dash followed by two digits, often referred to as the dash level. A publication with a higher dash level is more current than one with a lower dash level. For example, in the publication order number GC28-1747-07, the dash level 07 means that the publication is more current than previous levels, such as 05 or 04.
- If a hardcopy publication and a softcopy publication have the same dash level, it is possible that the softcopy publication is more current than the hardcopy publication. Check the dates shown in the Summary of Changes. The softcopy publication might have a more recently dated Summary of Changes than the hardcopy publication.
- To compare softcopy publications, you can check the last two characters of the publication's filename (also called the book name). The higher the number, the more recent the publication. Also, next to the publication titles in the CD-ROM booklet and the readme files, there is an asterisk (*) that indicates whether a publication is new or changed.

Part 1. IPv6 Overview

This section contains the following chapters:

Chapter 1. Introduction

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, which is approximately 20 years old and is approaching the end of its physical limits. The most significant issue surrounding IPv4 is the growing shortage of IPv4 addresses. In theory, 32 bits allow over 4 billion nodes, each with a globally unique address. In practice, the interaction between routing and addressing makes it impossible to exploit more than a small fraction of that number of nodes. Consequently, there is a growing concern that the continued growth of the Internet will 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 have been kept in IPv6, and functions that did not work well in IPv4 have been removed.

z/OS Communications Server Version 1 Release 4 is the first release to incorporate IPv6 features. Not all IPv6 features are supported in this release. This release enables you to do the following:

- Build an IPv6 network
- Start using IPv6-enabled applications
- Enable existing IPv4 applications to be IPv6 applications

This document describes the support available and how to implement it. This chapter discusses some of the major differences between IPv4 and IPv6.

For more information on some of the features that are not yet supported, refer to Part 4, "Advanced topics" on page 105.

Expanded routing and addressing

IPv6 uses a 128-bit address space, which has no practical limit on global addressibility and provides 340 billion billion billion billion unique addresses. This is enough addresses so that every person can have a single IPv6 network with as many as 18 000 000 000 000 000 000 nodes on it, and still the address space would be almost completely 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

As important as the expanded address space is the use of hierarchical address formats. The IPv4 addressing hierarchy includes network, subnet, and host components in an IPv4 address. IPv6, with its 128-bit addresses, 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 as follows:

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. Most IPv6 extension headers are not examined or processed by intermediate nodes, in contrast to IPv4.

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 further 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 CS does not support DHCPv6.

Manual configuration of addresses may be used in environments where complete local control is required (as with VIPA or additional LOOPBACK addresses).

New protocol for neighbor node interaction

Neighbor Discovery (NeD) corresponds to a combination of the IPv4 protocols ARP, ICMP Router Discovery, and ICMP Redirect. Nodes (hosts and routers) use Neighbor Discovery to determine the link-layer addresses for neighbors known to reside on attached links and to quickly purge cached values that become invalid. Hosts also use Neighbor Discovery to find neighboring routers that are willing to forward packets on their behalf. Neighbor Discovery 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:

- Obtain configuration information which includes:
  - Router Discovery, which defines how hosts can automatically locate routers that reside on an attached link.
  - Prefix Discovery, which specifies how hosts discover the set of prefixes that are defined as being on-link (IPv6 address prefixes that reside on the shared link, such as an ethernet link), as well as those which are to be used when implementing Stateless Address Autoconfiguration.
  - Parameter Discovery, which 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's 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 should 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 should 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. The following chart provides a quick reference of these differences:

Table 1. IPv4/IPv6 comparison

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, refer to Chapter 2, “IPv6 addressing” on page 9.
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 refer to “Multicast scope” on page 17.
Fragmentation is supported at originating hosts and intermediate routers.	Fragmentation is not supported at routers. It is only supported at the originating host. For more information refer to “Fragmentation in an IPv6 network” on page 21.
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 refer to “Extension headers” on page 21.
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 refer to “Option to provide QoS classification data” on page 98.
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, refer to “Router advertisements” on page 25. 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 refer to “Address resolution” on page 28.
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 refer to “Multicast Listener Discovery (MLD)” on page 24.
Addresses must be configured either manually or through DHCP.	Addresses may be automatically assigned using stateless address autoconfiguration, assigned using DHCPv6, or manually configured. DHCPv6 is not supported in z/OS CS V1R4.

Table 1. IPv4/IPv6 comparison (continued)

IPv4	IPv6
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. For more information refer to "DNS" on page 54.
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. For more information refer to "Resolving names into IPv6 addresses" on page 55.
For QoS, IPv4 supports both differentiated and integrated services.	Differentiated and integrated services are both supported. In addition, IPv6 provides flow label that can be used for more granular treatment of packets. QoS is not supported in z/OS CS V1R4.

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. There is no support for an IPv6-only stack. There are several advantages of running in a dual-mode stack configuration:

- 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.

For more detailed information on dual-mode stack support, refer to "Dual-mode stack" on page 36.

Chapter 2. IPv6 addressing

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.

There are 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:

  ```
  FEDC:BA98:7654:3210:FEDC:BA98:7654:3210
  ```

  ```
  1080:0:0:0:8:800:200C:417A
  ```

  Note that it is not necessary 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 bullet).

- Due to some methods of allocating certain styles of IPv6 addresses, it will be common for addresses to contain long strings of zero bits. In order to make writing addresses containing zero bits easier, a special syntax is available to compress the zeros. The use of :: indicates multiple groups of 16 bits of zeros. The :: can only appear once in an address. The :: can also be used to compress both leading and trailing zeros in an address.

  For example the following addresses:

  ```
  1080:0:0:0:8:800:200C:417A  a unicast address
  FF01:0:0:0:0:0:0:101        a multicast address
  0:0:0:0:0:0:0:1             the loopback address
  0:0:0:0:0:0:0:0             the unspecified addresses
  ```

  may be represented as:

  ```
  1080::8:800:200C:417A       a unicast address
  FF01::101                   a multicast address
  ::1                         the loopback address
  ::                          the unspecified addresses
  ```

- 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 6 high-order 16-bit pieces of the address, and the d's are the decimal values of the 4 low-order 8-bit pieces of the address (standard IPv4 representation). This is used for IPv4-compatible IPv6 addresses and IPv4-mapped IPv6 addresses. These types of addresses are used to hold embedded IPv4 addresses in order to carry IPv6 packets over IPv4 routing infrastructure. The address can be expressed in the following manner:

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

  or in compressed form:

  ```
  ::13.1.68.3
  ::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
is an IPv6 address in any of the notations listed above.

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

For example, the following are legal representations of the 60-bit prefix 12AB00000000CD3 (hexadecimal):

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

The following are not legal representations of the preceding prefix:

```
12AB:0:0:CD3/60  -  may drop leading zeros, but not
trailing zeros, within any 16-bit chunk of the address.

12AB::CD30/60  -  address to left of "/" expands to

12AB:0000:0000:0000:0000:0000:0000:CD30

12AB::CD3/60  -  address to left of "/" expands to

12AB:0000:0000:0000:0000:0000:0000:0CD3
```

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 follows:

```
the node address      12AB:0:0:CD30:123:4567:89AB:CDEF

and its subnet number 12AB:0:0:CD30::/60

can be abbreviated as 12AB:0:0:CD30:123:4567:89AB:CDEF/60
```

IPv6 address space

The type of a IPv6 address is identified by the high-order bits of the address, as follows:

Table 2. Address type representation

Address type	Binary prefix	IPv6 notation
Unspecified	00...0 (128 bits)	::/128
Loopback	00...1 (128 bits)	::1/128
Multicast	11111111	FF00::/8
Link-local unicast	1111111010	FE80::/10
Site-local unicast	1111111011	FEC0::/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 may be listening on an Anycast address. Like unicast, a packet sent to an Anycast address will be 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 on different IPv6 addresses, refer to “Categories of IPv6 addresses” on page 12.

IPv6 addressing model

IPv6 unicast addresses of all types (excluding loopback and unspecified) may be assigned to a node’s interfaces.

All physical interfaces (excluding VIPA and loopback) are required to have at least one link-local unicast address. z/OS CS only allows a single link-local address per interface. Other platforms may have more than one. A single interface may be assigned multiple unicast or anycast IPv6 addresses. Multiple IPv6 multicast groups of any scope may be joined on a single interface. A unicast address or a set of unicast addresses may 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 may 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 may 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, site-local, or global scope, while a multicast address supports interface-local, link-local, subnet-local, admin-local, site-local, organization-local, and global scopes. See “Unicast IPv6 addresses” on page 12 and “Multicast IPv6 Addresses” on page 17 for more discussions on 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, while the set of links within a site and all directly attached interfaces comprise an instance of a site-local scope zone. A scope zone has the following properties:

- A scope zone is comprised of a contiguous set of interfaces and the links to which the interfaces are attached.
- An interface can only belong to 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, and can be connected to multiple site-local scope zones if it is directly connected to more than one site.
- 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 which contain a source or destination address of a given scope can only be routed 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. For instance, the same site-local address may be assigned to different interfaces, whether owned by the same node or different nodes, as long as the interfaces are not in the same site-local scope zone.
- Scope zones associated with the inbound and intended outbound interfaces are compared to determine if 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 only valid 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. Three 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, site-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. Anycast addresses are not supported in z/OS CS V1R4.

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

Unicast IPv6 addresses

IPv6 unicast addresses are aggregatable with prefixes of arbitrary bit-length similar to IPv4 addresses under Classless Interdomain Routing (CIDR).

There are several types of unicast addresses in IPv6, in particular global unicast, site-local unicast, and link-local unicast. There are also some special-purpose subtypes of global unicast, such as 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:

Network Prefix
The network prefix is 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
The subnet ID is used by an individual organization to identify subnets within its site. The organization can use these 16 bits to create 65 536 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	network prefix	subnet ID	interface ID

Figure 3. Global unicast address format

Local use address

There are two types of local-use unicast addresses defined, link-local and site-local. The link-local address is for use on a single link and the site-local address is for use in a single site.

Link-local addresses: Link-local addresses have the following format:

10 bits	54 bits	64 bits
1111111010	0	interface ID

Figure 4. Link-local address format

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 may be used to communicate with other nodes on the same link. A link-local address is automatically assigned.

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

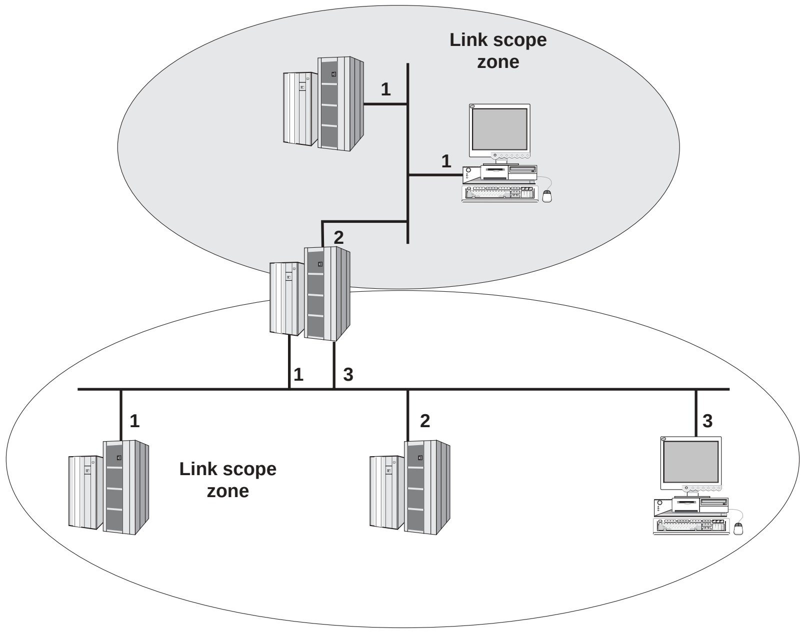

Figure 5. Link-local scope zones

Figure 5 depicts two separate link-local scope zones. More than one interface may be connected to the same link for fault tolerance or extra bandwidth. Some nodes may allow the same link-local zone index to be assigned to each interface connected to the same physical link, while others may assign a unique link-local zone index to each interface even when more than one interface is connected to the same physical link. z/OS CS V1R4 takes the latter approach, assigning a unique link-local zone index to each physical interface.

Site-local addresses: Site-local addresses have the following format:

10 bits	38 bits	16 bits	64 bits
1111111011	0	subnet ID	interface ID

Figure 6. Site-local address format

Site-local addresses are designed to be used for addressing inside of a site without the need for a global prefix. A site-local address cannot be reached from another site. A site-local address is not automatically assigned to a node. It must be assigned using automatic or manual configuration.

Routers will not forward any packets with site-local source or destination addresses outside of the site.

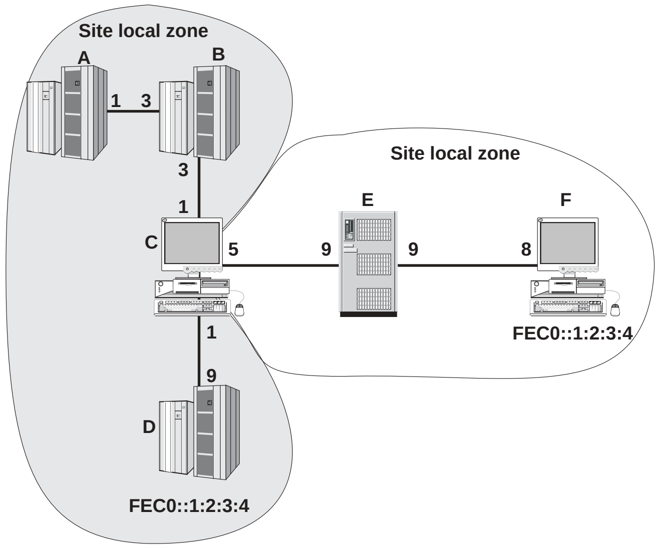

Figure 7. Site-local scope zones

Nodes connected to the same site-local scope zone may communicate with each other using site-local addresses. However, nodes which are not connected to the same site-local scope zone may not communicate using site-local addresses but must instead use global addresses.

Figure 7 depicts two site-local scope zones. In this configuration, node A can communicate with node D using site-local addresses since they are both within the same site-local scope zone. However, node A cannot communicate with node F using site-local addresses because the two nodes are not connected to the same site-local scope zone. Instead, node A must use global addresses when communicating with node F. Since node C is connected to both site-local scope zones, it may use the appropriate site-local address when communicating with both node A and node F.

z/OS CS V1R4 supports connecting to a single site-local scope zone and cannot be connected to two or more site-local scope zones at the same time. For example, z/OS CS V1R4 could be either node A or node F in Figure 7, as both are connected to only a single site-local scope zone, but could not be node C, as node C is connected to two site-local scope zones.

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 interface. It may 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.

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 will be dropped.

Unspecified address

The address 0:0:0:0:0:0:0:0 is called the unspecified address. It will not be 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.

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 occuring 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 8. 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 will be derived directly from that interface's link-layer address. z/OS CS will not allow two links to have the same local address. Some implementations may 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 CS builds the interface identifier when the interface becomes active, in the following way:

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.

24bits	16bits	24bits
MAC addr (bytes 1-3)	instance value	MAC addr (bytes 4-6)

Figure 9. 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.

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 may 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:

8	4	4	112 bits
11111111	flgs	scope	group ID

Figure 10. Multicast address format

flgs is a set of 4 flags:

0	0	0	T

Figure 11. 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).

The following table lists the defined values for the scope field:

Table 3. 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
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 may 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:

- 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 will work with a particular address.

Tentative

An address whose uniqueness on a link is being verified, prior to its assignment to an interface. A tentative address is not considered assigned to an interface in the usual sense. An interface discards received packets addressed to a tentative address, unless those packets are related to Duplicate Address Detection (DAD). For more information on DAD, refer to "Duplicate Address Detection (DAD)" on page 28.

Deprecated

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

Preferred

An address assigned to an interface whose use is unrestricted. Preferred addresses may be used as the source or destination address of packets sent from or to the interface, respectively.

Unavailable

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

Chapter 3. IPv6 protocol

This chapter describes the z/OS CS Version 1 Release 4 implementation of the IPv6 protocol. It is assumed that the reader is 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. This is in contrast to IPv4 options, which are contained in the IP header itself. Not all IPv6 extension headers are supported in z/OS CS V1R4. The V1R4 stack supports receipt of the following extension headers:

- Routing
- Fragmentation
- Hop-by-hop option
- Destination option

Authentication headers and Encapsulating Security Payload headers are unsupported for IPv6 in z/OS CS V1R4. Received IPv6 packets containing these headers will be silently discarded.

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. In order 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 detailed information refer to “Path MTU discovery” on page 22.

The IPv6 IP header does not contain information about fragments. The fragmentation extension header carries this information. z/OS CS allows for 2048 active IPv6 reassemblies in progress at any given time. z/OS CS 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. In order to try to ensure a UDP/RAW packet will not be dropped due to fragmentation, z/OS CS will always send it using the minimum MTU (1280) unless the MTU for the destination is learned from an ICMPv6 Packet Too Big message.

Consider a situation where the MTU was learned by way of Path MTU discovery. Then, the network topology changes, reducing the MTU to this particular destination. UDP/RAW will send with the original learned MTU, and will receive a Packet Too Big Message. In this case, this packet will be dropped, but subsequent sends will learn the changed MTU and will 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 usually 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 CS 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

IPv6 static routes (both replaceable and non-replaceable) are supported for z/OS CS V1R4 by using BEGINROUTES profile statements. The GATEWAY statement in the TCP/IP profile does not support IPv6 static routes.

Hosts can learn the network prefixes for all directly attached links from their routers. By checking to see if another host's IPv6 address is constructed from a network prefix of one of the directly attached links, it is possible to determine if that host is on a directly attached link or on a remote link. If it is on a directly attached link, data can be sent directly to that host without going through a router; otherwise, it must be sent through some router.

There are some limitations to be aware of. First, if a host has multiple interfaces attached to more than one link, it must decide which interface to send the packet over. If there are multiple routers on the link attached to the interface, it must decide which router should send the packet. To make these decisions, it needs a route in its routing table. Normally, this is populated by a dynamic routing protocol. z/OS CS V1R4 does not yet support dynamic routing protocols, and will use the default route when selecting which router on which interface to send the packet. This behavior may not produce the desired results, therefore, static routes may be defined to direct the traffic over the best interface using the appropriate router.

Dynamic routes for IPv6 are learned by router advertisements or ICMPv6 redirects. They are not a replacement for dynamic routing protocols such as RIP or OSPF. Replaceable static routes can be replaced by dynamic routes. If the dynamic route is deleted, the replaceable static route will be re-added. ICMPv6 redirects will replace static routes regardless of whether or not they are replaceable. Use the IGNOREREDIRECT keyword on the IPCONFIG6 statement in the TCP/IP profile to prevent the stack from adding routes learned by ICMPv6 redirects.

Since z/OS CS V1R4 does not support dynamic routing protocols, routes to VIPAs cannot be advertised. For this reason, using a network prefix defined as being on-link for the interfaces which are associated with the VIPA is recommended. In this way, routers and hosts will believe the VIPA is on a physical interface and will send Neighbor Discovery messages (the IPv6 equivalent of an ARP request) to get the MAC address of the interface. This is not typically the recommended way to set up VIPAs. Normally, they can be associated with interfaces on different LANs. But without a dynamic routing protocol, it is either take the recommended approach or define static routes at all routers on the same links as the z/OS system.

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, then the route with the most specific prefix is chosen.
- Default route.

For IPv4, there is a concept 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. Since 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. You can control whether multiple routes are selected by defining the MULTIPATH keyword on the IPCONFIG6 statement.

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

The MTU used when using a route that belongs to a multipath group is the minimum MTU of all routes in the multipath group.

How does a vary obey command affect routes?

When a vary obey command is issued and the profile contains a BEGINROUTES block, the following will occur:

- All replaceable static routes will be deleted and replaced by any replaceable routes defined in the BEGINROUTES block.
- All static routes will be deleted and replaced by any static routes defined in the BEGINROUTES block.
- All routes learned by way of ICMPv6 redirects will be deleted.
- Routes learned by way of router advertisements are not affected by the obey file processing, with one exception:
  - If the obeyfile profile contains a non-replaceable static route to the same destination for which a route exists that was learned by way of router advertisements, the router advertisement route will be deleted.

ICMPv6

The IP protocol concerns itself with moving data from one node to another. However, in order for IP to perform this task successfully, there are many other functions that need to be carried out: error reporting, route discovery, and diagnostics, among 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 (NeD), 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).

There are two types of ICMPv6 messages. Error messages are used to report errors in the forwarding or delivery of IPv6 packets. Informational messages provide diagnostic functions and additional host functionality such as MLD and NeD. Not all ICMPv4 messages have equivalents in ICMPv6. The following ICMPv6 messages are supported:

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

Multicasting

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.

Multicast Listener Discovery (MLD)

MLD is the protocol used by an IPv6 router to discover the presence of multicast listeners (that is, nodes wishing to receive multicast packets) on its directly attached links, and to discover specifically which multicast addresses are of interest to those listeners. This information is then provided to whichever multicast routing protocol is being used by the router, in order to ensure that multicast packets are delivered to all links where there are interested receivers. MLD is derived from IGMPv2. One important difference to note 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 and stops listening for a multicast address and responds to queries about multicast addresses. z/OS CS V1R4 implements the listener function.

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

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

An MLD query message is sent by a router 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) which 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.

Neighbor discovery (NeD)

Neighbor discovery 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, and ICMP Redirect). It maintains routes, MTU, retransmit times, reachability time, and prefix information based on information received from the routers. NeD uses Duplicate Address Detection (DAD) to verify the host's home addresses are unique on the LAN.

NeD uses Address Resolution to determine the link-layer addresses for neighbors on the LAN and Reachability Detection to determine neighbor reachability.

Router advertisements

Router advertisements are sent by routers to announce their availability. z/OS CS V1R4 receives router advertisements but does not originate them.

If the router advertisement indicates that the sending router should be used as a default router, a neighbor cache entry will be created/updated for the sending router and the following information will be stored in the host's routing table:

- IPv6 dynamic default route will be added (if not added by a previous advertisement).
- Next hop of default route will be the advertisement's source address.
- Interface of default route will be the interface on which the advertisement was received.
- Length of time that route will remain valid is set or reset using the Lifetime value from the advertisement.

If a non-replaceable static default route exists, then no dynamic default route will be added due to the received router advertisement.

If a replaceable static default route exists, the dynamic default route will be added due to the received router advertisement, replacing the replaceable route. The replaceable static default route will be reinstated if the dynamic default route is later removed.

If the router advertisement indicates that the sending router should not be used as a default router, the following will occur:

- If an IPv6 dynamic default route exists with the advertisement's source as its next hop and the receiving interface as its interface, it will be deleted.
- Any IPv6 dynamic indirect routes with the advertisement's source as its next hop and the receiving interface as its interface will be deleted.
- A neighbor cache entry will be created/updated for the sending router. The neighbor cache entry contains data from the router advertisement such as: indication that neighbor is a router, indication that neighbor is not a default router, and link-local and link-layer address of neighbor.

A router advertisement can contain Prefix Information Options. These options inform nodes of additional specific routes that are available to them, and indicate prefixes for autoconfiguring addresses. A Prefix Information Option contains the on-link and autonomous flags. The on-link flag, when set, indicates that 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. The autonomous flag, when set, indicates that autoconfigure processing needs to be performed for the prefix on the shared link. A Prefix Information Option can have just the on-link flag set, just the autonomous flag set, or both flags set.

The sending router indicates that a prefix is on-link by setting the on-link flag and specifying a nonzero Valid Lifetime value for the prefix. If the Prefix Information Option indicates that the prefix is on-link, the following will occur:

- An IPv6 dynamic direct route will be added (if not added by a previous advertisement).
- The destination of the route will be the prefix being processed.
- The interface of the route will be the interface on which the advertisement was received.
- The length of time that route will remain valid is set or reset using the Valid Lifetime value from the Prefix Information Option.

If a non-replaceable static direct route exists to this prefix via this interface, then the dynamic direct route will not be added. If a replaceable static direct route exists to this prefix via this interface, the dynamic direct route will be added, replacing the replaceable route. The replaceable static direct route will be reinstated if the dynamic direct route is later removed.

The sending router can indicate that a prefix is no longer on-link by setting the on-link flag and specifying a zero Valid Lifetime value for the prefix. In this case, if an IPv6 dynamic direct route exists with the prefix being processed as its destination and the receiving interface as its interface, it will be deleted.

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, the following will occur:

- An IPv6 home address will be added to the receiving interface for the autoconfigured address (if not added by a previous advertisement).
- An IPv6 implicit route will be added for the receiving interface and the autoconfigured address (if not added by a previous advertisement).
- The length of time that home address and implicit route will remain valid is set or reset using Valid Lifetime value from the Prefix Information Option.

- The length of time that home address will remain preferred (not deprecated) will be set or reset using the Preferred Lifetime value from the Prefix Information Option.

If addresses are manually configured for an IPv6 interface via the INTERFACE statement, autoconfiguration of addresses for that interface is disabled. If a prefix is not 64 bits in length, it will not be used for autoconfiguration of addresses. Unlike the prefix route and default route, the implicit route and home address cannot be immediately deleted. They must age out. If Valid Lifetime value is set to infinity, the implicit route and home address will not time out. For more information on autoconfiguration, refer to “Stateless address autoconfiguration” on page 29.

Route timeouts

The valid lifetime for each type of route will be updated (extending the life of the route) by the periodic receipt of router adverisements 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 a Valid Lifetime value of infinity, the associated implicit and/or prefix route is considered permanent and will not age unless a future Prefix Information Option for the prefix contains a non-infinity Valid Lifetime value.

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 a zero Valid Lifetime value. Expiration of the valid lifetime for an implicit route cannot be made immediate since the minimum lifetime allowed is two hours. It must age out naturally.

Vary obey considerations

If a non-replaceable static route in the obey file has the same destination as an existing route that was added due to a received Router Advertisement, the existing route will be replaced by the non-replaceable static route.

If the obey file specifies a manually configured home address for an interface that already has autconfigured addresses, the autoconfigured addresses will be deleted along with their associated implicit routes.

With the exception of the two preceding rules, all autoconfigured home addresses and routes added due to received Router Advertisements will be maintained through obey file processing.

Redirect processing

A node may 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 CS can be configured to ignore the IPv6 Redirects sent by routers by defining the IGNOREREDIRECT keyword on the IPCONFIG6 statement. If processing of Redirect messages is enabled, z/OS CS will begin using the new destination 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 will only be processed 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 will be deleted from the route table to keep it from being used by Multipath processing.

If Redirect processing is disabled, z/OS CS will silently discard the Redirect message.

Duplicate Address Detection (DAD)

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

- The interface is started (the autoconfigured link-local address and manually configured addresses/prefixes are checked).
- A vary obey is issued 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.

You can disable DAD checking by specifying DUPADDRDET 0 on the INTERFACE statement.

Duplicate Address Detection 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 due to 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 will be 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 CS host will still respond 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. Note that 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 known as *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 will soon be sent to this neighbor. Building the cache entry reduces the overhead of having to perfom the task of address resolution for the neighbor at a later time.

The NETSTAT ND/-n command can be issued to display information for a specific neighbor or all neighbor cache entries. It will display the neighbor link-layer address, state, whether the neighbor is a router or host, and if a router is a default router. The following are the 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 may provide confirmation).

Probe
: Neighbor solicitations are sent to verify neighbor reachability.

Neighbor unreachability detection

Neighbor unreachability detection is used to verify 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 one is received, then 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 will always be used to generate and assign a link-local address to a physical IPv6 interface. If it is unable to assign a link-local address, then interface activation will fail. No other addresses will be assigned to the interface (whether they are assigned using stateless address autoconfiguration or manual configuration) until a link-local address has been successfully assigned.

Stateless address autoconfiguration

The larger address field of IPv6 solves a number of problems inherent in IPv4, but the size of the address itself may 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. By doing this, a network can become operational with minimal action on the part of the TCP/IP administrator. Stateless autoconfiguration is supported for a physical interface (for example, QDIO) in z/OS CS 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 system startup, the host obtains an interface token from the interface hardware to create an interface ID. It generates its own addresses using a combination of router advertised prefixes and interface IDs.
2. Duplicate address detection is performed for the address. If a duplicate is not detected or DAD is disabled for the interface (DUPADDRDET 0 specified on the INTERFACE statement), the local address is added.
3. 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 and a valid lifetime. Router Advertisements contain the valid lifetime and preferred lifetime for a prefix. An IPv6 address goes through two phases to handle the expiration of an address gracefully:

   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

- A manually configured address/prefix on an interface disables stateless autoconfiguration for the interface.
- **INTERFACE *name* DELADDR *addr/prefix*** and **INTERFACE *name* DEPRADDR *addr/prefix*** issued via OBEYFILE are not valid for autoconfigured addresses.
- A VARY OBEYFILE profile that contains ADDADDR or DELADDR INTERFACE statements can affect stateless autoconfiguration:
  - **INTERFACE *name* ADDADDR *addr/prefix*** issued via OBEYFILE results in stateless autoconfigured addresses on the interface to be deleted. Stateless autoconfiguration capability will be disabled.
  - If the DELADDR removes the last manually configured address/prefix, stateless autoconfiguration is enabled and subsequent router advertisements can generate autoconfigured addresses.
- Autoconfigured addressses are not automatically added to DNS. Consider using VIPA addresses in conjunction with autoconfigured addresses.

IP address takeover following an interface failure

The TCP/IP stack in z/OS CS 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 CS 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 via the backup's link-layer (MAC) address. Users targeting the original IP address will see no outage due to the failure, and will be unaware that any failure occurred.

Since 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, or vice versa.

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 will be 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

All VIPAs must be manually configured. VIPA interfaces are always active. IPv6 VIPAs may be site-local or global. Link-local VIPAs are not allowed since link-local addresses are for use only on the associated LAN and there is no VIPA LAN.

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 on the algorithm, refer to "Default source address selection" on page 34.

VIPA recommendations

- Use different prefixes for IPv6 static VIPAs and for the IPv6 addresses assigned to real interfaces.

  Having static VIPAs configured with different prefixes than real addresses will reduce the likelihood of address collisions between the manually configured VIPAs and the autoconfigured addresses of the real interfaces. This is also necessary as Duplicate Address Detection (DAD) is not performed for VIPA addresses.
- To allow other hosts that share a LAN with the z/OS TCP/IP stack to access the IPv6 VIPAs without the need for manual route configuration, a router on each LAN should include the VIPA prefix in its router advertisements. The router advertisements should define the prefix as being on-link and should indicate that the prefix not be used for autoconfiguration.

  As a result of a router advertising the VIPA prefix as being on-link, each host will add a direct route to its route table indicating that the VIPA prefix can be reached on the link, without going through a router.

As a result of a router advertising that the VIPA prefix should not be used for autoconfiguration, each host on the LAN will not autoconfigure addresses on its real interfaces using the VIPA prefix.

Default address selection

The IPv6 addressing architecture allows multiple unicast addresses to be assigned to interfaces. These addresses may have different reachability scopes (link-local, site-local, or global). These addresses may 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 will result in more addresses per node. For example, a node may have multiple interfaces, some of them tunnels or virtual interfaces, or a site may have multiple ISP attachments with a global prefix per ISP.

The end result is that IPv6 implementations will often be faced with multiple possible source and destination addresses when initiating communication. It is desirable 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, will very often need to choose between IPv6 and IPv4 when initiating communication. For example, DNS name resolution may yield both IPv6 and IPv4 addresses with the network protocol stack having both IPv6 and IPv4 source addresses available. In such cases, a simple policy to always prefer IPv6 or always prefer IPv4 can produce poor behavior. As one example, suppose 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).

Default destination address selection

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

Establishing connectivity may depend on whether an IPv6 address or an IPv4 address is selected, thus making this sorting function even more important.

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

The default destination address selection algorithm takes a list of destination addresses and sorts them to generate 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 will be selected by TCP/IP when the application initiates an outbound connection or sends a datagram 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 itself does not explicitly specify a source IP address (using bind or ipv6_pktinfo) when initiating a connection or sending a datagram. If the application does explicitly specify a source address, then the destination address selected by this algorithm may not be optimal. The decision which the application makes may assume that a different source address will be used.

Rule 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 prior to the unreachable address.

Rule 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 interrogating the high order bits of the address. The destination address can be a multicast or unicast address. Unicast Link-Local is mapped to multicast Link-Local, unicast Site-Local to multicast Site-Local, and unicast Global scope to multicast Global scope.

Rule 3: Avoid deprecated addresses.
If one address is deprecated and the other is non-deprecated, then the non-deprecated address is placed prior to the other address.

Rule 4: Prefer matching address formats.
If one address format matches its associated source address format and the other destination does not meet this criteria, then place the destination with the matching format prior to the other address.

Rule 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.

Rule 6: Use 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.

Rule 7: Leave the order unchanged.
No rule selected a better address of these two; 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 will select 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 and/or dynamically generated) or the addresses configured for the outbound interface's SOURCEVIPA interface. Any address which is not preferred or deprecated is excluded from 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.

Rule 1: Prefer same address.
If either address is the destination address, choose that address as the source address and terminate the entire algorithm.

Rule 2: Prefer appropriate scope.
If the scope of one address is preferable to the scope of the other address, then the address with better scope is the better address of these two.

As an example, how is the scope of one source address (SA) preferable to the scope of another source address (SB) for the given destination address (D)?

- If scope of SA < scope of SB: If scope of SA < scope of D then SB is the best address of SA and SB; otherwise SA is the best address.
- If scope of SB < scope of SA: If scope of SB < scope of D then SA is the best address of SA and SB; otherwise SB is the best address.

Rule 3: Avoid deprecated addresses.
If one address is deprecated and the other is preferred, then the preferred address is the better address of these two.

Rule 4: Use longest matching prefix.
If one address has a longer CommonPrefixLength with the destination than the other address, then the address with the longer CommonPrefixLength is the better address of these two.

Rule 5: Leave the order unchanged.
No rule selected a better address of these two; they are equally good. Choose the first address as the better address of these two.

VIPA considerations with source address selection

If SOURCEVIPA is configured for the outbound interface and the application has not requested that SOURCEVIPA be ignored (via Ignore Source VIPA socket option), the source address will be selected from the SOURCEVIPA interface's addresses. Otherwise, source address will be selected from the outbound interface's addresses. Note that selection of a Source VIPA address for IPv6 is done differently from IPv4. It is determined by the SOURCEVIPAINTERFACE parameter 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 following set of rules:

Table 4. Source address selection

Source address selection for communication to IPv6 destinations		TCP, UDP, and RAW
IPCONFIG6 NOSOURCEVIPA	1. Is the socket already bound to a local IPv6 address?	Do not change the local address, use it as it is.
	2. Is the socket unbound (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. Is the socket already bound to a local IPv6 address?	Do not change the local address, use it as it is.
	2. Has setsockopt() with the NOSOURCEVIPA option been 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. Is there a SOURCEVIPAINTERFACE option on the IPv6 INTERFACE definition over which the IP packet is about to be sent?	Use the IPv6 source address selection algorithm to select an IPv6 VIPA address from the IPv6 virtual interface pointed to by the SOURCEVIPAINTERFACE option.
	4. Is there no SOURCEVIPAINTERFACE option on the IPv6 INTERFACE definition 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).

Migration and coexistence

How to enable IPv6 communication between IPv6 islands in an IPv4 world

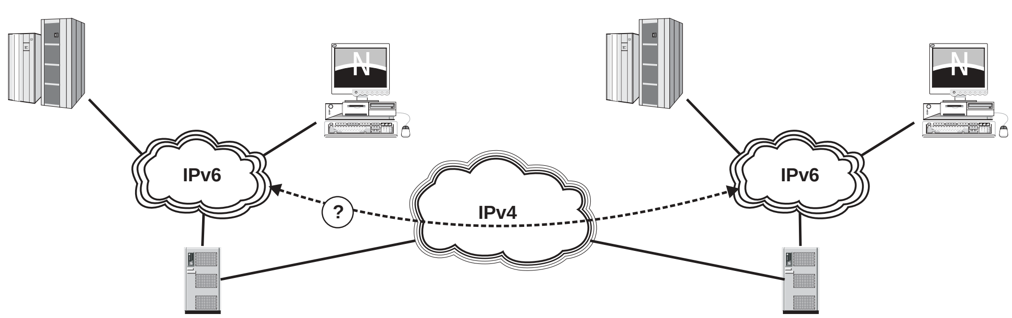

Figure 12. Communicating between IPv6 islands in an IPv4 world

Tunneling

Tunneling provides a way to utilize 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. That is, 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 users of the network, and is not detectable by network diagnostic tools such as traceroute.

z/OS CS does not support being a tunnel endpoint. This means that the z/OS CS stack will have to have an IPv6 interface connected to an IPv6 capable router. The router will be relied upon to handle all tunneling issues.

For more information refer to “Tunneling” on page 107.

How to enable end-to-end communication between IPv4 and IPv6 applications

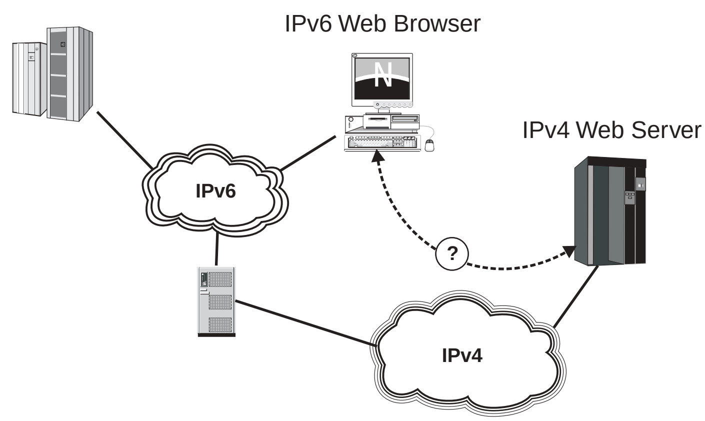

Figure 13. Communicating between IPv4 and IPv6 applications

Dual-mode stack

z/OS CS can be an IPv4-only stack or a 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. There is a socket option that makes an IPv6-enabled application require all peers to be IPv6. Refer to “Socket option to control IPv4 and IPv6 communications” on page 82 for detailed information on 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, then it cannot communicate with an IPv4 partner, since the native IPv6 address cannot be converted to an IPv4 address.

If a partner is IPv6, then all communication will use IPv6 packets.

If a partner is IPv4, the following will occur:

- Both source and destination will be IPv4-mapped IPv6 addresses.

- On inbound, the transport protocol layer will map the IPv4 address to its corresponding IPv4-mapped IPv6 address before returning to the application with AF_INET6 addresses.
- On outbound the transport protocol layer will convert the IPv4-mapped addresses to native IPv4 addresses and send IPv4 packets.

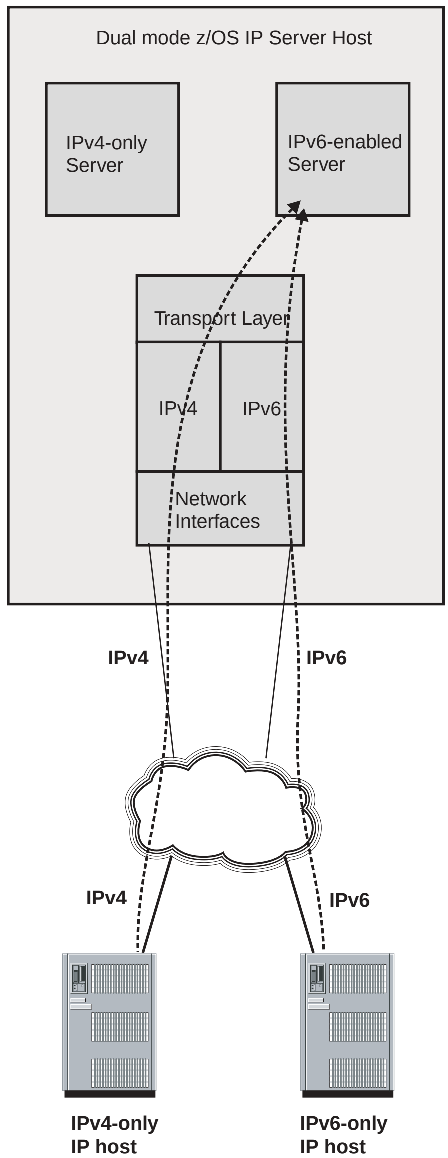

Figure 14. IPv6 application on dual-mode stack

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 will be native IPv4 addresses and the packet will be an IPv4 packet.

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

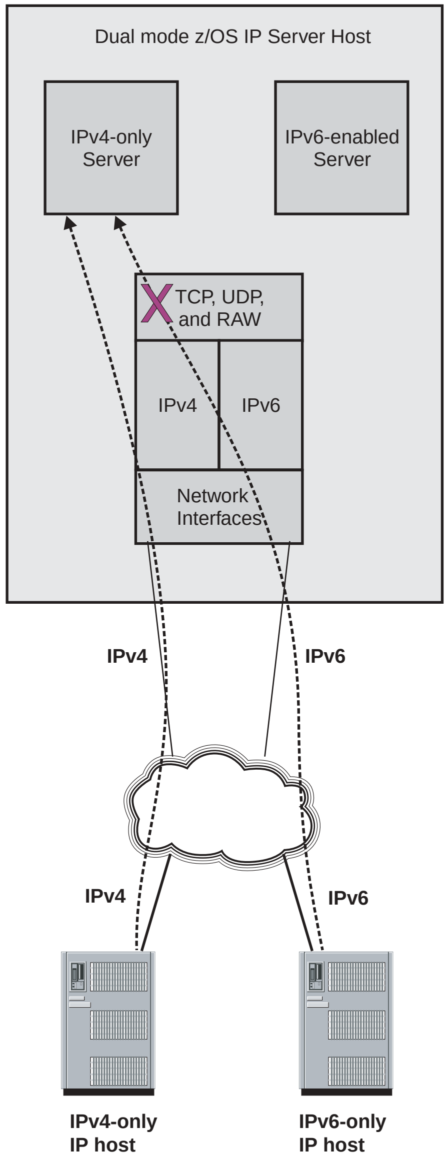

Figure 15. IPv4-only application on a dual-mode stack

Application Layer Gateways (ALG) and protocol translation

When IPv6-only nodes begin to appear in the network, AF_INET6 applications on these nodes may need to communicate with AF_INET applications. 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. 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.

There are numerous RFCs that describe solutions in this area. One solution is a SOCKS64 implementation that works as a 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 socksified 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.

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. There are several limitations with NAT-PT:

- It is mandatory that all requests and responses pertaining to a session be routed through the same NAT-PT translator.
- There is a protocol translation limitation since 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 need to be incorporated.
- Lack of end-to-end security. The two end nodes that seek IPSec network level security must both use IPv4 or IPv6.
- Translation of DNS messages and DNSSEC. An IPv4 end-node that demands DNS replies be signed will reject replies that have been tampered with by NAT-PT.

z/OS CS 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 running 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.

Considerations for configuring z/OS for IPv6

The following section describes some general considerations for configuring IPv6 on z/OS, including cases where multiple types of TCP/IP stacks are present. For example, some users may be using the z/OS CS Anynet Sockets over SNA stack and the z/OS CS TCP/IP stack on the same system. As a result the term *stack* or *TCP/IP stack* in the following sections is used as a generic term that describes a protocol stack that can be defined as a UNIX System Services AF_INET Physical File System (PFS) in the BPXPRMxx parmlib member, such as z/OS CS TCP/IP or z/OS CS Anynet Sockets over SNA.

IPv6 stack support

IPv4-only stack

Some TCP/IP stacks only support IPv4 interfaces and are only capable of sending or receiving 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.

Both z/OS CS TCP/IP and z/OS CS AnyNet® Sockets over SNA can be started as IPv4-only stacks.

IPv6-only stack

An IPv6-only stack supports IPv6 interfaces but 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 via an IPv6-only stack, but cannot send and receive IPv4 packets.

Neither z/OS CS TCP/IP nor z/OS CS AnyNet Sockets over SNA can 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. Such TCP/IP stacks are generally referred to as a dual-mode stack IP stacks. This does not mean there are two separate TCP/IP stacks running on such a node; it just means that the TCP/IP stack has built-in support for both IPv4 and IPv6.

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

z/OS CS TCP/IP can be started as a dual-mode stack, while AnyNet Sockets over SNA cannot.

INET considerations

IPv4-only stack

An IPv4-only stack supports AF_INET applications, but does not support AF_INET6 applications. There are two ways to start an IPv4-only stack in an integrated sockets environment:

- The first, and easiest, is to not code an AF_INET6 statement in BPXPRMxx. By not enabling AF_INET6, the underlying TCP/IP stack will be started as an IPv4-only stack, even if it is capable of supporting IPv6. This is the only way to start z/OS CS TCP/IP as an IPv4-only stack in an integrated sockets environment.
- The second way is to run a TCP/IP stack which is not capable of supporting IPv6, such as AnyNet Sockets over SNA. When starting a TCP/IP stack which does not support IPv6, the stack ignores any AF_INET6 definitions which may 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 only send and receive IPv4 packets over IPv4 interfaces. However, applications will be 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. To enable AF_INET6 support in an integrated sockets environment, the following two conditions must be met:

- AF_INET6 must be configured in BPXPRMxx. Note that AF_INET6 support may 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. Note that if a dual-mode capable TCP/IP stack is started before configuring BPXPRMxx then it will remain an IPv4-only stack as long as it remains active. However, if it is stopped and then restarted, it will restart as a dual-mode TCP/IP stack if AF_INET6 is configured in BPXPRMxx at the time it is restarted.

To enable AF_INET6 support for z/OS CS TCP/IP, z/OS CS TCP/IP must be started as a dual-mode stack. z/OS CS TCP/IP does not support being started as an IPv6-only stack. Stated another way, if AF_INET6 is coded in BPXPRMxx, AF_INET must also be coded. If it is not, then the z/OS TCP/IP stack will fail to initialize.

Common INET considerations

Enabling AF_INET6 support in a Common INET environment

To enable AF_INET6 support in a Common INET environment, the following two conditions must be met:

- AF_INET6 must be configured in BPXPRMxx. Note that AF_INET6 support may 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 will remain an IPv4-only stack as long as it remains active. However, if it is stopped and then restarted, it will restart as a dual-mode TCP/IP stack if AF_INET6 is configured in BPXPRMxx at the time it is restarted.

If either condition is not met, then AF_INET6 support is not enabled.

Note: Starting some z/OS CS TCP/IP stacks with AF_INET6 support and some without AF_INET6 support is not recommended. 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, allowing these TCP/IP stacks to become dual-mode stacks. Once this is done, all applications which are capable of opening AF_INET6 sockets should be stopped and restarted, which will allow the restarted applications to communicate over IPv4 and IPv6 networks.

Disabling AF_INET6 support in a Common INET environment

There are two ways to disable AF_INET6 support in a Common INET environment:

- Stop all active dual-mode TCP/IP stacks while IPv4-only stacks remain active. Applications will no longer be able to open AF_INET6 sockets, although they can continue to use any AF_INET6 sockets which are already open and not bound to one of the stopped dual-mode TCP/IP stacks. However, applications will be able to open AF_INET sockets.

- Dynamically disable AF_INET6 in BPXPRMxx, and stop all dual-mode TCP/IP stacks which are active. When restarted, the dual-mode capable TCP/IP stacks will start as IPv4-only stacks. This is, in effect, 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.

To allow AF_INET6 applications to communicate with an IPv4 peer over IPv4-only stack, Common INET provides AF_INET6 transformations. The AF_INET6 transformations convert AF_INET6 socket calls to the corresponding AF_INET socket calls prior to 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. Note that even with this transformation, AF_INET6 applications must use IPv4-mapped IPv6 addresses to communicate with IPv4 applications.

The following figure shows a mixture of dual-mode stacks and IPv4-only stacks:

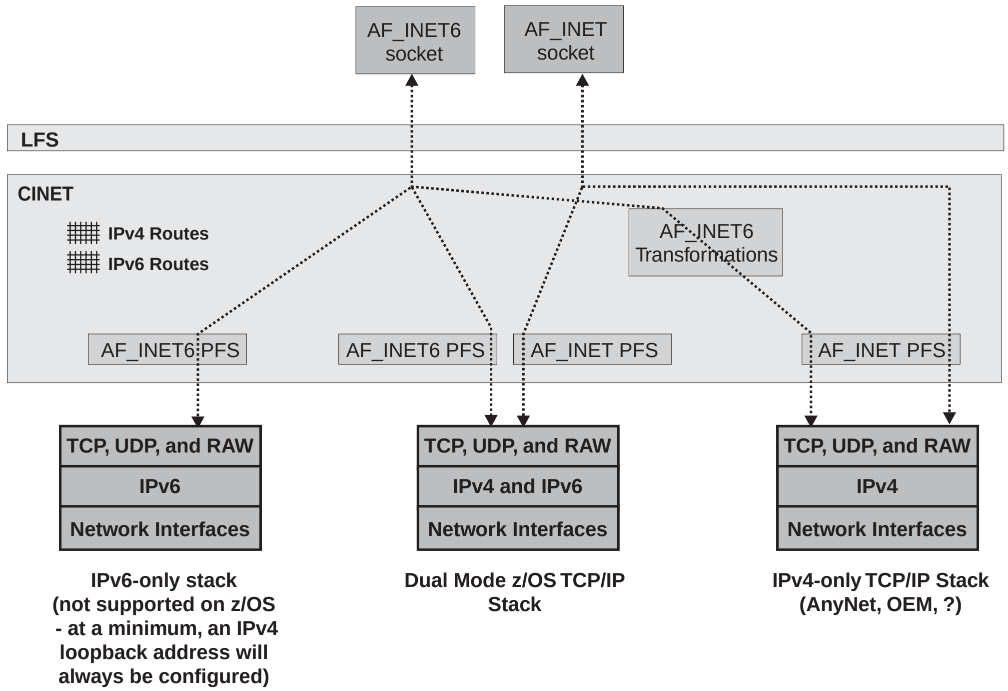

Figure 16. Mixing dual-mode and IPv4-only stacks

Configuration recommendations for a Common INET environment

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

If AF_INET6 support is dynamically configured in BPXPRMxx, it is recommended that all dual-mode capable TCP/IP stacks be stopped and restarted. Once the TCP/IP stacks have been stopped and restarted, all IPv6-enabled applications should be stopped and restarted.

Part 2. IPv6 enablement

This section contains the following chapters:

Chapter 4, “Configuring support for z/OS V1R4” on page 47 describes the IPv6 function provided in z/OS CS Version 1 Release 4 and how to enable it.

Chapter 5, “Configuration recommendations” on page 61 contains recommendations and guidance information for implementing the IPv6 functions provided in z/OS CS Version 1 Release 4.

Chapter 4. Configuring support for z/OS V1R4

Before you begin

Ensure important features are supported over IPv6

Table 5. IPv6 supported features for z/OS CS V1R4

Link-layer device support	Comments
OSA-Express in QDIO mode	Fast and Gigabit Ethernet support for IPv6 traffic is configured via an INTERFACE statement of type IPAQENET6.
Virtual IP Addressing support	
Virtual Device/Interface Configuration for static VIPA	This is configured via an INTERFACE statement of type VIRTUAL6.
IP routing functions	
Dynamic Routing - Autoconfiguration	
Static Route Configuration	This is configured via BEGINROUTES statement.
Multipath Routing Groups	Multipath is enabled via the IPCONFIG6 statement.
Accounting functions	
SMF	
Security functions	
Stack and Port Access Control	

Table 6. IPv6 supported applications for z/OS CS V1R4

Application	Protocol	Comments
Server applications		
FTP Server	TCP	Kerberos/GSS security protocol is not supported on IPv6 connections. RACF® allows all IPv6 clients to log in to FTP server. You cannot configure RACF to prevent this. RFC 2428 restriction: server requires data connection to use same protocol as control connection.
Inetd server	TCP	
Otelnetd server	TCP	Kerberos/GSS security protocol is not supported on IPv6 connections.
Orshd server	TCP	Kerberos/GSS security protocol is not supported on IPv6 connections.
Orexecd server	TCP	

Table 6. IPv6 supported applications for z/OS CS V1R4 (continued)

Application	Protocol	Comments
UNIX named (BIND 9.2 based)	TCP,UDP	
Client applications		
FTP client	TCP	Kerberos/GSS security protocol is not supported on IPv6 connections. Only connections to IPv4 FTP servers are socksified.
UNIX rexec client	TCP	
Command-type applications		
UNIX/TSO Netstat	UDP	
UNIX/TSO Traceroute	UDP,Raw	
UNIX/TSO Ping	Raw	
UNIX dig (BIND 9.2 based)	TCP,UDP	
UNIX nslookup (BIND 9.2 based)	TCP,UDP	
UNIX nsupdate (BIND 9.2 based)	TCP,UDP	

Assess automation and application impacts due to 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 since IP addresses now have a longer format.

Determine how remote sites will connect to the local host

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

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

In V1R4, z/OS does not support being a tunnel endpoint. However, z/OS may 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 which supports one of several tunneling protocols. The tunnel endpoint used by z/OS may be attached to the same LAN to which z/OS attaches, or may be attached to a remote network link. In either case, the presence of the tunnel endpoint is transparent to z/OS, and from the z/OS perspective traffic is routed over the native IPv6 network.

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 which is connecting. In general, this is not a good idea for IPv4, as the client address can often be unpredictable, either due to the client using DHCP to obtain its address, or due to the client accessing the server from behind a NAT (Network Address Translator) 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. And, 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.

The result is client IP addresses are unpredictable and subject to frequent change. In addition, it is possible, and even likely, that a server will be 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.

Considerations when using BIND parameter on PORT statement

The PORT statement reserves a port for the use of a particular server. It normally does not distinguish between IPv4 and IPv6; the port is reserved regardless of which type of address the application uses. The BIND keyword on the PORT statement allows you to force an INADDR_ANY listener to listen on a particular IP address. You may now specify an IPv6 address on this keyword. INADDR_ANY listeners will be converted to an IPv4 address, but will ignore an IPv6 address on the BIND keyword. IN6ADDR_ANY listeners will be converted to either an IPv4 address (the IPv4-mapped form of that address) or an IPv6 address, depending on what is specified with the BIND keyword.

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

The SHAREPORT keyword allows you to start multiple instances of the server and have connections automatically load balanced between them. All IPv4 connection requests will be load balanced between the set of IPv4 listeners (including AF_INET6 IN6ADDR_ANY listeners), while all IPv6 connection requests will be load balanced between the set of IPv6 listeners.

Security considerations

On z/OS V1R4, not all security features which are supported over an IPv4 transport are enabled when communicating via an IPv6 transport. For instance, Stack and Port Access Control, TLS, SSL, and Kerberos (Kerberos Version 5 and GSSAPIs) are enabled for both IPv4 and IPv6, whereas IPSec and Intrusion Detection are enabled for IPv4 but not for IPv6. Refer to Table 35 on page 119 for a list of features supported for IPv4 and/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 which are only supported over IPv4 are not exercised. The result is that for the same local application, some security features may be exercised when communicating via IPv4, but not when communicating via 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 prior to enabling AF_INET6 on a given LPAR. If only a subset of applications utilize such a security feature, then it is sufficient to ensure that those applications communicate only over the IPv4 transport. Several methods exist to ensure the IPv4 transport is used:

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

Application programming considerations

Refer to Part 3, “Application enablement” on page 65 for information on application programming considerations.

How to enable IPv6 support

The z/OS V1R4 Communications 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

Once a stack has been started, you cannot change its mode without stopping and restarting the stack.

You can configure either a single AF_INET or both AF_INET and AF_INET6. Although coding AF_INET6 alone is not prohibited, TCPIP will not start since the master socket is AF_INET and the call to open it will fail.

IPv4-only BPXPRMxx sample definition:

```
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 will be the IPv4 specified value.

CINET IPv4-only BPXPRMxx sample definition:

Multiple TCP/IP stacks in one MVS image or LPAR are only supported 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 prior to 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 stack (IPv4/IPv6) support is defined by using two NETWORK statements in the BPXPRMxx member. Each TCP/IP stack is defined in the BPXPRMxx parmlib member with SUBFILESYSTYPE. All z/OS CS stacks defined under the two NETWORK statements will be IPv4/IPv6 stacks. The following example shows the definitions for three dual (IPv4/IPv6) 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)
```

Enabling AF_INET6 support in z/OS Communications Server

Configuring z/OS CS IPv6 support

The following configuration statements have been added or changed to allow IPv6 addresses to be configured. Refer to the *z/OS Communications Server: IP Configuration Reference* for detailed information on each of these statements.

Changed statements:

BEGINROUTES
: Code this statement to add static IPv6 routes to the IP routing table. BEGINROUTES with IPv6 addresses coded will be rejected if the stack is not enabled for IPv6. The GATEWAY statement does not support IPv6 routes.

PKTTRACE
: IPv6 must be enabled for IPv6 addresses to be coded on these configuration statements.

PORT (BIND IP address)
: IPv6 must be enabled for IPv6 addresses to be coded on these configuration statements.

DELETE PORT (BIND IP address)
: IPv6 must be enabled for IPv6 addresses to be coded on these configuration statements.

IPCONFIG
: A FORMAT keyword has been added to control the format of the command output if the stack is not enabled for IPv6.

New statements:

INTERFACE
: An IPv6-enabled stack still uses DEVICE and LINK to define IPv4 interfaces. However, you cannot use DEVICE and LINK to define IPv6 interfaces. You must use the INTERFACE statement to define IPv6 interfaces. The stack must be enabled for IPv6 to use this statement.

IPCONFIG6
: This statement will be rejected if the stack is not enabled for IPv6. However, the SOURCEVIPA option has a dependency on the INTERFACE statement. You must specify the SOURCEVIPAINTERFACE keyword on the INTERFACE statement for each interface on which you desire that SOURCEVIPA take effect.

Resolver

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. Refer to “Name and address resolution functions” on page 72 for more details.
- New DNS resource records are defined to represent hosts with IPv6 addresses, and therefore new network flows between resolvers and name servers (in place of DNS IPv4 A records).
- A new algorithm is defined to describe how a resolver needs to sort a list of IP addresses returned for a multihomed host. Refer to “Default destination address selection” on page 32 for more information.

- New statements in the resolver configuration files are defined, and new search orders are implemented for local host tables processing.

Resolver configuration

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

ETC.IPNODES is a new local host file (in the style of /etc/hosts) which may contain both IPv4 and IPv6 addresses. IPv6 addresses can only be defined in ETC.IPNODES. 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

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 would be required to maintain two different local host files (for example, IPv4 addresses in HOSTS.LOCAL, IPv6 and IPv4 addresses in ETC.IPNODES) for your system.

A much simpler approach is to utilize the new COMMONSEARCH statement in the resolver setup file. By specifying COMMONSEARCH, the user indicates 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 utilize the same single file. The use of COMMONSEARCH not only reduces IPv6 and IPv4 searching to a single search order, but also reduces the z/OS UNIX and native MVS environments to a single search order as well.

For detailed information on search orders, refer to *z/OS Communications Server: IP Configuration Guide*.

IPv4-only configuration statements

Only IPv4 addresses may be specified on the NAMESERVER and NSINTERADDR TCPIP.DATA statements. This implies that all resolver communications with a name server will occur using AF_INET sockets, even when resource records related to IPv6 addresses are being queried.

The other statement in the TCPIP.DATA data set that currently supports IP address specification is the SORTLIST directive. SORTLIST is used for sorting IPv4 addresses only; the default destination address selection algorithm is used to sort IPv6 addresses.

IPv6/IPv4 configuration statements

COMMONSEARCH/NOCOMMONSEARCH resolver setup statement:

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

GLOBALIPNODES resolver setup statement:

Use this statement to specify the global local host file.

DEFAULTIPNODES resolver setup statement:

Use this statement to specify the default local host file.

Steps for implementing the resolver functions

- Add new resolver setup statements.
- Create the IPNODES local host files.
- Add IPv6 resource records to DNS.

For detailed information on setting up the resolver, refer to the *z/OS Communications Server: IP Configuration Guide*.

Resolver communications with the Domain Name System (DNS)

n order to retrieve IPv6 data from the proper name server, you must 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 data. The z/OS resolver can only use IPv4 to communicate with a name server.

DNS

With the introduction of 128-bit addresses, IPv6 makes it more difficult for the network user to be able to identify another network user by means of the IP address of the network device. The use of the DNS becomes even more of a necessity.

The name resolution process over an IPv6 network or of names with IPv6 addresses is no different than in an IPv4 environment. It uses the same recursive process, but different record types and content, and possibly a different network transport.

z/OS CS ships two name servers, one based on BIND 4.9.3 and the other based on BIND 9. The two z/OS CS name server versions are denoted as v4 and v9. Only the v9 server and associated name server tools are IPv6-capable.

z/OS CS V1R4 adds support for the DNS name server listening for and responding to queries over an IPv6 network using the following:

- New resource record types, AAAA and A6, which map the domain name to the IPv6 address
- New reverse domains, ip6.arpa and ip6.int, which are used to support address-to-domain name lookups

.

Resolving names into IPv6 addresses

Your existing DNS domains will require name-to-address mappings for your IPv6 interfaces, including your IPv6 static VIPAs. This can be done using the quad-A record (AAAA) or the A6 record. For more information on configuring AAAA and A6 records for IPv6, refer to the *z/OS Communications Server: IP Configuration Guide*.

AAAA records

These records provide the IPv6 equivalent for the IPv4 A records. AAAA records are very similar to A records in how they are administered in DNS and in how stub resolvers access them over the TCP/IP network. The basic difference between the two is that AAAA records support IPv6 addresses which are four times larger than IPv4 addresses (hence the quad A notation).

A6 records

These records provide another way to represent IPv6 addresses in DNS. The A6 resource record is experimental and is not recommended for use.

Resolving IPv6 addresses into names

This will involve the creation of entirely new reverse domains within the DNS. Address-to-name mapping for IPv4 is done with the in-addr.arpa. However, address-to-name mapping for IPv6 is done with the ip6.arpa and ip6.int domains.

ip6.arpa

The domain ip6.arpa is introduced to handle queries in a similar fashion as the existing in-addr.arpa domain. For example, the IPv6 address 3ffe:8050:201:1860:42::1 could be represented using the nibble format as:

```
$ORIGIN 0.6.8.1.1.0.2.0.0.5.0.8.e.f.f.3.ip6.arpa.
1.0.0.0.0.0.0.0.0.0.0.0.2.4.0.0 14400 IN PTR host.example.com.
```

ip6.int

The domain ip6.int is a second IPv6 reverse domain. It is almost identical to the ip6.arpa domain:

```
$ORIGIN 0.6.8.1.1.0.2.0.0.5.0.8.e.f.f.3.ip6.int.
1.0.0.0.0.0.0.0.0.0.0.0.2.4.0.0 14400 IN PTR host.example.com.
```

ip6.int was the original RFC standard domain for mapping IPv6 addresses to names. Though the use of this domain has been deprecated by the IETF, many older and relatively recent resolver implementations still rely on the ip6.int domain. Configuring both the ip6.arpa and ip6.int domains will ensure access to DNS reverse mapping records to users of all platforms.

DNS setup

Use the following steps as a guide to set up your DNS:

1. Add and modify statements in your name server configuration file.
   - Add new reverse zone statements.
   - Add IPv6-specific options (optional).
   - Modify options which can take IPv4 or IPv6 addresses to include IPv6 information (optional).
2. Add IPv6-specific records to your existing forward zones (for hosts that are now IPv6-capable).
   - IPv6 address records - AAAA and A6
3. Create new IPv6 reverse zone files.
   - IPv6 reverse domains: ip6.arpa and ip6.int

- Use the same PTR records from IPv4 with a similar label format.

Refer to *z/OS Communications Server: IP Configuration Guide* for detailed information regarding DNS setup.

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:

- FTP - All FTP exits have been enhanced to support IPv6 addresses except for FTPSMFEX. Samples for these exits are provided in SEZAINST. Refer to *z/OS Communications Server: IP Migration* for more information on changes to these exits:
  - FTCHKCMD
  - FTCHKCM1
  - FTCHKCM2
  - FTCHKJES
  - FTCHKPWD
  - FTPOSTPA
  - FTPOSTPR

Which applications started with inetd are IPv6 enabled?

The following z/OS UNIX applications support IPv6 addresses:

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

What has to be changed?

The inetd.conf file must be modified to support the IPv6-enabled applications. For the z/OS UNIX servers to support IPv6 connections, tcp6 must be specified for the protocol of the service name in the inetd.conf file. When tcp6 is defined, IPv4 clients are also supported.

The z/OS UNIX rsh server and Telnet server support Kerberos for IPv4 connections, but not for IPv6 connections.

How does IPv6 affect 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. In z/OS V1R2, a new type of TCP/IP SMF record, type 119, was introduced. The type 119 SMF records were created to provide a standardized structure for all SMF records provided by TCP/IP. This included a standard representation of IP addresses appearing across all type 119 records. This representation already uses the IPv6 address format, which in the V1R2 time frame will always contain IPv4-mapped IPv6 addresses. Including IPv6 addresses in these records should not require any changes to the SMF type 119 record formats or any exits that process this data. Also, note that the type 119 records constitute a superset of the older type 118 records in terms of data that is available. It is recommended that users exploiting IPv6 migrate to the SMF 119 record.

The following records support IPv6 addresses in V1R4:
- FTP Client Transfer Completion
- FTP Server Transfer Completion
- FTP Server Logon Failure
- Port Statistics
- TCP Connection Initiation
- TCP Connection Termination
- UDP Socket Close

All other Type 119 SMF records do not include IPv6 information (for example, IP statistics does not report IPv6 or ICMPv6 statistics).

Type 118 FTP client and server transfer completion records are generated for IPv6 connections. In this case, the FTP records will 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 on SMF records, see the *z/OS Communications Server: IP Configuration Guide*.

How does IPv6 affect SNMP?

Most of the SNMP functions are not yet IPv6 enabled. The SNMP TCP/IP Subagent supports TCP scalar counter MIB objects from RFC 2012 and UDP scalar counter MIB objects from RFC 2013. The values of these counters will reflect both IPv4 and IPv6 processing. But the TCP/IP subagent's interface, TCP connection table, and UDP listener table data only reflect IPv4 processing.

Monitoring the TCP/IP network

How does IPv6 affect Netstat?

1. In order to accommodate full IPv6 address information, almost all of the Netstat reports have been redesigned.

   If the TCP/IP stack is IPv6 enabled, most reports will be displayed in a different format that with IPv4. This may impact applications that are used to parse Netstat output. The same considerations apply to applications which use IP addresses in their automation since IP addresses now have a longer size. If the TCP/IP stack is not IPv6 enabled, the report format is unchanged unless the FORMAT LONG parameter is specified on the Netstat command or on the IPCONFIG PROFILE statement.
2. IPv6 statistic information is added to the Netstat STATS/-S report.
3. Information regarding whether the stack is IPv6 enabled or not is added to the Netstat UP/-u report.
4. For a server that opens an AF_INET6 socket, binds to INADDR6_ANY, and does a socketopt with IPv6_V6ONLY against the socket, the local address information in the connection related reports will contain the text (IPV6_ONLY).

```
Netstat ALLCONN/-a example on an IPv6 enabled stack:
 MVS TCP/IP NETSTAT CS V1R4       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
```

Control of output format

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

In order 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 will instruct Netstat to produce output in one of the above formats. FORMAT SHORT is only applicable 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. Whenever a user specifies the Netstat command line format option, it will override the stack-wide format parameter on an IPv4-only stack.

What has changed?

The following Netstat reports have been modified to support IPv6:

- Netstat ALL/-A
- Netstat ALLCONN/-a
- Netstat BYTEINFO/-b
- Netstat CONFIG/-f
- Netstat CONN/-c
- Netstat DEVLINKS/-d
- Netstat HOME/-h
- Netstat PORTLIST/-o
- Netstat ROUTE/-r
- Netstat SOCKET/-s
- Netstat STATS/-S
- Netstat TELNET/-t
- Netstat UP/-u

The following Netstat report is added to display Neighbor Discovery cache information:

- Netstat ND/-n

The IP address filter support, IPADDR/-I, is enhanced to accept both IPv4 and IPv6 addresses.

Note: The Netstat GATE/-g is not enhanced to support IPv6 routes. Netstat ROUTE/-r is the recommended alternative.

For more detailed information regarding Netstat, refer to *z/OS Communications Server: IP System Administrator's Commands*.

How does IPv6 affect Ping and Traceroute?

Ping and Traceroute provide the following support for IPv6:

- IPv6 IP addresses, or host names that resolve to IPv6 IP addresses, can be used for destinations.
- IPv6 IP addresses can be used as the source IP address for the command's outbound packets.
- IPv6 IP addresses or interface names can be used as the outbound interface.
- A new ADDRTYPE/-A command option can be specified 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

How does IPv6 affect IPCS?

IPCS formatting has been enhanced for IPv6 for TCPIPCS dump analysis and CTRACE components SYSTCPIP and SYSTCPDA. For detailed information regarding IPCS, refer to *z/OS Communications Server: IP Diagnosis*.

How does IPv6 affect packet and data tracing?

Packet and data trace functions have been enhanced for IPv6 to allowing tracing of IPv6 addresses. For detailed information regarding trace functions, refer to *z/OS Communications Server: IP Diagnosis*.

Chapter 5. Configuration recommendations

Connecting to an IPv6 Network

z/OS CS TCP/IP supports IPv6 network access via OSA-Express in QDIO mode using Fast Ethernet or Gigabit Ethernet. For LPAR to LPAR communication, IPv6 uses an intermediate LAN. This can be a shared OSA-Express adapter, two OSA-Express adapters on the same LAN, or two OSA-Express adapters on different LANs.

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.

To transport IPv6 traffic to another host, z/OS CS TCP/IP must be connected via an OSA-Express adapter to a native IPv6 LAN. Note that a router within the network may tunnel the IPv6 packet across an IPv4 network to a remote IPv6 LAN or host. However, z/OS CS TCP/IP cannot be the tunnel endpoint, and the tunneling by an intermediate router is transparent to z/OS CS TCP/IP.

IPv6 address assignment

Use stateless address autoconfiguration for physical interfaces

IPv6 addresses for physical interfaces may be manually defined or may be automatically assigned by stateless address autoconfiguration. It is recommended that stateless address autoconfiguration be used for this assignment. Using stateless address autoconfiguration reduces the amount of definition required to enable IPv6 support, while making future site renumbering simpler.

Use static VIPAs

Using static VIPAs removes hardware as a single point of failure for connections being routed over the failed hardware. It is recommended that at least one static VIPA be configured for each LAN to which z/OS CS TCP/IP is connected. Each VIPA thus configured should be associated with all OSA-Express adapters connected to that same LAN.

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

Selecting the network prefix

The network prefix for a static VIPA should be selected from the set of on-link prefixes which are advertised by one or more routers attached to the LAN. By using an on-link prefix, hosts and routers attached to the LAN will use neighbor discovery address resolution to obtain a link-layer address for the VIPA. z/OS CS TCP/IP will select a link-layer address of an attached physical interface when responding to the query, and the attached host or router will forward the packet to z/OS CS TCP/IP. This avoids the need to define static routes for VIPAs at hosts and routers attached to the same LAN as z/OS CS TCP/IP.

z/OS CS TCP/IP does not perform duplicate address detection for VIPAs, as they are not assigned to a physical interface attached to the LAN. To avoid possible address collisions, it is recommended that the network prefix used for static VIPAs is configured so that autonomous autoconfiguration is not enabled. In doing so, the network prefix will not be used by hosts for autoconfiguring IPv6 addresses using stateless address autoconfiguration.

Selecting the interface identifier

The interface identifier for a VIPA must be unique among all IP addresses which are created using the combination of network prefix and interface identifier. Any scheme may be used in generating the interface identifiers, so long as they are unique. By using a network prefix which is not used by stateless address autoconfiguration, it is only necessary to ensure the interface identifier is unique among all VIPAs which 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 may be used. After some time period, the deprecated network prefixes are deleted, along with all IPv6 addresses which use the network prefix.

For addresses which are autoconfigured, 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 which use the prefix should be deprecated using the INTERFACE DEPRADDR statement. Once the prefix has expired, addresses which use the prefix should be deleted using the INTERFACE DELADDR statement.

Update DNS definitions

Include static VIPAs in DNS

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

Define both 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 be 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 does not exist, data may be tunneled over the IPv4 network, even when a native IPv4 path exists.

This may lead to longer connection establishment to an AF_INET application which resides on a dual-stack host. The client will first attempt to connect using each IPv6 address defined for the dual-stack host before attempting to connect via IPv4. A well-behaved client will cycle through all the addresses returned and will, ultimately, connect 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, as well as other potential problems, it is recommended that two host names be configured in DNS. The existing host name should continue to be used for IPv4 connectivity, so as to minimize disruption when connecting to unmodified AF_INET server applications. A new host name should also be defined, for which both IPv4 and IPv6 should be configured. When connecting using the old host name, AF_INET6 clients will connecting using IPv4. When connecting using the new host name, AF_INET6 clients will attempt to connect using IPv6 and, that failing, will fall back and connect using IPv4.

Using two host names allows the client to choose the network path which will be 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.

It should be noted that 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 only necessary during the initial transition phase when native IPv6 connectivity does not exist and applications have not yet been enabled for IPv6. Once both of these occur, a single host name may be used.

Use source VIPA

It is recommended that source VIPA be configured on IPv6 hosts. Using source VIPA allows an IPv6 address to be resolved to a host name, assuming the recommendations in "Update DNS definitions" on page 62 are implemented.

Define static routes to improve network path selection

Hosts are able to learn default routes from routers attached to local LANs via neighbor discovery algorithms. Hosts may then use the default routes when sending packets to remote hosts. If a host selects a non-optimal router when sending data, the router may redirect the host to use a more optimal router when sending data to the remote host, as long as the optimal router is on the same LAN as the original router.

When a host is connected to multiple LANs, this processing may result in non-optimal network paths being used. For instance, if a host selects a router on one LAN, but the optimal router is on another LAN, the router on the first LAN cannot redirect the host to the second LAN. In this case, a static route may be configured to allow the host to initially select the optimal network path.

When defining static routes, it is recommended the following guidelines be followed:

Use subnet routes instead of host routes

Remote IP addresses are difficult to predict. When using extensions to stateless address autoconfiguration, some clients may change their IP addresses on a routine basis, such as once an hour or once a day. In addition, these addresses may be created using cryptographic algorithms, making it difficult to impossible to predict what IP address a client may 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, it is recommended that subnet routes be defined. 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 defining the gateway router for a static route, it is recommended that the link-local address for the router be used. Link-local addresses do not change as the result of site renumbering, minimizing 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 may be used to provide IPv6 connectivity to the two sites. z/OS CS 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 “How to enable IPv6 communication between IPv6 islands in an IPv4 world” on page 35 for more information on 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 CS 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 (ALG) and protocol translation” on page 38 for more information.

Part 3. Application enablement

Before reading this part, you should have a good understanding of the information presented in Part 1, “IPv6 Overview” on page 1.

This part contains the following chapters:

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 would 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 discusses advanced IPv6 API functions that can be used by specialized IP applications.

For detailed information on specific APIs, refer to the following documentation:

- TCP/IP socket APIs are defined in the *z/OS Communications Server: IP Application Programming Interface Guide*.
- UNIX (LE) C/C++ socket APIs are defined in the *z/OS C/C++ Run-Time Library Reference*.
- UNIX Systen Services Callable APIs are defined in the *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

Chapter 6. API support

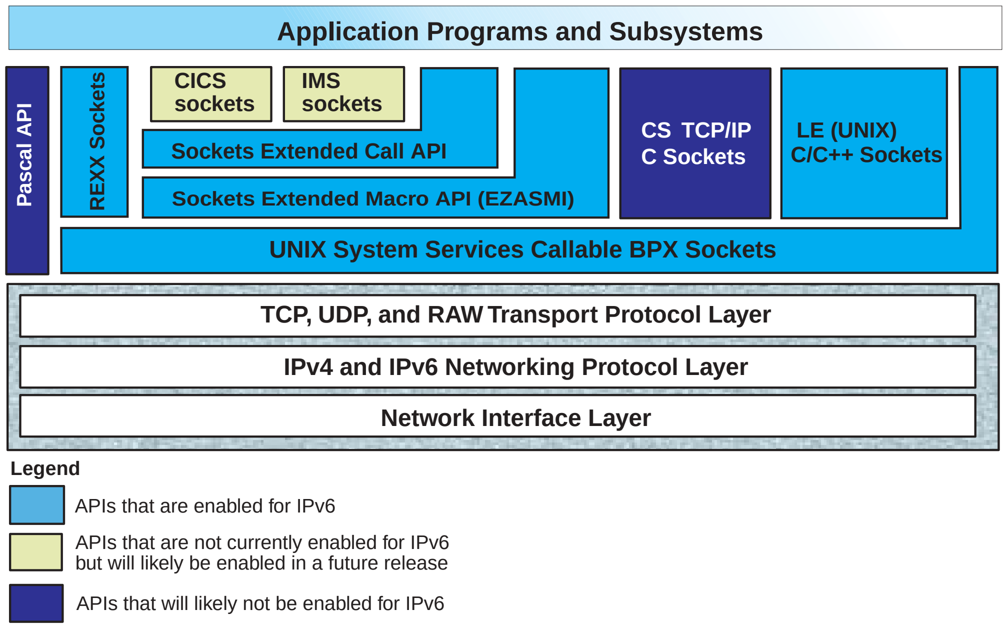

Figure 17. z/OS socket APIs

z/OS provides a versatile and diverse set of socket API libraries to support the various z/OS application environments. The figure above describes the relationship of the various z/OS socket APIs and the level of IPv6 present for each API.

There are 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 CS)

UNIX Socket APIs

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, refer to the *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 on this API, refer to the *z/OS C/C++ Run-Time Library Reference*.

Native TCP/IP socket APIs

The following TCP/IP Services APIs are included in this library. For more information on these APIs (excluding CICS), refer to *z/OS Communications Server: IP Application Programming Interface Guide*.

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

The CICS socket interface enables you to write CICS applications that act as clients or servers in a TCP/IP-based network. Applications can be written in C language, using the C sockets programming interface, or they can be written in COBOL, PL/I, or assembler, using the Extended Sockets programming interface. This API currently supports TCP/IP communications over IPv4 only but will likely support IPv6 communications in a future release. For more information, refer to the *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 executes 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 likely support IPv6 communications in a future release. For more information, refer to *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 only supports TCP/IP communications over IPv4. It is unlikely that this API will be enhanced to support IPv6 in the future. Applications using this API are encouraged to migrate their application to one of the other socket APIs that are IPv6 enabled.

TCP/IP C/C++ Sockets

The C/C++ Sockets interface supports IPv4 socket function calls that can be invoked from C/C++ programs. This API is very similar to the UNIX C socket API which 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. Existing applications that will be enabled for IPv6 should consider migrating to the UNIX C socket API.

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

- Resource Reservation Setup Protocol API (RAPI)
- Sun and NCS Remote Procedure Call (RPC)
- SNMP Agent Distributed Programming Interface (DPI®)
- X Window System and OSF/Motif
- X/Open Transport Interface (XTI)

These APIs do not support IPv6 communications.

Chapter 7. Basic Socket API extensions for IPv6

All examples in this chapter are shown using LE C; see *z/OS C/C++ Run-Time Library Reference* for details.

Introduction

While IPv4 addresses are 32 bits long, 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 via the API. This chapter describes the basic extensions to the socket interface and new features of IPv6 as described in the Internet Engineering Task Force (IETF) draft, *Basic Socket Interface Extensions for IPv6*.

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 discusses key differences between IPv4 and IPv6. It is assumed that you have a basic knowledge of IPv4 socket programming for clients and servers.

Protocol families

IPv4 socket applications use a protocol family of AF_INET (equivalent to PF_INET). 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 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 following information:

Table 7. sockaddr format for AF_INET

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 internet address
reserved	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 8. sockaddr format for AF_INET6

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 internet 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 the table below.

Table 9. 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	Refer to "Multicast IPv6 Addresses" on page 17

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 vice versa. The primary new APIs are getaddrinfo, getnameinfo, and freeaddrinfo. The APIs are designed to work with both IPv4 and IPv6 addressing. The use of these new APIs should be considered if an application is being designed for eventual use in an IPv6 environment.

The manner in which hostname (getaddrinfo) or IP address (getnameinfo) resolution is performed is dependent upon Resolver specifications contained in the Resolver setup files and TCPIP.DATA configuration files. These specifications determine whether the APIs will query a nameserver first, then search the local host tables, or whether the order will be reversed, or even if one of the steps will be eliminated completely. The specifications also control, if local host tables have to be searched, which tables will be accessed. For detailed information on Resolver setup, refer to "Resolver configuration" on page 53.

Protocol-independent nodename and service name translation

Getaddrinfo is conceptually a replacement for the existing gethostbyname and getservbyname APIs. Getaddrinfo takes an input hostname, or an input servicename, or both, and returns (when resolution is successful) one or more addrinfo structures. Getaddrinfo can also accept as input, a hostname or a servicename in numeric form, and will return the same value in presentation form using the addrinfo structure. An addrinfo structure contains the following output information:

- Pointer to sockaddr_in or sockaddr_in6 structure containing an IP address and service port
- 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 hostname (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. It is recommended that the application not manipulate the chain of addrinfo structures returned via getaddrinfo, but rather that the application simply return the entire chain, as received, back to the Resolver via freeaddrinfo.

In addition tohostname 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 via 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 primarily be used for cases where a *servicename* was being resolved, as might typically have been done previously via getservbyname.
- Various input flag settings:
  - AI_ADDRCONFIG
  - AI_ALL
  - AI_CANONNAME
  - AI_NUMERICHOST
  - AI_NUMERICSERV
  - AI_PASSIVE
  - AI_V4MAPPED

In the absence of any specific input from the application, the Resolver will assume that any sockaddr type is acceptable (that is, both IPv4 and IPv6 addresses) as output. Thus, by default, the Resolver will search for both IPv6 and IPv4 address via DNS and/or via local host files (such as /etc/hosts). Obviously, this may not always the best choice for the application issuing getaddrinfo. By using the above input fields, an application issuing getaddrinfo( ) can influence the processing performed by the Resolver function for that given 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). So, for example, if AF_INET is specified, the Resolver will not perform any searches for IPv6 addresses for *hostname*, since the output requested must be an IPv4 address.
- The application can specify:
  - that both IPv6 and IPv4 addresses should be returned
  - that IPv4 should only be returned if there are no IPv6 addresses resolved
  - that only IPv6 addresses should be returned
  - that only IPv4 addresses should be returned.

  This information, indicated by the input combination of family type and the AI_ALL and AI_V4MAPPED flags, controls to a large extent the types of searches performed by the Resolver during the course of the processing.
- The application can specify that IPv6 addresses should only be returned when the system has IPv6 interfaces defined, and can specify that IPv4 addresses should only be returned when IPv4 interfaces are defined. This preference, indicated via the AI_ADDRCONFIG flag, allows the application to eliminate resolution searches looking for addresses that cannot be used anyway 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 via the AI_PASSIVE flag, and is only applicable in the absence of an input *hostname*.
- 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 (for hostname) or AI_NUMERICSERV (for servicename) flags, which indicate 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 socktypes and protocols (TCP and UDP) be used for the getservbyname processing. This preference is indicated via the socktype and protocol settings.

With such a flexible interface, the application programmer must decide what inputs make sense for the capabilities of the application being created or modified. Table 10 on page 75 shows the two most likely application usages:

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

and the suggested getaddrinfo input settings that coincide with that functionality.

Table 10. 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 will return one or more addrinfo structures, each pointing to an IPv4 address saved in an AF_INET sockaddr. No addrinfos will be returned if there is no IPv4 interfaces defined on the system. No searches of any kind will be performed for IPv6 addresses as part of this request.
(IPv6 capable) Application wants all known addresses for hostname, in IPv6 format when the system supports IPv6, or in IPv4 format otherwise.	AF_UNSPEC	AI_ADDRCONFIG, AI_ALL -**or** -AI_ADDRCONFIG, AI_V4MAPPED, AI_ALL	Getaddrinfo will return one or more addrinfo structures, each pointing to a sockaddr structure. The sockaddrs will consist 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 will be returned only if there is an IPv6 interface defined on the system, and the IPv4 addresses will be returned only if there is an IPv4 interface defined.

An application with no interest in utilizing IPv6 will want to utilize the first entry in Table 10. Otherwise, if there is some interest in utilizing IPv6 functionality, an application would achieve the greatest amount of 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, then the Resolver will endeavor to return AF_INET6 sockaddrs to the application; otherwise, the Resolver will return AF_INET sockaddrs to the application. The choice of coding or not coding AI_V4MAPPED in this situation comes down to 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 10 should be sufficient for most application usages. However, there are other likely application capability models possible, and Table 11 on page 76 provides some guidance on how to code the Getaddrinfo invocations for those applications.

Table 11. 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 will return one or more addrinfo structures, each pointing to an IPv6 address saved in an AF_INET6 sockaddr. No addrinfos will be returned if there is no IPv6 interfaces defined on the system. No searches of any kind will be performed for IPv4 addresses as part of this request.
Application prefers IPv6 addresses, requires IPv6 address format, but can handle mapped IPv4 addresses if necessary.	AF_INET6	AI_ADDRCONFIG, AI_V4MAPPED	Getaddrinfo will return one or more addrinfo structures, each pointing to an AF_INET6 sockaddr. The addresses within the sockaddrs will 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 prefers IPv6 addresses, but can handle native IPv4 addresses if necessary.	AF_UNSPEC	AI_ADDRCONFIG	Getaddrinfo will return one or more addrinfo structures, each pointing to a sockaddr structure. The sockaddrs will consist of one of the following sets: • All AF_INET6 sockaddrs, containing IPv6 addresses, if there is an IPv6 interface defined on the system and IPv6 addresses exist for hostname • All AF_INET sockaddrs containing 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 will return one or more addrinfo structures, each pointing to an AF_INET6 sockaddr. The addresses within the sockaddrs will 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.

Table 11. Getaddrinfo application capabilities 2 (continued)

Application capabilities	Sockaddr family to request	Additional flags to set	Expected outputs
Application wants all known addresses for hostname, in native (IPv6 or IPv4) format.	AF_UNSPEC	AI_ADDRCONFIG, AI_ALL	Getaddrinfo will return one or more addrinfo structures, each pointing to a sockaddr structure. The sockaddr structures will be a mixture of AF_INET6 sockaddrs (each containing an IPv6 address) and AF_INET sockaddrs (each containing an IPv4 address). The IPv6 addresses will be returned only if there is an IPv6 interface defined on the system, and the IPv4 addresses will be returned only if there was an IPv4 interface defined for the system.
Application wants all known addresses for hostname, regardless of system connectivity, in native format.	AF_UNSPEC	AI_ALL	Getaddrinfo will return 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/or AF_INET sockaddrs (each containing an IPv4 address), depending on the address resolution.
Default settings when IPv6 is enabled on the system.	AF_UNSPEC	NONE	Getaddrinfo will return one or more addrinfo structures, each pointing to a sockaddr structure. The sockaddrs will consist of one of the following sets: • All AF_INET6 sockaddrs, containing IPv6 addresses, if there is an IPv6 address defined for hostname in any queried domain name server or defined in a local hosts table. No searches for IPv4 addresses are performed for hostname. • All AF_INET sockaddrs, containing IPv4 addresses, if there are no IPv6 addresses found for hostname. In either case, the actual availability of IPv6 or IPv4 interfaces on the system is not taken into consideration.

Table 11. Getaddrinfo application capabilities 2 (continued)

Application capabilities	Sockaddr family to request	Additional flags to set	Expected outputs
Default settings when IPv6 is not enabled on the system.	AF_UNSPEC	NONE	Getaddrinfo will return 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/or AF_INET sockaddrs (each containing an IPv4 address), depending on the address resolution performed. The actual availability of IPv6 or IPv4 interfaces on the system in not taken into consideration.

Regardless of the application model in use, and because output from getaddrinfo can be a chain of addrinfo structures, the recommendation is that the application 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 will clean up the storage when the process terminates, but storage constraints might occur before then if a large number of getaddrinfo APIs are performed.

Socket address structure to host name and service name

Conceptually, Getnameinfo is a replacement for the existing gethostbyaddr and getservbyport APIs. Getnameinfo takes an input *IP address*, or an input *port number*, or both, and returns (when resolution is successful) the host name and/or the service location. These parameters are passed in a sockaddr structure which also contains the address family.

In addition to *IP address* or *port number*, one of which must be present on a valid getnameinfo invocation, the application may specify additional input to the Resolver on the getnameinfo invocation. This input is optional. The input settings include the following (various input flag settings may 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_NUMERICXXXX flag is on.
- NI_NUMERICSERV specifies that the numeric form of the service name, the port number, is returned instead of the service name.
- 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 is to assume that the service is a stream service.

Address conversion functions

IP addresses often need to be provided to a socket application in character (string) format. Also, it is common for socket applications to need to display IP addresses in string format. Two new functions have been provided that work for IPv4 and IPv6 addresses.

inet_ntop	Convert a binary IP address (either v4 or v6) 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 are not usable for IPv6 addresses.

Table 12. Address conversion functions

Function	USS Assembler Callable services	C/C++ using LE	REXX	Socket Extended macro/call
inet_pton	no	yes	no	no
inet_ntop	no	yes	no	no
PTON	no	no	no	yes
NTOP	no	no	no	yes

Address testing macros

The following macros can be used to test for special IPv6 addresses.

Table 13. Address testing macros

Macros	Assembler Callable services	C/C++ using LE	REXX	Socket Extended macro/call
IN6_IS_ADDR_UNSPECIFIED	no	yes	no	no
IN6_IS_ADDR_LOOPBACK	no	yes	no	no
IN6_IS_ADDR_MULTICAST	no	yes	no	no
IN6_IS_ADDR_LINKLOCAL	no	yes	no	no
IN6_IS_ADDR_SITELOCAL	no	yes	no	no
IN6_IS_ADDR_V4MAPPED	no	yes	no	no
IN6_IS_ADDR_V4COMPAT	no	yes	no	no
IN6_IS_ADDR_MC_NODELOCAL	no	yes	no	no
IN6_IS_ADDR_MC_LINKLOCAL	no	yes	no	no
IN6_IS_ADDR_MC_SITELOCAL	no	yes	no	no
IN6_IS_ADDR_MC_ORGLOCAL	no	yes	no	no
IN6_IS_ADDR_MC_GLOBAL	no	yes	no	no

The macros behave in the following manner:

- The first seven 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 and IN6_IS_ADDR_SITELOCAL return true only for the two types of local-use IPv6 unicast addresses (link-local and site-local), and that by this definition, the IN6_IS_ADDR_LINKLOCAL macro returns false for the IPv6 loopback address (::1). These two macros do not return true for IPv6 multicast addresses of either link-local scope or site-local scope.

Interface identification

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

A new function, if_nameindex(), has been provided to allow socket applications to obtain a list of interface names and their corresponding index. Also, two new functions, if_nametoindex() and if_indextoname() allow translation of an interface name to its 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++ (LE applications) a new ioctl function code (SIOCGIFNAMEINDEX) is provided. Refer to Table 14 to determine which APIs support this new ioctl.

Table 14. Function calls

Function/IOCTL	USS Assembler Callable services	C/C++ using LE	REXX	Socket Extended macro/call
if_nametoindex	no	yes	no	no
if_indextoname	no	yes	no	no
if_nameindex	no	yes	no	no
SIOCGIFNAMEINDEX	yes	no	yes	yes
if_freenameindex	no	yes	no	no

Socket options to support IPv6 (IPPROTO_IPV6 level)

A group of socket options is defined to support IPv6. They are defined with a level of IPPROTO_IPV6. The individual options begin with IPV6_ . These options are only allowed 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 will have no effect. For example, the IPV6_UNICAST_HOPS socket option is used to set a hop limit value in the IPv6 header. Since IPv4 packets are used with IPv4-mapped IPv6 addresses, the hop limit value will not be used.

Note that the Sockets Extended macro/call APIs do not use level as an input to getsockopt() and setsockopt(). However, other IPv6 enabled APIs do. For detailed information on setsockopt() and getsockopt() input and output refer to the API specific documentation.

Table 15. Socket options for getsockopt() and setsockopt()

Socket options getsockopt() setsockopt()	USS Assembler Callable services	C/C++ using LE	REXX	Sockets Extended macro/call
IPV6_UNICAST_HOPS	yes	yes	yes	yes
IPV6_MULTICAST_IF	yes	yes	yes	yes
IPV6_MULTICAST_LOOP	yes	yes	yes	yes
IPV6_MULTICAST_HOPS	yes	yes	yes	yes
IPV6_JOIN_GROUP	yes	yes	yes	yes
IPV6_LEAVE_GROUP	yes	yes	yes	yes
IPV6_V6ONLY	yes	yes	yes	yes

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 will cause the stack default to be used.

A getsockopt() with this option will return the value set by a setsockopt(). If a setsockopt() has not been done, the stack's default value will be 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. Refer to the *z/OS Communications Server: IP Configuration Reference* for more information about the IPCONFIG6 statement.

This function is similar to the IPv4 socket option IP_TTL.

Options to control sending of multicast packets

The following three options allow an application to control certain features in the sending 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 will return the value set by setsockopt(). If a setsockopt() has not been done, a value of 0 will be returned.

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 between 0 and 255 inclusive. A socket option value of -1 is used to clear the socket option. This will cause the default value of 1 to be used.

A getsockopt() with this option will return the value set by a setsockopt(). If a setsockopt() has not been done, the default value of 1 will be 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. Refer to the *z/OS Communications Server: IP Configuration Reference* for more information on the IPCONFIG6 statement.

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 or not. 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 will return the value set by a setsockopt(). If a setsockopt() has not been done, the default value of 1 (enabled) will be returned.

This function is similar to the IPv4 socket option IP_MULTICAST_LOOP.

Options to control receiving of multicast packets

IPV6_JOIN_GROUP
This socket option allows 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 will select 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.

Getsockopt() does not support this option.

This function is similar to the IPv4 socket option IP_ADD_MEMBERSHIP.

IPV6_LEAVE_GROUP
This socket option is used by 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.

Getsockopt() does not support this option.

This function is similar to the IPv4 socket option IP_DROP_MEMBERSHIP.

Socket 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 will enable the option; a value of 0 will disable the option.

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

If an application wants to enable this option, the setsockopt() must be set prior to 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. Check the API specific documentation for information on a specific socket option.

Chapter 8. Enabling an application for IPv6

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 sections in this chapter describe some of the general considerations involved in enabling an application for IPv6. Note that while many of the examples and references in this chapter assume the use of C/C++ sockets supported by the Language Environment (LE) 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 refer to Chapter 7, “Basic Socket API extensions for IPv6” on page 71 and Chapter 9, “Advanced socket APIs” on page 95 and the documentation for the specific API you are using. It is also assumed that readers of this chapter have some familiarity with IPv6 in general and IPv6 support on z/OS CS. Parts 1 and 2 of this publication serve as a good starting point for this background information.

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 will also continue to have IPv4 connectivity. This is accomplished with dual-mode stack support which 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 will continue to function, over an IPv4 network, without any changes. However, at some point during the IPv6 deployment process, some IP hosts may only have connectivity to IPv6 networks or have a TCP/IP protocol stack that is only capable of IPv6 communications. Various migration and coexistence techniques can be employed to allow IPv6-only hosts to communicate with IPv4-only applications as described in “Migration and coexistence” on page 35. However, in the absence of these mechanisms, an application will need 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 may or may 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 will be enabled on the systems that the application is running on. There may be exceptions to this rule. For example, applications may 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 run-time test to determine whether IPv6 is enabled on the system on which they are executing. If the system is not enabled for IPv6 then 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.

Determining if a system is enabled for IPv6 can be done 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 16. Using socket() to determine IPv6 enablement

Affected socket API call	Changes required
socket()	Specify AF_INET6 as the Address Family (or domain) parameter. This API call will fail if the system is not enabled for IPv6.

An alternative mechanism that can be used by TCP/IP client applications to determine whether IPv6 is enabled involves the use of the new getaddrinfo() API. 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 prior to 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

With this information, a client application can be coded in a manner that allows it to be protocol-independent without having to perform specific run-time checks to determine whether IPv6 is enabled or not and without having to have dual-path logic (IPv4 versus IPv6). An example of this approach follows:

```
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: %d\n",sock);
    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);
}
```

When this example executes on a system where IPv6 is not enabled, only IPv4 addresses will be returned in AF_INET format (in sockaddr_in structures). When this identical example executes on a IPv6-enabled system, both IPv4 and IPv6 addresses will be returned, and the IPv4 addresses will be 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 (sockaddr_in) 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. The following table describes the necessary changes:

Table 17. sockaddr structure changes

Affected Socket API calls	Changes required
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 sufficently 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 will depend on the address family of the input socket.
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

Since IPv6 addresses have a different format and size from IPv4 addresses, 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. Note that a formatted IPv6 address takes up significantly more space than a formatted IPv4 address (46 bytes versus 16 bytes) and this may affect the layout of any messages and displays that include an IP address.

Table 18. Address conversion function changes

Affected API call	Changes required
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 is 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 on resolver APIs, refer to “Name and address resolution functions” on page 72.

Table 19. Resolver API changes

Affected API call	Changes required
gethostbyname()	Use new getaddrinfo() API. These APIs 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 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 the following table.

Table 20. Special IPv6 address changes

Socket API calls	Changes required
Binding a socket to the IPv4 wildcard address (INADDR_ANY - 0.0.0.0)	Specify IPv6 INADDR6_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.

Note: Refer to Chapter 7, “Basic Socket API extensions for IPv6” on page 71 for details on any constant definitions available for these special IPv6 addresses and the socket API 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 will be 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 21. givesocket() and takesocket() changes

Affected API call	Changes required
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 more granularity in the scope of multicast addressing and new socket options to allow an application to exploit this support. The following table lists IPv4 multicast setsockopt() and getsockopt() options and the equivalent IPv6 multicast options.

Table 22. Multicast options

Multicast function	IPv4	IPv6
Level specified on setsockopt()/getsockopt()	IP_PROTO	IPPROTO_IPV6
Joining a multicast group	IP_ADD_MEMBERSHIP	IPV6_JOIN_GROUP
Leaving a multicast group	IP_DROP_MEMBERSHIP	IPV6_LEAVE_GROUP
Select outbound interface for sending multicast datagrams	IP_MULTICAST_IF	IPV6_MULTICAST_IF
Set maximum hop count	IP_MULTICAST_TTL	IPV6_MULTICAST_HOPS
Enabling multicast loopback	IP_MULTICAST_LOOP	IPV6_MULTICAST_LOOP

In addition to the changes in the setsockopt() and getsockopt() options, the input/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 refer to “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 may need to send them twice, once to an IPv4 multicast address and once to an IPv6 multicast address. Also, note that in order to perform this type of processing the application will need 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. An alternative solution may be to first enable all the receiver applications for IPv6 and then enable the sender applications.

IP addresses may 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 will require changes when enabling for IPv6, as the IPv6 addresses have a different format from IPv4 addresses. The following options may be considered in dealing with these changes:

- Determine whether IP addresses are really needed in the data exchanged by the applications.
- Change the processing of the partner applications 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 may have multiple IP addresses.
- In many cases you may 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. The following is a list of options that may be considered in 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
  - Make the decision based on the partner application's IP address. If the partner's source IP address is an IPv4 address then only use IPv4 addresses; otherwise, use an IPv6 address. This option may 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: %s\n",inet_ntoa(clientSa.sin_addr));
  c = rc;
  rc = write(c,"hello\n",6);
  close (s);
  close (c);
  return 0;
}
```

Figure 18. IPv4 TCP server program

Example of the simple TCP server program enabled for IPv6

The simple TCP server program is now shown with the changes (in bold) 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 %d, family is %d\n",s,family);
```

Figure 19. Simple TCP server program enabled for IPv6 (Part 1 of 2)

```
 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;
}
```

Figure 19. Simple TCP server program enabled for IPv6 (Part 2 of 2)

Chapter 9. Advanced socket APIs

The advanced socket API for IPv6 support includes:

- IPv6 RAW socket support.
- New socket options.
- New ancillary data objects on sendmsg/recvmsg.
- Ability to receive inbound packet information including arriving interface index, destination IP address, and hop limit via ancillary data.
- Ability to set outgoing packet information including interface to use, source IP address, and hop limit. This may be set by socket options or ancillary data with some restrictions.

Not all features of the advanced socket API for IPv6 have been implemented. UNIX System Services C/C++ and UNIX System Services 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.

Controlling 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. Once enabled, the option remains in effect for the socket until 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 via sendmsg(). The value of the ancillary data is only in effect for that packet. 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 100.

An application can also receive ancillary data on the recvmsg() API.

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_ respectively. These options are only allowed on AF_INET6 sockets. In most cases, these options can be set on an AF_INET6 socket that is using IPv4-mapped IPv6 addresses but will have no effect. For example, the IPV6_HOPLIMIT ancillary data option is used to set a hop limit value in the IPv6 header. Since IPv4 packets are used with IPv4-mapped IPv6 addresses, the hop limit value will not be used.

Table 23. Sockets options at the IPPROTO_IPV6 level

Socket options getsockopt() setsockopt()	Assembler Callable Services	C/C++ using LE	REXX	Sockets Extended macro/call
IPV6_USE_MIN_MTU	Y	Y	N	N
IPV6_RECVPKTINFO	Y	Y	N	N
IPV6_RECVHOPLIMIT	Y	Y	N	N
IPV6_PKTINFO	Y	Y	N	N

Table 23. Sockets options at the IPPROTO_IPV6 level (continued)

Socket options getsockopt() setsockopt()	Assembler Callable Services	C/C++ using LE	REXX	Sockets Extended macro/call
IPV6_CHECKSUM	Y	Y	N	N

Table 24. Ancillary data on Sendmsg() (Level = IPPROTO_IPV6)

Ancillary data on sendmsg()	Assembler Callable Services	C/C++ using LE	REXX	Sockets Extended macro/call
IPV6_USE_MIN_MTU	Y	Y	N	N
IPV6_PKTINFO	Y	Y	N	N
IPV6_HOPLIMIT	Y	Y	N	N
IP_QOS_ CLASSIFICATION	Y	Y	N	N

Table 25. Ancillary data on Recvmsg() (Level = IPPROTO_IPV6)

Ancillary data on recvmsg()	Assembler Callable Services	C/C++ using LE	REXX	Sockets Extended macro/call
IPV6_PKTINFO	Y	Y	N	N
IPV6_HOPLIMIT	Y	Y	N	N

Option to request that minimum MTU size be used

IPV6_USE_MIN_MTU (used with TCP, UDP and RAW applications)

For IPv6, only the endpoint nodes may fragment a packet. Path MTU discovery determines the largest packet that can be sent to a destination without requiring fragmentation by an intermediate node (since that is not supported). In some cases, an application may not want have the overhead of 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, the minimum MTU size (1280 bytes) will be used to send packets which otherwise may have required fragmentation. This bypasses path MTU discovery processing.

This option can be enabled or disabled for a socket with a setsockopt(). This option can be enabled or disabled for a single send operation with ancillary data on the sendmsg(). A nonzero option value will enable the option, a value of 0 will disable the option.

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

Options to control the sending of packets for UDP and RAW applications

IPV6_PKTINFO (used with UDP and RAW applications)

The IPV6_PKTINFO option allows the application to provide two 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.

This option can be enabled or disabled for a socket with a setsockopt(). This option can be enabled or disabled for a single send operation with ancillary data on the sendmsg(). To disable the option, both the IPv6 address and the interface index should be specified as 0 in the option value.

A getsockopt() with this option will return the value set by setsockopt(). If a setsockopt() has not been done a value of 0 will be returned.

See “Understanding options for setting the source address” on page 102 for a discussion of the interaction of socket options and ancillary data for the setting of the source address. See “Understanding options for specifying the outgoing interface” on page 102 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.

This option can only be set with ancillary data on sendmsg(). It is not valid on setsockopt().

Note that the IPV6_UNICAST_HOPS socket option and the IPV6_MULTICAST_HOPS socket option are available to set a hop limit value also. See “Understanding hop limit options” on page 101 for a discussion of the interaction of IPV6_UNICAST_HOPS, IPV6_MULTICAST_HOPS and IPV6_HOPLIMIT.

Options to provide information about received packets

IPV6_RECVPKTINFO (used with UDP and RAW applications)

The IPV6_RECVPKTINFO socket option allows an application to receive two 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 will be returned as ancillary data on the recvmsg() API. The ancillary data level will be IPPROTO_IPV6. The option name will be IPV6_PKTINFO. For a detailed explanation of ancillary data see “Using ancillary data on sendmsg() and recvmsg()” on page 100.

This option can only be enabled or disabled with a setsockopt(). IPV6_RECVPKTINFO is not valid as ancillary data on sendmsg(). A nonzero option value will enable the option; a value of 0 will disable the option.

A getsockopt() with this option will return the value set by a setsockopt(). If a setsockopt() has not been done, the default value of 0 (disabled) will be 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 will be returned as ancillary data on the recvmsg() API. The ancillary data level will be IPPROTO_IPV6. The option name will be IPV6_HOPLIMIT. For a UDP or RAW application, if this option is enabled, the IPV6_HOPLIMIT ancillary data will be returned with each recvmsg(). For a TCP application, if this option is enabled, IPV6_HOPLIMIT ancillary data will only be returned on recvmsg() 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 100.

This option can only be enabled or disabled with a setsockopt(). IPV6_RECVHOPLIMIT is not valid as ancillary data on sendmsg(). A nonzero option value will enable the option; a value of 0 will disable the option.

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

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 will be computed and stored for outbound packets; the checksum will be verified for inbound packets. Note that this socket option is not applicable for ICMPv6 RAW sockets because the TCP/IP protocol stack will always provide checksum processing for them.

This option can only be enabled or disabled 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 between 0 and 65534 inclusive. A value of -1 causes the option to be disabled.

A getsockopt() with this option will return the value set by a setsockopt(). If a setsockopt() has not been done, the value of -1 (disabled) will be returned.

Option to provide QoS classification data

IP_QOS_CLASSIFICATION (used with TCP applications)

This option allows the application to provide QoS classification data. It is a z/OS CS-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, refer to the programming interfaces in the *z/OS Communications Server: IP Programmer’s Reference* for providing classification data to be used in differentiated services policies.

Socket option to support ICMPv6 (IPPROTO_ICMPV6 level)

Table 26. Sockets options at the IPPROTO_ICMPV6 level

Socket options getsockopt() setsockopt()	Assembler Callable Services	C/C++ using LE	REXX	Sockets Extended macro/call
ICMP6_FILTER	N	Y	N	N

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 may only be interested in receiving 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 will cause the TCP/IP protocol stack's default filter to be in effect.

A getsockopt() with this option will return the value set by a setsockopt(). If a setsockopt() has not been done, the TCP/IP protocol stack's default filter will be returned. For more information on default filtering, refer to "ICMP considerations" on page 103.

Note that the following macros are provided in the LE C/C++ environment to manipulate the filter value.

void ICMP6_FILTER_SETPASSALL(struct icmp6_filter *);	Specifies that all ICMPv6 messages are passed to the application.
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 will be 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 will not be 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 TCP/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 TCP/IP network.

These extensions to the sendmsg() and recvmsg() API are only available to applications using the following socket API libraries:

- z/OS IBM C/C++ sockets with the z/OS Language Environment(R). For more information on these APIs refer to the *z/OS C/C++ Run-Time Library Reference*.
- z/OS UNIX System Services Assembler Callable services socket APIs. For more information on these APIs refer to *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 UNIX System Services Assembler Callable services. For simplicity, only the C/C++ version of the data structures are shown in this section:

```
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     */
 };
```

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 will be 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 below) that describes the ancillary data (also defined in socket.h and BPXYMSGH respectively):

```
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[]; */
   };
```

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

Understanding 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 with three 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 precedence order that will be used to determine the hop limit value for a packet is as follows:

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 precedence order that will be used to determine the hop limit value for a packet is as follows:

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.

Understanding 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 precedence order that will be used to determine the source IP address for a packet is as follows:

1. If IPV6_PKTINFO ancillary data is specified on sendmsg() with a nonzero source IP address, use its value.
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.

Understanding options for specifying the outgoing interface

A UDP or RAW application can influence the outgoing interface for a packet with the IPV6_PKTINFO option and 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 responding to a peer, UDP and RAW applications should use the sockaddr_in6 structure which they received, and should not zero out the scope ID field. When sending an unsolicited packet (for example, not responding to one which was received), the scope ID field should be zero, and UDP and RAW applications should use the IPV6_PKTINFO or IPV6_MULTICAST_IF options to select the outgoing interfaces.

The precedence order that will be used to determine the outgoing interface for a packet is as follows:

1. If the send operation specifies a destination sockaddr structure with a scope ID then the scope ID will be used if valid (note that a scope ID should only be provided with a link-local address).
2. If IPV6_PKTINFO ancillary data is specified on sendmsg() with a nonzero interface index, use its value.
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.

Why use RAW sockets?

- 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 that of IPv4 RAW sockets, specifically with regards to the following:

- RAW protocol values allowed
- Application visibility of IP headers
- ICMP considerations
- Checksumming 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 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 selected IP header information for incoming and outgoing datagrams via socket options and ancillary data. For example:

- 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 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 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 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 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 off.

ICMP considerations

IPv6 RAW ICMPv6 applications can set the ICMP6_FILTER socket option to specify which ICMPv6 message types the socket will receive. By default, the following message types will be blocked (will not be 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

Checksumming 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 off.

Part 4. Advanced topics

Chapter 10. Advanced concepts and topics

Tunneling

Tunneling Overview

When IPv6 or IPv6/IPv4 systems are separated from other similar systems that they wish 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. z/OS CS cannot be an endpoint in V1R4.

While there are many tunneling protocols which may be used, all share certain common features and processing:

- The source tunnel endpoint determines that an IPv6 packet needs to be tunneled over an IPv4 network. How this is determined depends on the tunneling protocol which is being used. Once this decision has been made, the source tunnel endpoint adds an IPv4 header to the IPv6 packet. The protocol value in the IPv4 header is set to 41, indicating this is an IPv6 over IPv4 tunnel packet. The source and destination addresses in the IPv4 header are set based on the tunneling protocol which is being 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 notes 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.

The following is a subset of the available tunneling protocols, with descriptions of the more prevalent protocols. Others exist or are in the process of being defined.

The user should select one which is appropriate for their environment.

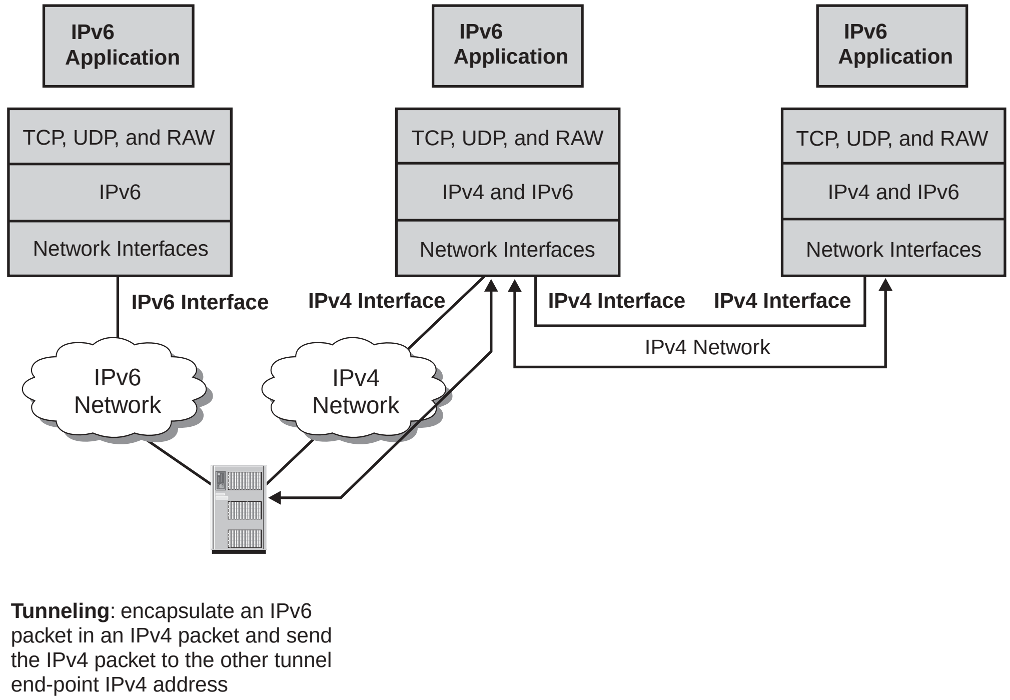

Figure 20. Tunneling

Configured tunnels

Configured tunneling refers to IPv6 over IPv4 tunneling where the IPv4 tunnel endpoint address is determined by configuration information on the encapsulating node. The tunnels may be unidirectional or bidirectional. Bidirectional configured tunnels behave 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.

The determination of which packets to tunnel is usually made by routing information on the encapsulating node. This is typically done via a routing table, which directs packets based on their destination address using the prefix mask and match technique.

Configured tunnels may 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 may be useful if the applications need to take advantage of IPv6 features which are not available in IPv4.

An IPv6/IPv4 host which is connected to datalinks with no IPv6 routers may 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 will only be used if a more specific route is not known.

Configured tunnels may also be used between routers, allowing isolated IPv6 networks to be connected via an IPv4 backbone. This connectivity can be accomplished by arranging tunnels directly with each IPv6 site to which connectivity is needed, but more typically is done by arranging a tunnel into a larger IPv6 routing infrastructure that can guarantee connectivity to all IPv6 end-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 will be used (presumably, an IPv6-capable version of BGP4).

Automatic tunnels

Automatic tunnels provide a simple mechanism to establish IPv6 connectivity between isolated dual-stack hosts and/or routers. In automatic tunneling, the IPv4 tunnel endpoint is determined from the IPv4 address embedded in the IPv4-compatible destination address of the IPv6 packet being tunneled. If the destination IPv6 address is IPv4-compatible, then the packet is sent via automatic tunneling. If the destination is IPv6-native, the packet cannot be sent via automatic tunneling. An IPv6-compatible address is identified by a ::/96 prefix and holds an IPv4 address in the low-order 32 bits. IPv4-compatible addresses are assigned exclusively to nodes that support automatic tunneling. It is globally unique as long as the IPv4 address is not from the private IPv4 address space.

When an IPv6 packet is sent over an automatic tunnel, the IPv6 packet is encapsulated within an IPv4 header as described in “Tunneling Overview” on page 107. The source IPv4 address is an address of the interface the packet is sent over, and the destination IPv4 address is the low-order 32 bits of the IPv6 destination address. The packet is always sent in this form, even if the tunnel endpoint is on an attached link.

Automatic tunneling may be either host-host or router-host. A source host will send an IPv6 packet to an IPv6 router if possible, but that router may not be able to do the same and may have to perform automatic tunneling to the destination host itself. Because of the preference for the use of IPv6 routers rather than automatic tunneling, the tunnel will always be as short as possible. However, the tunnel will always extend all the way to the destination host. In order to use a tunnel that does not extend all the way to the recipient, another tunneling protocol must be used.

There are several issues to be aware of when using automatic tunnels. First, it does not solve the address exhaustion problem of IPv4, as it requires each tunnel endpoint to have an IPv4 address from which the IPv6 compatible address is created. Second, the use of IPv4 compatible addresses cause IPv4 addresses to be included in the IPv6 routing table, which in turn can cause a dramatic increase in the size of the IPv6 routing table. Because of these concerns, it is generally recommended that other tunneling protocols, such as 6to4 tunnels, be used in preference to automatic tunnels.

6to4 tunnels

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, i.e., it is 2002::/16 when expressed as an IPv6 address prefix.

The format for a 6to4 address is as follows:

16 bits	32 bits	16 bits	64 bits
0x0002	V4ADDR	Subnet ID	Interface ID

Figure 21. 6to4 address format

Thus, this prefix has exactly 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 exactly 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 will discover 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 via 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 will return 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 Overview" on page 107. 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

6over4

The Interface Identifier of an IPv4 interface using 6over4 is the 32-bit IPv4 address of that interface, padded to the left with 0's, 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 22. 6over4 address format

Site-local and global unicast addresses are generated by prepending a 64-bit prefix to the 6over4 Interface Identifier. These prefixes may be learned in any of the normal manners, for example, as part of stateless address autoconfiguration or via 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 virutal 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 may be manually configured or dynamically learned via Stateless Address Autoconfiguration.

Since 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 may 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. Such a 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 just means that the TCP/IP stack has built-in support for both IPv4 and IPv6. In this document the term dual mode stack or IP node is a TCP/IP stack that supports both IPv4 and IPv6 protocols.

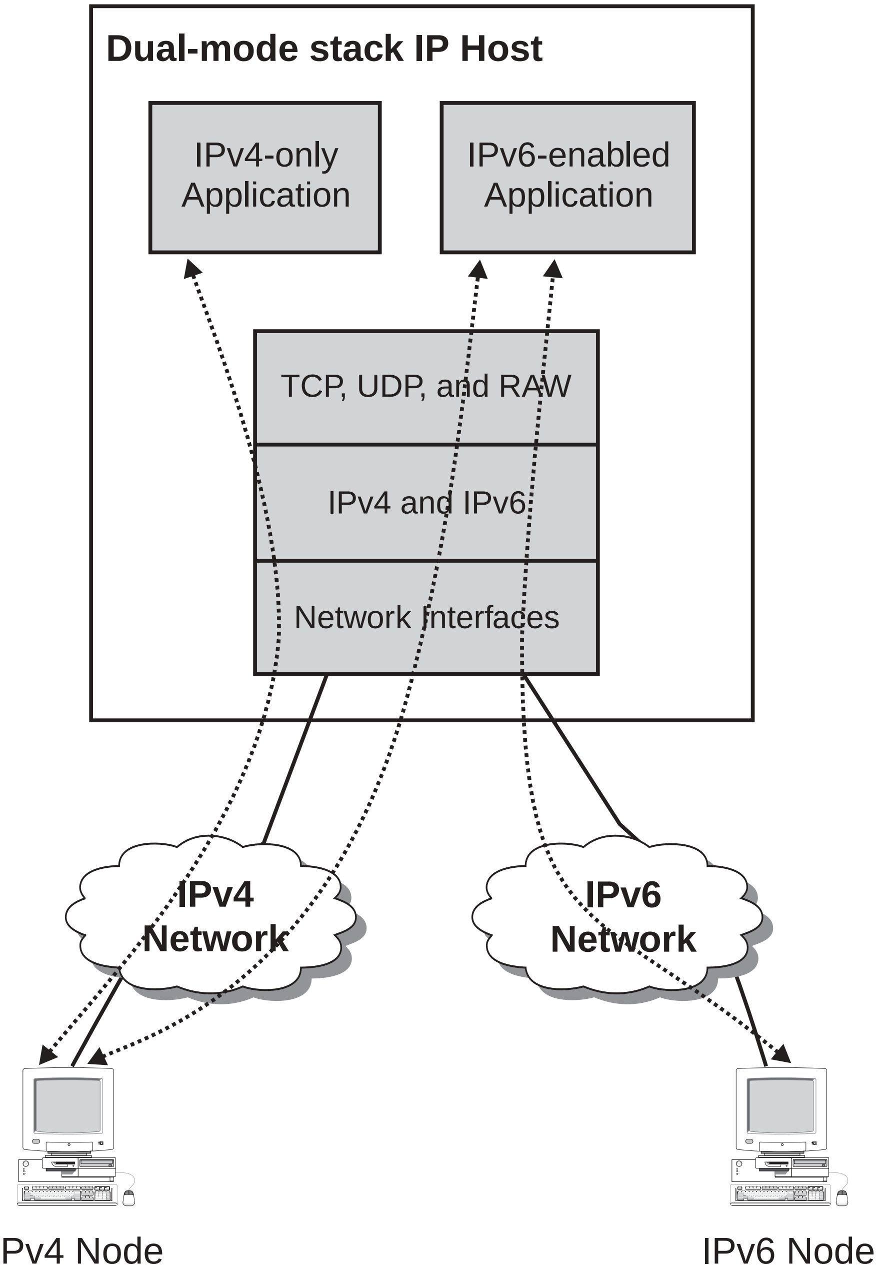

Figure 23. Dual-mode stack IP host

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 are only able to communicate using IPv4 addresses. IPv6-enabled applications that use AF_INET6 sockets may communicate using both IPv4 and IPv6 addresses (on a dual mode host). AF_INET and AF_INET6 applications may thus 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.

	Appl. on a dual mode host	
	IPv4-only	IPv6-enabled
IPv4-only partner	✓	✓
IPv6-only partner		✓

Figure 24. Application communication on a dual-mode host

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:
- No access to the source code (vendor product, or source no longer available).
- The sockets API implementation does not yet (or will never) 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 being 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

The key to successful adoption and deployment of IPv6 is how to 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 approaches possible, with some of the more likely being described below. The transition strategy a company chooses to take will vary based on the particular needs of that company.

This section provides an introduction to a few transition mechanisms which may be used when migrating to an IPv6 network.

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 which are only interconnected using the IPv4 backbone. Tunneling of IPv6 packets over the IPv4 network may 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 may be used.

Application Layer Gateways (ALGs) allow an IPv6-only applications to communicate to an IPv4-only peer. Using an ALG, the client connects to the ALG using its native protocol (IPv4 or IPv6) and the ALG connects to the server using the other protocol (IPv6 or IPv4, respectively).

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 on which the client applications reside) need to be socksified 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 vice versa. This translation typically involves some form of network address translation (NAT) in addition to the protocol translation (PT) function. It may execute in a specialized node which resides between an IPv4 network and an IPv6 network, or it may execute in the host which 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, protocol translation has several issues which make it a less-than ideal solution:

- Protocol translation is not foolproof. It is difficult to determe exactly how long to keep the mappings between the real IPv6 address and the locally mapped IPv4 address around. Eventually, an address is going to be reused before all servers have stopped accessing the address.
- Some applications may use the remote IP address as a means of performing a security check. Unless AH or an IPSec tunnel is used then this really is not foolproof, but it is still done. If the IPv4 address is a locally mapped address then any checks such as this are broken.
- Displays and traces of the remote IP address will be meaningless. Today, many applications put out 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 which performed the NAT function. The DNS resolver and/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 vice versa. As such, it may not be possible to translate the content of an IPv4 packet into an equivalent IPv6 packet, or an IPv4 packet into an equivalent IPv6 packet.

Stateless IP/ICMP Translation Algorithm (SIIT)

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. 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.

For more detailed information on SIIT, refer to *Stateless IP/ICMP Translator (SITT)*, RFC 2765.

Network Address Translation - Protocol Translation (NAT-PT)

Protocol translation may occur at a specialized node which resides between IPv4 and IPv6 networks. This node is typically referred to as a NAT-PT device as it must both 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, it would seem as though NAT-PT is a superior solution. However, due to the limitations of NAT-PT and familiarity with SOCKS, it is actually more likely that SOCKS64 will be used for allowing IPv4-only applications to communicate with IPv6-only peers.

For more detailed information on NAT-PT, refer to *Network Address Translation - Protocol Translation*, RFC 2766.

Chapter 11. IPv6 support tables

IPv6 standards supported on z/OS V1R4

Note that RFCs are not implemented in their entirety.

Table 27. Supported IPv6 standards on z/OS CS V1R4

Standard	RFC or Internet Draft
Internet Protocol, Version 6 (IPv6) Specification	2460
IP Version 6 Addressing Architecture	2373
An IPv6 Aggregatable Global Unicast Address Format	2374
Stateless address autoconfiguration	2462
Path MTU discovery	1981
Internet Control Message Protocol (ICMPv6) for IPv6	2463
Neighbor discovery	2461
Default address selection	draft-ieft-ipngwg-default-addr-select-06
Multicast listener discovery - host	2710
IPv6 Router Alert Option	2711
Basic Socket Interface Extensions for IPv6	draft-ieft-ipngwg-rfc2553bis-05
Advanced Sockets API for IPv6	2292
DNS Extensions to support IP version 6	1886
DNS Extensions to Support IPv6 Address Aggregation and Renumbering	2874
FTP Extensions for IPv6 and NATs	2428
Transmission of IPv6 Packets over Ethernet Networks	2464

z/OS specific features

The following tables summarize z/OS TCP/IP features and the level of support provided in an IPv6 network. It is anticipated that many more of these features will be enabled for IPv6 support in subsequent releases of the z/OS Communications Server.

Table 28. 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 via an INTERFACE statement of type IPAQENET6
CTC	Y	N	
LCS	Y	N	
CLAW	Y	N	
CDLC (3745/3746)	Y	N	
SNALINK LU0 and LU6.2	Y	N	

Table 28. Link-layer device support (continued)

Link-layer device support	IPv4 support	IPv6 support	Comments
X.25 NPSI	Y	N	
NSC HyperChannel	Y	N	
MPC Point-Point	Y	N	
ATM	Y	N	
HiperSockets	Y	N	
XCF	Y	N	

Table 29. Virtual IP Addressing support

Virtual IP Addressing support	IPv4 support	IPv6 support	Comments
Virtual Device/Interface Configuration for static VIPA	Y	Y	

Table 30. Sysplex support

Sysplex support	IPv4 support	IPv6 support	Comments
Sysplex Distributor	Y	N	
Dynamic VIPA	Y	N	
Dynamic XCF	Y	N	

Table 31. IP routing functions

IP routing functions	IPv4 support	IPv6 support	Comments
Dynamic Routing - OSPF	Y	N	
Dynamic Routing - RIP	Y	N	
Static Route Configuration via BEGINROUTE statement	Y	Y	
Static Route Configuration via GATEWAY statement	Y	N	
Multipath Routing Groups	Y	Y	

Table 32. Misc. IP/IF-layer functions

Misc. IP/IF-layer functions	IPv4 support	IPv6 support	Comments
Path MTU Discovery	Y	Y	
Configurable Device or Interface Recovery Interval	Y	Y	
Link-Layer Address Resolution	Y	Y	
ARP/Neighbor Cache PURGE Capability	Y	Y	
Datagram Forwarding Enable/Disable	Y	Y	

Table 33. Transport-layer functions

Transport-layer functions	IPv4 support	IPv6 support	Comments
Fast Response Cache Accelerator	Y	N	
Enterprise Extender	Y	N	
Server-BIND Control	Y	Y	
UDP Checksum Disablement Option	Y	N	

Table 34. Network management and accounting functions

Network management and accounting Functions	IPv4 support	IPv6 support	Comments
SNMP	Y	N	
Policy-Based Networking	Y	N	
SMF	Y	Y	

Table 35. Security functions

Security functions	IPv4 support	IPv6 support	Comments
IPSec	Y	N	
IP Filtering	Y	N	
Network Access Control	Y	N	
Stack and Port Access Control	Y	Y	
Intrusion Detection Services	Y	N	

Applications not enabled for IPv6

Table 36. V1R4 applications not enabled for IPv6

Server Applications	IPv4 support	IPv6 support
SMTP/NJE server	Y	N
TN3270(E) server	Y	N
Rlogind server	Y	N
MVS rexecd/rshd server	Y	N
MVS Miscellaneous server	Y	N
Sendmail server/client	Y	N
Popper	Y	N
CICS sockets listener	Y	N
ISAKMP server	Y	N
NDB server	Y	N
SNMPD server	Y	N
MVS LPD server	Y	N
Syslogd server	Y	N
TFTPD server	Y	N

Table 36. V1R4 applications not enabled for IPv6 (continued)

Server Applications	**IPv4 support**	**IPv6 support**
DHCPD 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
Orouted server	Y	N
OMPROUTE server	Y	N
NCPROUTE	Y	N
NPF	Y	N
Pagent	Y	N
RSVP daemon	Y	N
TRMD daemon	Y	N
UNIX named (BIND 4.9.3 based)	Y	N
Client Applications		
TSO telnet client	Y	N
TSO rexec client	Y	N
TSO rsh client	Y	N
TSO lpr client	Y	N
Command-type Applications		
TSO nslookup	Y	N
UNIX nslookup (BIND 4.9.3 based)	Y	N
UNIX nsupdate (BIND 4.9.3 based)	Y	N
TSO lprm	Y	N
TSO dig	Y	N
UNIX dig	Y	N
TSO rpcinfo	Y	N
UNIX rpcinfo	Y	N

Appendix A.

Related protocol specifications (RFCs)

This appendix lists the related protocol specifications 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.

These documents can be obtained from:

Government Systems, Inc.
Attn: Network Information Center
14200 Park Meadow Drive
Suite 200
Chantilly, VA 22021

You can see Internet drafts at http://www.ietf.org/ID.html. See "Internet Drafts" on page 128 for draft RFCs implemented in z/OS V1R4 Communications Server.

You can also request RFCs through electronic mail, from the automated NIC mail server, by sending a message to `service@nic.ddn.mil` with a subject line of `RFC` *nnnn* for text versions or a subject line of `RFC` *nnnn*`.PS` for PostScript versions. To request a copy of the RFC index, send a message with a subject line of `RFC INDEX`.

For more information, contact `nic@nic.ddn.mil`.

Many RFCs are available online. Hard copies of all RFCs are available from the NIC, either individually or by subscription. Online copies are available using FTP from the NIC at the following Web address: http://www.rfc-editor.org/rfc.html.

Use FTP to download the files, using the following format:

```
RFC:RFC-INDEX.TXT
RFC:RFCnnnn.TXT
RFC:RFCnnnn.PS
```

where:

nnnn Is the RFC number.
TXT Is the text format.
PS Is the PostScript format.

Many features of TCP/IP Services are based on the following RFCs:

RFC	Title and Author
768	*User Datagram Protocol* J.B. Postel
791	*Internet Protocol* J.B. Postel
792	*Internet Control Message Protocol* J.B. Postel
793	*Transmission Control Protocol* J.B. Postel

821 *Simple Mail Transfer Protocol* J.B. Postel

822 *Standard for the Format of ARPA Internet Text Messages* D. Crocker

823 *DARPA Internet Gateway* R.M. Hinden, A. Sheltzer

826 *Ethernet Address Resolution Protocol or Converting Network Protocol Addresses to 48.Bit Ethernet Address for Transmission on Ethernet Hardware* D.C. Plummer

854 *Telnet Protocol Specification* J.B. Postel, J.K. Reynolds

855 *Telnet Option Specification* J.B. Postel, J.K. Reynolds

856 *Telnet Binary Transmission* J.B. Postel, J.K. Reynolds

857 *Telnet Echo Option* J.B. Postel, J.K. Reynolds

858 *Telnet Suppress Go Ahead Option* J.B. Postel, J.K. Reynolds

859 *Telnet Status Option* J.B. Postel, J.K. Reynolds

860 *Telnet Timing Mark Option* J.B. Postel, J.K. Reynolds

861 *Telnet Extended Options—List Option* J.B. Postel, J.K. Reynolds

862 *Echo Protocol* J.B. Postel

863 *Discard Protocol* J.B. Postel

864 *Character Generator Protocol* J.B. Postel

877 *Standard for the Transmission of IP Datagrams over Public Data Networks* J.T. Korb

885 *Telnet End of Record Option* J.B. Postel

896 *Congestion Control in IP/TCP Internetworks* J. Nagle

903 *Reverse Address Resolution Protocol* R. Finlayson, T. Mann, J.C. Mogul, M. Theimer

904 *Exterior Gateway Protocol Formal Specification* D.L. Mills

919 *Broadcasting Internet Datagrams* J.C. Mogul

922 *Broadcasting Internet Datagrams in the Presence of Subnets* J.C. Mogul

950 *Internet Standard Subnetting Procedure* J.C. Mogul, J.B. Postel

952 *DoD Internet Host Table Specification* K. Harrenstien, M.K. Stahl, E.J. Feinler

959 *File Transfer Protocol* J.B. Postel, J.K. Reynolds

974 *Mail Routing and the Domain Name System* C. Partridge

1006 *ISO Transport Service on top of the TCP Version 3* M.T.Rose, D.E. Cass

1009 *Requirements for Internet Gateways* R.T. Braden, J.B. Postel

1011 *Official Internet Protocols* J. Reynolds, J. Postel

1013 *X Window System Protocol, Version 11: Alpha Update* R.W. Scheifler

1014 *XDR: External Data Representation Standard* Sun Microsystems Incorporated

1027 *Using ARP to Implement Transparent Subnet Gateways* S. Carl-Mitchell, J.S. Quarterman

1032 *Domain Administrators Guide* M.K. Stahl

1033 *Domain Administrators Operations Guide* M. Lottor

1034 *Domain Names—Concepts and Facilities* P.V. Mockapetris

1035 *Domain Names—Implementation and Specification* P.V. Mockapetris

1042 *Standard for the Transmission of IP Datagrams over IEEE 802 Networks* J.B. Postel, J.K. Reynolds

1044 *Internet Protocol on Network System's HYPERchannel: Protocol Specification* K. Hardwick, J. Lekashman

1055 *Nonstandard for Transmission of IP Datagrams over Serial Lines: SLIP* J.L. Romkey

1057 *RPC: Remote Procedure Call Protocol Version 2 Specification* Sun Microsystems Incorporated

1058 *Routing Information Protocol* C.L. Hedrick

1060 *Assigned Numbers* J. Reynolds, J. Postel

1073 *Telnet Window Size Option* D. Waitzman

1079 *Telnet Terminal Speed Option* C.L. Hedrick

1091 *Telnet Terminal-Type Option* J. VanBokkelen

1094 *NFS: Network File System Protocol Specification* Sun Microsystems Incorporated

1096 *Telnet X Display Location Option* G. Marcy

1101 *DNS encoding of network names and other types* P.V. Mockapetris

1112 *Host Extensions for IP Multicasting* S. Deering

1118 *Hitchhikers Guide to the Internet* E. Krol

1122 *Requirements for Internet Hosts—Communication Layers* R.T. Braden

1123 *Requirements for Internet Hosts—Application and Support* R.T. Braden

1155 *Structure and Identification of Management Information for TCP/IP-Based Internets* M.T. Rose, K. McCloghrie

1156 *Management Information Base for Network Management of TCP/IP-Based Internets* K. McCloghrie, M.T. Rose

1157 *Simple Network Management Protocol (SNMP)* J.D. Case, M. Fedor, M.L. Schoffstall, C. Davin

1158 *Management Information Base for Network Management of TCP/IP-based internets: MIB-II* M.T. Rose

1179 *Line Printer Daemon Protocol* The Wollongong Group, L. McLaughlin III

1180 *TCP/IP Tutorial* T.J. Socolofsky, C.J. Kale

1183 *New DNS RR Definitions* C.F. Everhart, L.A. Mamakos, R. Ullmann, P.V. Mockapetris, (Updates RFC 1034, RFC 1035)

1184 *Telnet Linemode Option* D. Borman

1187 *Bulk Table Retrieval with the SNMP* M.T. Rose, K. McCloghrie, J.R. Davin

1188 *Proposed Standard for the Transmission of IP Datagrams over FDDI Networks* D. Katz

1191 *Path MTU Discovery* J. Mogul, S. Deering

1198 *FYI on the X Window System* R.W. Scheifler

1207 *FYI on Questions and Answers: Answers to Commonly Asked “Experienced Internet User” Questions* G.S. Malkin, A.N. Marine, J.K. Reynolds

1208 *Glossary of Networking Terms* O.J. Jacobsen, D.C. Lynch

1213 *Management Information Base for Network Management of TCP/IP-Based Internets: MIB-II* K. McCloghrie, M.T. Rose

1215 *Convention for Defining Traps for Use with the SNMP* M.T. Rose

1228 *SNMP-DPI Simple Network Management Protocol Distributed Program Interface* G.C. Carpenter, B. Wijnen

1229 *Extensions to the Generic-Interface MIB* K. McCloghrie

1230 *IEEE 802.4 Token Bus MIB* K. McCloghrie, R. Fox

1231 *IEEE 802.5 Token Ring MIB* K. McCloghrie, R. Fox, E. Decker

1236 *IP to X.121 Address Mapping for DDN* L. Morales, P. Hasse

1267 *A Border Gateway Protocol 3 (BGP-3)* K. Lougheed, Y. Rekhter

1268 *Application of the Border Gateway Protocol in the Internet* Y. Rekhter, P. Gross

1269 *Definitions of Managed Objects for the Border Gateway Protocol (Version 3)* S. Willis, J. Burruss

1270 *SNMP Communications Services* F. Kastenholz, ed.

1321 *The MD5 Message-Digest Algorithm* R. Rivest

1323 *TCP Extensions for High Performance* V. Jacobson, R. Braden, D. Borman

1325 *FYI on Questions and Answers: Answers to Commonly Asked ″New Internet User″ Questions* G.S. Malkin, A.N. Marine

1340 *Assigned Numbers* J.K. Reynolds, J.B. Postel

1348 *DNS NSAP RRs* B. Manning

1349 *Type of Service in the Internet Protocol Suite* P. Almquist

1350 *TFTP Protocol* K.R. Sollins

1351 *SNMP Administrative Model* J. Davin, J. Galvin, K. McCloghrie

1352 *SNMP Security Protocols* J. Galvin, K. McCloghrie, J. Davin

1353 *Definitions of Managed Objects for Administration of SNMP Parties* K. McCloghrie, J. Davin, J. Galvin

1354 *IP Forwarding Table MIB* F. Baker

1356 *Multiprotocol Interconnect on X.25 and ISDN in the Packet Mode* A. Malis, D. Robinson, R. Ullmann

1363 *A Proposed Flow Specification* C. Partridge

1372 *Telnet Remote Flow Control Option* D. Borman, C. L. Hedrick

1374 *IP and ARP on HIPPI* J. Renwick, A. Nicholson

1381 *SNMP MIB Extension for X.25 LAPB* D. Throop, F. Baker

1382 *SNMP MIB Extension for the X.25 Packet Layer* D. Throop

1387 *RIP Version 2 Protocol Analysis* G. Malkin

1388 *RIP Version 2—Carrying Additional Information* G. Malkin

1389 *RIP Version 2 MIB Extension* G. Malkin

1390 *Transmission of IP and ARP over FDDI Networks* D. Katz

1393 *Traceroute Using an IP Option* G. Malkin

1397 *Default Route Advertisement In BGP2 And BGP3 Versions of the Border Gateway Protocol* D. Haskin

1398 *Definitions of Managed Objects for the Ethernet-Like Interface Types* F. Kastenholz

1416 *Telnet Authentication Option* D. Borman, ed.

1464 *Using the Domain Name System to Store Arbitrary String Attributes* R. Rosenbaum

1469 *IP Multicast over Token-Ring Local Area Networks* T. Pusateri

1535 *A Security Problem and Proposed Correction With Widely Deployed DNS Software* E. Gavron

1536 *Common DNS Implementation Errors and Suggested Fixes* A. Kumar, J. Postel, C. Neuman, P. Danzig, S.Miller

1537 *Common DNS Data File Configuration Errors* P. Beertema

1540 *IAB Official Protocol Standards* J.B. Postel

1571 *Telnet Environment Option Interoperability Issues* D. Borman

1572 *Telnet Environment Option* S. Alexander

1577 *Classical IP and ARP over ATM* M. Laubach

1583 *OSPF Version 2* J. Moy

1591 *Domain Name System Structure and Delegation* J. Postel

1592 *Simple Network Management Protocol Distributed Protocol Interface Version 2.0* B. Wijnen, G. Carpenter, K. Curran, A. Sehgal, G. Waters

1594 *FYI on Questions and Answers: Answers to Commonly Asked "New Internet User" Questions* A.N. Marine, J. Reynolds, G.S. Malkin

1695 *Definitions of Managed Objects for ATM Management Version 8.0 Using SMIv2* M. Ahmed, K. Tesink

1706 *DNS NSAP Resource Records* B. Manning, R. Colella

1713 *Tools for DNS debugging* A. Romao

1723 *RIP Version 2—Carrying Additional Information* G. Malkin

1766 *Tags for the Identification of Languages* H. Alvestrand

1794 *DNS Support for Load Balancing* T. Brisco

1832 *XDR: External Data Representation Standard* R. Srinivasan

1850 *OSPF Version 2 Management Information Base* F. Baker, R. Coltun

1876 *A Means for Expressing Location Information in the Domain Name System* C. Davis, P. Vixie, T. Goodwin, I. Dickinson

1886 *DNS Extensions to support IP version 6* S. Thomson, C. Huitema

1901 *Introduction to Community-Based SNMPv2* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

1902 *Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

1903 *Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

1904 *Conformance Statements for Version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

1905 *Protocols Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

1906 *Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

1907 *Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

1908 *Coexistence between Version 1 and Version 2 of the Internet-Standard Network Management Framework* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

1912 *Common DNS Operational and Configuration Errors* D. Barr

1918 *Address Allocation for Private Internets* Y. Rekhter, B. Moskowitz, D. Karrenberg, G.J. de Groot, E. Lear

1928 *SOCKS Protocol Version 5* M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, L. Jones

1939 *Post Office Protocol-Version 3* J. Myers, M. Rose

1981 *Path MTU Discovery for IP version 6* J. McCann, S. Deering, J. Mogul

1982 *Serial Number Arithmetic* R. Elz, R. Bush

1995 *Incremental Zone Transfer in DNS* M. Ohta

1996 *A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)* P. Vixie

2010 *Operational Criteria for Root Name Servers* B. Manning, P. Vixie

2011 *SNMPv2 Management Information Base for the Internet Protocol Using SMIv2* K. McCloghrie

2012 *SNMPv2 Management Information Base for the Transmission Control Protocol Using SMIv2* K. McCloghrie

2013 *SNMPv2 Management Information Base for the User Datagram Protocol Using SMIv2* K. McCloghrie

2052 *A DNS RR for specifying the location of services (DNS SRV)* A. Gulbrandsen, P. Vixie

2065 *Domain Name System Security Extensions* D. Eastlake, C. Kaufman

2096 *IP Forwarding Table MIB* F. Baker

2104 *HMAC: Keyed-Hashing for Message Authentication* H. Krawczyk, M. Bellare, R. Canetti

2132 *DHCP Options and BOOTP Vendor Extensions* S. Alexander, R. Droms

2133 *Basic Socket Interface Extensions for IPv6* R. Gilligan, S. Thomson, J. Bound, W. Stevens

2137 *Secure Domain Name System Dynamic Update* D. Eastlake

2163 *Using the Internet DNS to Distribute MIXER Conformant Global Address Mapping (MCGAM)* C. Allocchio

2168 *Resolution of Uniform Resource Identifiers using the Domain Name System* R. Daniel, M. Mealling

2178 *OSPF Version 2* J. Moy

2181 *Clarifications to the DNS Specification* R. Elz, R. Bush

2205 *Resource ReSerVation Protocol (RSVP) Version 1* R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin

2210 *The Use of RSVP with IETF Integrated Services* J. Wroclawski

2211 *Specification of the Controlled-Load Network Element Service* J. Wroclawski

2212 *Specification of Guaranteed Quality of Service* S. Shenker, C. Partridge, R. Guerin

2215 *General Characterization Parameters for Integrated Service Network Elements* S. Shenker, J. Wroclawski

2219 *Use of DNS Aliases for Network Services* M. Hamilton, R. Wright

2228 *FTP Security Extensions* M. Horowitz, S. Lunt

2230 *Key Exchange Delegation Record for the DNS* R. Atkinson

2233 *The Interfaces Group MIB Using SMIv2* K. McCloghrie, F. Kastenholz

2240 *A Legal Basis for Domain Name Allocation* O. Vaughn

2246 *The TLS Protocol Version 1.0* T. Dierks, C. Allen

2308 *Negative Caching of DNS Queries (DNS NCACHE)* M. Andrews

2317 *Classless IN-ADDR.ARPA delegation* H. Eidnes, G. de Groot, P. Vixie

2320 *Definitions of Managed Objects for Classical IP and ARP over ATM Using SMIv2* M. Greene, J. Luciani, K. White, T. Kuo

2328 *OSPF Version 2* J. Moy

2345 *Domain Names and Company Name Retrieval* J. Klensin, T. Wolf, G. Oglesby

2352 *A Convention for Using Legal Names as Domain Names* O. Vaughn

2355 *TN3270 Enhancements* B. Kelly

2373 *IP Version 6 Addressing Architecture* R. Hinden, M. O'Dell, S. Deering

2374 *An IPv6 Aggregatable Global Unicast Address Format* R. Hinden, M. O'Dell, S. Deering

2375 *IPv6 Multicast Address Assignments* R. Hinden, S. Deering

2389 *Feature negotiation mechanism for the File Transfer Protocol* P. Hethmon, R. Elz

2428 *FTP Extensions for IPv6 and NATs* M. Allman, S. Ostermann, C. Metz

2460 *Internet Protocol, Version 6 (IPv6) S pecification* S. Deering, R. Hinden

2461 *Neighbor Discovery for IP Version 6 (IPv6)* T. Narten, E. Nordmark, W. Simpson

2462 *IPv6 Stateless Address Autoconfiguration* S. Thomson, T. Narten

2464 *Transmission of IPv6 Packets over Ethernet Networks* M. Crawford

2474 *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers* K. Nichols, S. Blake, F. Baker, D. Black

2535 *Domain Name System Security Extensions* D. Eastlake

2539 *Storage of Diffie-Hellman Keys in the Domain Name System (DNS)* D. Eastlake

2553 *Basic Socket Interface Extensions for IPv6* R. Gilligan, S. Thomson, J. Bound, W. Stevens

2571 *An Architecture for Describing SNMP Management Frameworks* D. Harrington, R. Presuhn, B. Wijnen

2572 *Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)* J. Case, D. Harrington, R. Presuhn, B. Wijnen

2573 *SNMP Applications* D. Levi, P. Meyer, B. Stewart

2574 *User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)* U. Blumenthal, B. Wijnen

2575 *View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)* B. Wijnen, R. Presuhn, K. McCloghrie

2578 *Structure of Management Information Version 2 (SMIv2)* K. McCloghrie, D. Perkins, J. Schoenwaelder

2640 *Internationalization of the File Transfer Protocol* B. Curtin

2665 *Definitions of Managed Objects for the Ethernet-like Interface Types* J. Flick, J. Johnson

2672 *Non-Terminal DNS Name Redirection* M. Crawford

2710 *Multicast Listener Discovery (MLD) for IPv6*S. Deering, W. Fenner, B. Haberman

2711 *IPv6 Router Alert Option*C. Partridge, A. Jackson

2758 *Definitions of Managed Objects for Service Level Agreements Performance Monitoring* K. White

2845 *Secret Key Transaction Authentication for DNS (TSIG)* P. Vixie, O. Gudmundsson, D. Eastlake, B. Wellington

2874 *DNS Extensions to Support IPv6 Address Aggregation and Renumbering* M. Crawford, C. Huitema

2941 *Telnet Authentication Option* T. Ts'o, ed., J. Altman

2942 *Telnet Authentication: Kerberos Version 5* T. Ts'o

2946 *Telnet Data Encryption Option* T. Ts'o

2952 *Telnet Encryption: DES 64 bit Cipher Feedback* T. Ts'o

2953 *Telnet Encryption: DES 64 bit Output Feedback* T. Ts'o, ed.

3060 *Policy Core Information Model—Version 1 Specification* B. Moore, E. Ellesson, J. Strassner, A. Westerinen

Internet Drafts

Several areas of IPv6 implementation include elements of the following Internet drafts and are subject to change during the RFC review process.

Advanced Sockets API for IPv6
W. Richard Stevens, Matt Thomas, Erik Nordmark, Tatuya Jinmei

Basic Socket Interface Extensions for IPv6
R.E. Gilligan, S. Thomson, J. Bound, J. McCann, W. R. Stevens

Default Address Selection for IPv6
R. Draves

Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification
A. Conta, S. Deering

IP Version 6 Addressing Architecture
R. Hinden, S. Deering

Appendix B.

Information APARs

This appendix lists information APARs for IP and SNA documents.

Notes:

1. Information APARs contain updates to previous editions of the manuals listed below. Documents updated for V1R4 are complete except for the updates contained in the information APARs that may be issued after V1R4 documents went to press.
2. Information APARs are predefined for z/OS V1R4 Communications Server and may not contain updates.
3. Information APARs for OS/390 documents are in the document called *OS/390 DOC APAR and PTF ++HOLD Documentation*, which can be found at http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/IDDOCMST/CCONTENTS.
4. Information APARs for z/OS documents are in the document called *z/OS and z/OS.e DOC APAR and PTF ++HOLD Documentation*, which can be found at http://publibz.boulder.ibm.com:80/cgi-bin/bookmgr_OS390/BOOKS/ZIDOCMST/CCONTENTS.

Information APARs for IP documents

Table 37 lists information APARs for IP documents.

Table 37. IP information APARs

Title	z/OS CS V1R4	z/OS CS V1R2	CS for OS/390 2.10 and z/OS CS V1R1	CS for OS/390 2.8
IP API Guide	ii13255	ii12861	ii12371	ii11635
IP CICS Sockets Guide	ii13257	ii12862		ii11626
IP Configuration				ii11620 ii12068 ii12353 ii12649 ii13018
IP Configuration Guide	ii13244	ii12498 ii13087	ii12362 ii12493 ii13006	
IP Configuration Reference	ii13245	ii12499	ii12363 ii12494 ii12712	
IP Diagnosis	ii13249	ii12503	ii12366 ii12495	ii11628
IP Messages Volume 1	ii13250	ii12857 ii13229	ii12367	ii11630 13230
IP Messages Volume 2	ii13251	ii12858	ii12368	ii11631
IP Messages Volume 3	ii13252	ii12859	ii12369 12990	ii11632 ii12883
IP Messages Volume 4	ii13253	ii12860		

Table 37. IP information APARs (continued)

Title	z/OS CS V1R4	z/OS CS V1R2	CS for OS/390 2.10 and z/OS CS V1R1	CS for OS/390 2.8
IP Migration	ii13242	ii12497	ii12361	ii11618
IP Network and Application Design Guide	ii13243			
IP Network Print Facility		ii12864		ii11627
IP Programmer's Reference	ii13256	ii12505		ii11634
IP and SNA Codes	ii13254	ii12504	ii12370	ii11917
IP User's Guide			ii12365 ii13060	ii11625
IP User's Guide and Commands	ii13247	ii12501	ii12365 ii13060	ii11625
IP System Admin Guide	ii13248	ii12502		
Quick Reference	ii13246	ii12500	ii12364	

Information APARs for SNA documents

Table 38 lists information APARs for SNA documents.

Table 38. SNA information APARs

Title	z/OS CS V1R4	z/OS CS V1R2	CS for OS/390 2.10 and z/OS CS V1R1	CS for OS/390 2.8
Anynet SNA over TCP/IP				ii11922
Anynet Sockets over SNA				ii11921
CSM Guide				
IP and SNA Codes	ii13254	ii12504	ii12370	ii11917
SNA Customization	ii13240	ii12872	ii12388	ii11923
SNA Diagnosis	ii13236	ii12490 ii13034`	ii12389	ii11915
SNA Messages	ii13238	ii12491	ii12382 ii12383	ii11916
SNA Network Implementation Guide	ii13234	ii12487	ii12381	ii11911
SNA Operation	ii13237	ii12489	ii12384	ii11914
SNA Migration	ii13233	ii12486	ii12386	ii11910
SNA Programming	ii13241	ii13033	ii12385	ii11920
Quick Reference	ii13246	ii12500	ii12364	ii11913
SNA Resource Definition Reference	ii13235	ii12488	ii12380 ii12567	ii11912 ii12568
SNA Resource Definition Samples				
SNA Data Areas	ii13239	ii12492	ii12387	ii11617

Other information APARs

Table 39 on page 133 lists information APARs not related to documents.

Table 39. Non-document information APARs

Content	Number
OMPROUTE	ii12026
iQDIO	ii11220
index of recomended maintenace for VTAM®	ii11220
CSM for VTAM	ii12657
CSM for TCP/IP	ii12658
AHHC, MPC, and CTC	ii01501
DLUR/DLUS for z/OS V1R2	ii12986
Enterprise Extender	ii12223
Generic resources	ii10986
HPR	ii10953
MNPS	ii10370
Performance	ii11710 ii11711 ii11712

Appendix C.

Accessibility

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 it to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to *z/OS TSO/E Primer*, *z/OS TSO/E User’s Guide*, and *z/OS ISPF User’s Guide Volume 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

IBM may not offer all of the products, services, or features discussed in this document. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION ″AS IS″ WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Site Counsel
IBM Corporation
P.O.Box 12195
3039 Cornwallis Road
Research Triangle Park, North Carolina 27709-2195
U.S.A

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

This product includes cryptographic software written by Eric Young.

If you are viewing this information softcopy, photographs and color illustrations may not appear.

You can obtain softcopy from the z/OS Collection (SK3T-4269), which contains BookManager and PDF formats of unlicensed books and the z/OS Licensed Product Library (LK3T-4307), which contains BookManager and PDF formats of licensed books.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

ACF/VTAM
Advanced Peer-to-Peer Networking
AFP
AD/Cycle
AIX
AIX/ESA
AnyNet
APL2
AS/400
AT
BookManager
BookMaster
CBPDO
C/370
CICS
CICS/ESA
C/MVS
Common User Access
C Set ++
CT
CUA
DATABASE 2
DatagLANce
DB2
DFSMS
DFSMSdfp
DFSMShsm
DFSMS/MVS
DPI
Domino
DRDA
eNetwork
Enterprise Systems Architecture/370
ESA/390
ESCON
eServer
ES/3090
ES/9000
ES/9370
EtherStreamer
Extended Services
FAA
Micro Channel
MVS
MVS/DFP
MVS/ESA
MVS/SP
MVS/XA
MQ
Natural
NetView
Network Station
Nways
Notes
NTune
NTuneNCP
OfficeVision/MVS
OfficeVision/VM
Open Class
OpenEdition
OS/2
OS/390
OS/400
Parallel Sysplex
Personal System/2
PR/SM
PROFS
PS/2
RACF
Resource Link
Resource Measurement Facility
RETAIN
RFM
RISC System/6000
RMF
RS/6000
S/370
S/390
SAA
SecureWay
Slate
SP
SP2
SQL/DS
System/360

- FFST
- FFST/2
- FFST/MVS
- First Failure Support Technology
- GDDM
- Hardware Configuration Definition
- IBM
- IBMLink
- IBMLINK
- IMS
- IMS/ESA
- InfoPrint
- Language Environment
- LANStreamer
- Library Reader
- LPDA
- MCS
- System/370
- System/390
- SystemView
- Tivoli
- TURBOWAYS
- UNIX System Services
- Virtual Machine/Extended Architecture
- VM/ESA
- VM/XA
- VSE/ESA
- VTAM
- WebSphere
- XT
- z/Architecture
- z/OS
- z/OS.e
- zSeries
- 400
- 3090
- 3890

Lotus, Freelance, and Word Pro are trademarks of Lotus Development Corporation in the United States, or other countries, or both.

Tivoli and NetView are trademarks of Tivoli Systems Inc. in the United States, or other countries, or both.

DB2 and NetView are registered trademarks of International Business Machines Corporation or Tivoli Systems Inc. in the U.S., other countries, or both.

The following terms are trademarks of other companies:

ATM is a trademark of Adobe Systems, Incorporated.

BSC is a trademark of BusiSoft Corporation.

CSA is a trademark of Canadian Standards Association.

DCE is a trademark of The Open Software Foundation.

HYPERchannel is a trademark of Network Systems Corporation.

UNIX is a registered trademark in the United States, other countries, or both and is licensed exclusively through X/Open Company Limited.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

ActionMedia, LANDesk, MMX, Pentium, and ProShare are trademarks of Intel Corporation in the United States, other countries, or both. For a complete list of Intel trademarks, see http://www.intel.com/sites/corporate/tradmarx.htm .

Other company, product, and service names may be trademarks or service marks of others.

Index

Communicating Your Comments to IBM

If you especially like or dislike anything about this document, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

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.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this document.
- If you prefer to send comments by FAX, use this number: 1-800-254-0206
- If you prefer to send comments electronically, use this network ID: usib2hpd@vnet.ibm.com

Make sure to include the following in your note:

- Title and publication number of this document
- Page number or topic to which your comment applies.

Readers' Comments — We'd Like to Hear from You

z/OS Communications Server
IPv6 Network and Application Design Guide
Version 1 Release 4

Publication No. SC31-8885-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

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.

Name

Address

Company or Organization

Phone No.

Readers' Comments — We'd Like to Hear from You
SC31-8885-00

Cut or Fold Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Software Reengineering
Department G71A/Bldg 503
Research Triangle Park, NC
U.S.A.
27709-9990

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold Along Line

Program Number: 5694-A01 and 5655-G52

Printed in U.S.A.

SC31-8885-00

Spine information:

IBM

z/OS Communications Server

z/OS V1R4.0 CS: IPv6 Network and Appl Design Guide

Version 1
Release 4