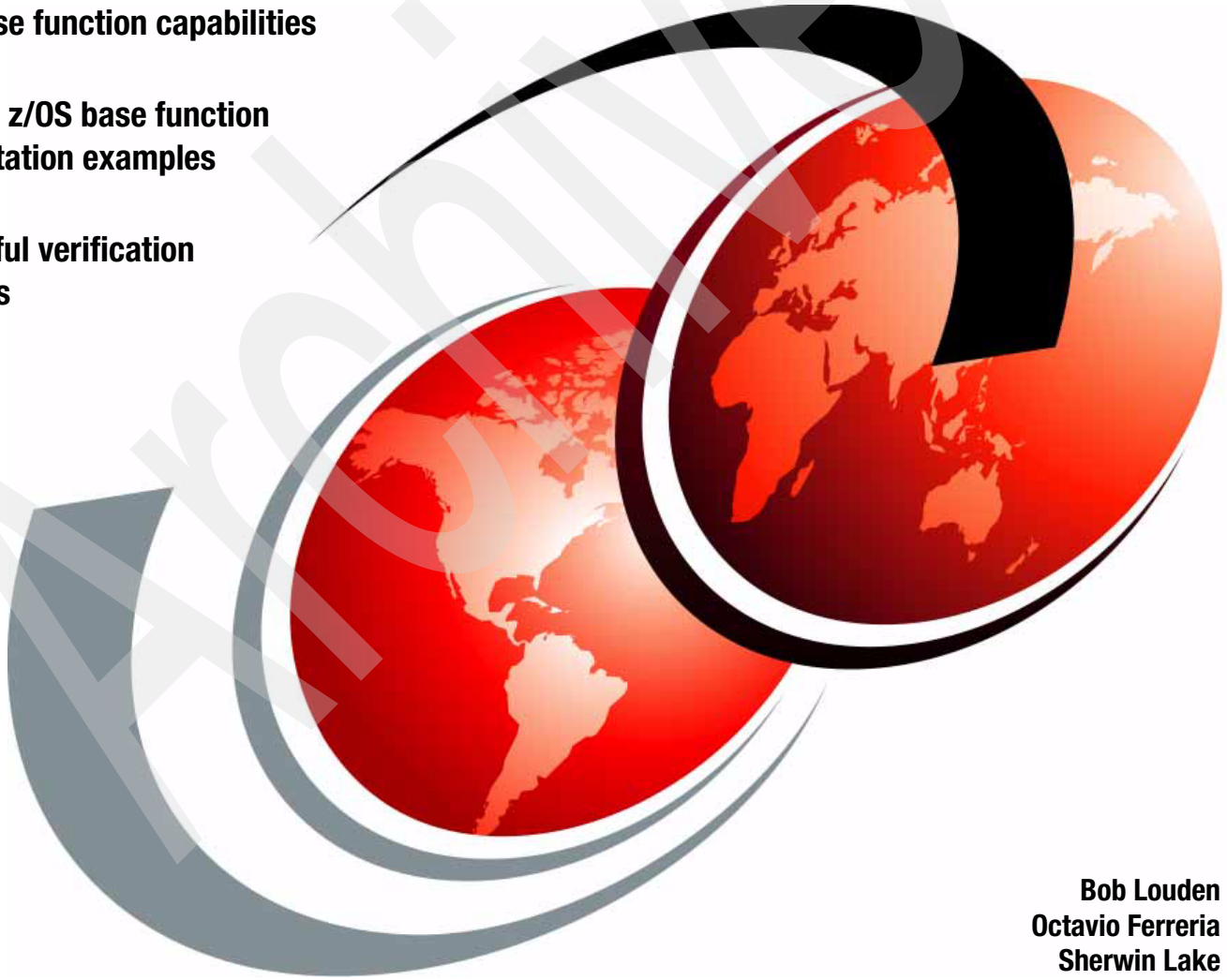
IBM

Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 1

Base Functions, Connectivity, and Routing

- Understand important CS for z/OS TCP/IP base function capabilities
- See CS for z/OS base function implementation examples
- Learn useful verification techniques



Bob Louden
Octavio Ferreria
Sherwin Lake

Redbooks

ibm.com/redbooks

International Technical Support Organization

Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 1 - Base Functions, Connectivity, and Routing

March 2006

SG24-7169-00

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (March 2006)

This edition applies to version 1, release 7, of z/OS Communications Server.

Contents

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. 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.

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.

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.

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.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

@server®
@server®
Redbooks (logo) ™
pSeries®
z/OS®
zSeries®
z9™
AIX®
CICS®
DB2®
DFSMS/MVS®
ESCON®
FFST™
FICON®
HiperSockets™
IBM®
IMS™
Language Environment®
Lotus®
MVS™
NetView®
OS/390®
Parallel Sysplex®
Redbooks™
RACF®
S/390®
System z9™
System/360™
S/390®
VTAM®
WebSphere®

The following terms are trademarks of other companies:

SNM, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

Preface

For more than 40 years, IBM® mainframes have supported an extraordinary portion of the world's computing work—providing centralized corporate databases and mission-critical enterprise-wide applications. The IBM System z9™, the latest generation of the IBM distinguished family of mainframe systems, has come a long way from its IBM System/360™ heritage. Likewise, its z/OS® operating system is far superior to its predecessors—providing, among many other capabilities, world-class, state-of-the-art, support for the TCP/IP Internet protocol suite.

TCP/IP is a large and evolving collection of communication protocols managed by the Internet Engineering Task Force (IETF), an open, volunteer organization. Because of its openness, the TCP/IP protocol suite has become the foundation for the set of technologies that form the basis of the Internet. The convergence of IBM mainframe capabilities with Internet technology, connectivity, and standards—particularly TCP/IP—is dramatically changing the face of information technology and driving requirements for ever more secure, scalable, and highly available mainframe TCP/IP implementations.

This new and improved *Communications Server (CS) for z/OS TCP/IP Implementation* series provides easy to understand step-by-step how-to guidance on enabling the most commonly used and important functions of CS for z/OS TCP/IP.

In *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 1 - Base Functions, Connectivity, and Routing*, SG24-7169, we begin by providing an introduction to CS for z/OS TCP/IP. We then discuss the System Resolver, showing the implementation of global and local settings for single and multi-stack environments. We then present implementation scenarios for TCP/IP base functions, connectivity, and routing. Finally, we discuss the IP version 6 support available with the z/OS V1R7.0 Communications Server.

For more specific information about CS for z/OS standard applications, high availability, and security, please reference the other volumes in the series. These are:

- *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 2 - Standard Applications*, SG24-7170
- *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 3 - High Availability, Scalability, and Performance*, SG24-7171
- *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 4 - Security*, SG24-7172

Our implementation environment

We wrote the four books in the *Communications Server for z/OS V1R7 Implementation* guides at the same time. Given the complexity of our test environment, we needed to be somewhat creative in organizing the environment so that each team could work with minimal coordination with (and interference from) the other teams.

The environment used for all four books

To enable concurrent work on each of the four books, we set up and shared the test environment illustrated in Figure 1 on page x.

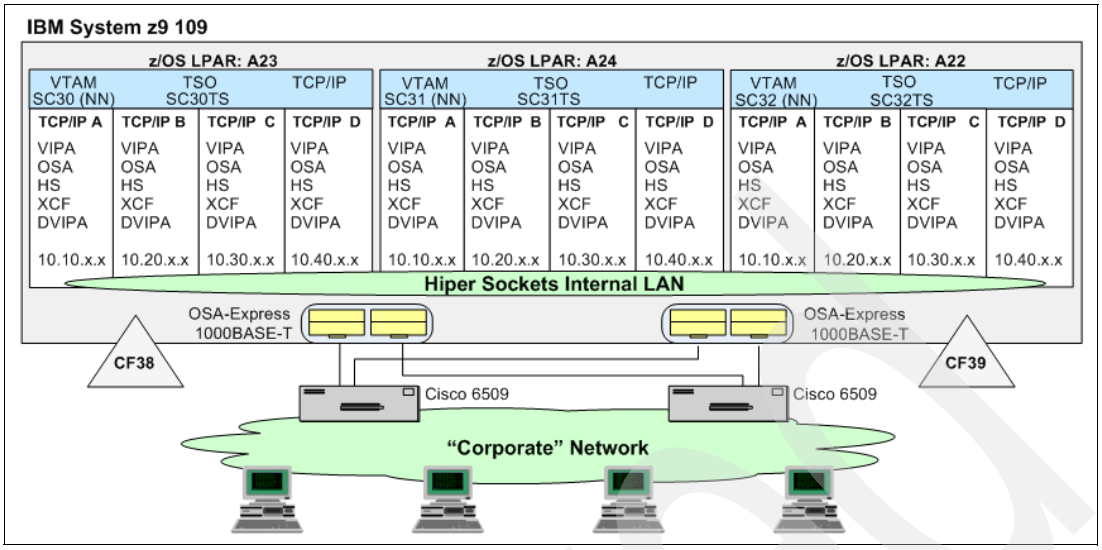


Figure 1 Our implementation environment

Our books were written (and implementation scenarios executed) using three logical partitions (LPARs) on an IBM System z9 109 (referred to as A22, A23, and A24). Because we were working on four books at the same time, we implemented *four* TCP/IP stacks on each LPAR (admittedly, a configuration not recommended for a production environment, but convenient for our purposes). Each LPAR shared:

- HiperSockets™ connectivity
- Coupling Facility connectivity (CF38 and CF39) for parallel sysplex scenarios
- Four OSA-Express2 1000BASE-T Ethernet ports cross-connected to a pair of Cisco 6509 switches

Finally, we shared ten workstations, representing corporate network access to the z/OS networking environment, for scenario verification (using applications such as TN3270 and FTP).

Our IP addressing convention is as follows:

- The first octet is the network (always 10 for our environment).
- The second Octet is the VLAN (10,20,30,40) assigned to the stack. (Essentially, except when required by a specific implementation scenario, each team's stacks shared a common VLAN.)
- The third octet refers to the device:
  - The addresses with the third octet of 2 or 3 are defined to the OSA devices.
  - The addresses with the third octet of 4, 5, or 6 are defined to the HiperSocket devices.
- The last octet is made up as follows:
  - The first two digits are the LPAR number.
  - The last digit is the CHPID number.

Important: Our use of multiple TCP/IP stacks on each LPAR and our TCP/IP addressing were set up for our convenience and are not recommended approaches for your environment.

Our focus for this book

Given the above actual environment, the (simplified) environment that we had to work with for this book is illustrated in Figure 2.

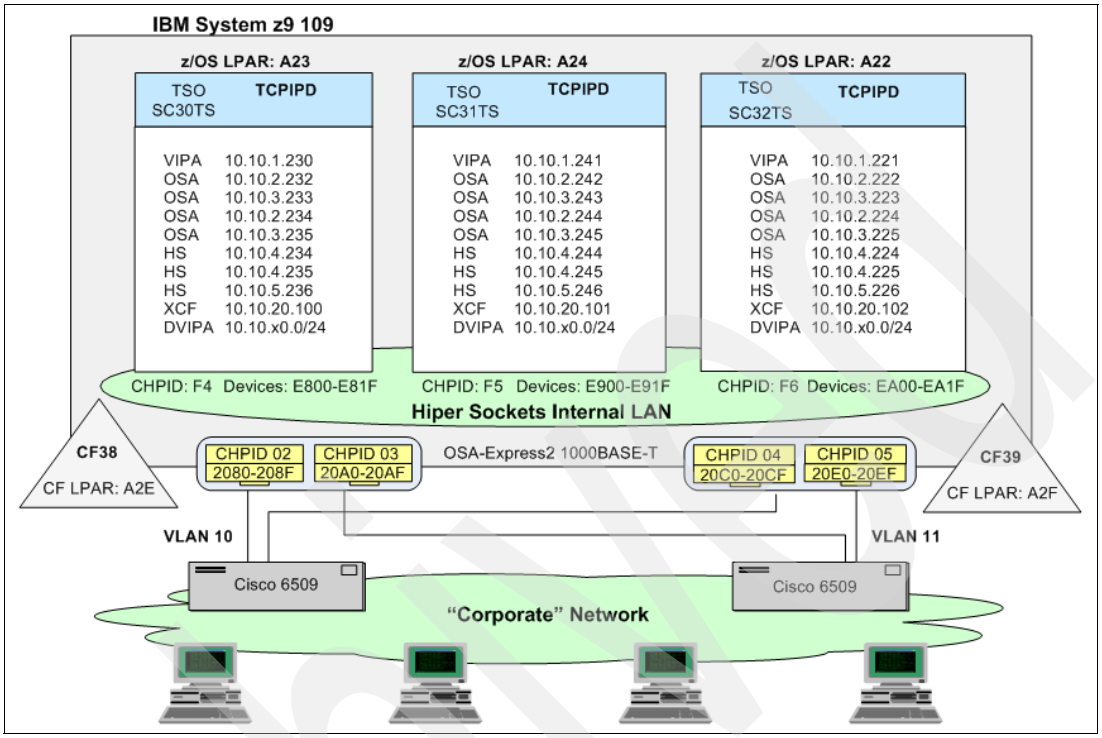


Figure 2 Our environment for this book

For the purposes of this book, we viewed the environment as three LPARs leveraging coupling facilities, HiperSockets, and OSA connectivity as required for our implementation scenarios.

Each of the books in our four-volume *Communications Server (CS) for z/OS TCP/IP Implementation* series was actually the result of very close cooperation and coordination across the entire team—a team with representation from seven different countries and with more than 200 years of combined information technology experience.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center. The following authors worked on the books in this series. The main authors who wrote this particular book are Bob Louden, Octavio Ferreria, and Sherwin Lake.

Octavio Ferreria is a Senior IT Specialist in IBM Brazil. He has 26 years of experience in IBM software support. His areas of expertise include z/OS Communications Server, SNA and TCP/IP, and the Communications Server in all platforms. For the last eight years, he has worked at the Area Program Support Group providing guidance and support to clients and designing networking solutions such as SNA/TCP/IP integration, z/OS Connectivity, Enterprise Extender design and implementation, and SNA-to-APPN migration.

Sherwin Lake is an IT Specialist with IBM Global Services in Research Triangle Park, North Carolina. Sherwin has worked over the past 30 years in VSE/SP, VM, and MVS™ systems environments. During that time he was an online and batch Applications Developer, Product Support Specialist, Network Analyst, IT Specialist, and Manager. Before joining IBM, Sherwin Managed the Technical Support group at the Trinidad and Tobago Telephone Company in Trinidad. Sherwin was a member of the team that migrated IBM from Office Vision to Lotus® Notes. He currently works with the Delivery part of IBM Global Services providing SNA and TCP/IP support to internal and external accounts. Sherwin holds a Bachelor of Science Degree in Computer Science/Math from the University of Miami.

Bob Louden is a consultant in the IBM Global Services, IT Services, Network Services Delivery practice. His twenty-three years as a networking professional with IBM have enabled him to develop strong professional and consulting skills, including leadership, project management, problem solving, and decision analysis. Bob's technical expertise includes wide-area and local-area networking and SNA and TCP/IP protocols. More important, however, is his ability to leverage technology understanding to develop business solutions.

Rama Ayyar is a Senior IT Specialist with the IBM Support Center in Sydney, Australia. Rama has over 22 years of experience with the MVS Operating System. His areas of expertise include TCP/IP, RACF®, DFSMS, z/OS Operating System, Configuration Management, Dump Analysis, and Disaster Recovery, and he has written six IBM Redbooks™. Rama holds a Master's Degree in Computer Science from the Indian Institute of Technology, Kanpur, and has been in the Computer industry for 34 years.

Valirio Braga is a Senior IT Specialist in Brazil working for the IBM Support Center. He has eight years of experience in networking. His areas of expertise include VTAM®, TCP/IP, CS for z/OS, and OSA. He has written the *OSA-Express Implementation Guide*, SG24-5948-04, and is a Cisco Certified Network Associate (CCNA).

Michael W. Jensen is a Senior IT Specialist with IBM Global Services, Strategic Outsourcing in Denmark. Michael has 23 years of experience with networking (OSA, SNA, APPN, TCP/IP, Cisco SNASw, SNA-to-APPN migration, and SNA/IP Integration) and IBM Mainframe systems, primarily focusing on z/OS Communications Server. His areas of expertise include design and implementation of Content Switching and load balancing solutions for the z/OS Sysplex environment using Cisco Series Products.

Garth Madella is an Information Technology Specialist with IBM South Africa. He has 20 years of experience in the S/390® networking software field. He has worked with IBM for nine years. His areas of expertise include VTAM, SNA, TCP/IP, and sysplex. He has written extensively on TCP/IP and Enterprise Extender issues.

Joel Porterie is a Senior IT Specialist who has been with IBM France for 28 years. He works for Network and Channel Connectivity Services in the EMEA Product Support Group. His areas of expertise include z/OS, TCP/IP, VTAM, OSA-Express, and Parallel Sysplex® for zSeries®. He has taught OSA-Express and FICON® problem determination classes and provided on-site assistance in these areas in numerous countries. He also co-authored the IBM Redbooks *Using the IBM S/390® Application StarterPak*, SG24-2095; *OSA-Express Gigabit Ethernet Implementation Guide*, SG24-5443; *OSA-Express Implementation Guide*, SG24-5948; and *Introduction to the New Mainframe: Networking*, SG24-6772.

Marc Price is a Staff Software Engineer with IBM Software Group in Raleigh, NC. Marc has over six years of experience with the Communications Server for z/OS as a member of the organization that develops and services the Communications Server for z/OS. His areas of expertise include TCP/IP, security, z/OS dump analysis, and a variety of operating systems. Marc holds a Bachelor's Degree in Computer Science from Purdue University in Indiana, USA. Marc has 10 years of experience in the computer industry.

Larry Templeton is a Network Architect with IBM Global Services, Network Outsourcing. He has 36 years of experience in IBM mainframe and networking systems, consulting with clients throughout the United States. His current responsibilities include architecting mainframe IP connectivity solutions, designing inter-company Enterprise Extender configurations, and assisting clients with high availability data center implementations.

Thomas Wienert is a Senior IT Specialist working for IBM z9 Field Technical Sales Support (FTSS) in Germany. He has over 20 years of experience with IBM networking. Thomas has been with IBM for 16 years and worked as a S/390 Systems Engineer in technical support and marketing. His areas of expertise include Communications Server for z/OS, Communication Controller for Linux®, OSA-Express, z/OS, Parallel Sysplex, and zSeries-related hardware. He has co-authored the *OSA-Express Implementation Guide*, SG24-5948-04, and is a Cisco certified Associate (CCNA).

Thanks to **Alfred Christensen**, Programming Consultant, Enterprise Networking Solutions, for his vision and drive to make these redbooks possible.

As is always the case with any complex technical effort such as this *Communications Server (CS) for z/OS TCP/IP Implementation* series, success would not have been possible without the advice, support, and review of many outstanding technical professionals. We are especially grateful for the significant expertise and contributions of content to these books from the Communications Server for z/OS Development team, especially from Jeannie Kristufek, Jeremy Geddes, Barry Mosakowski, Tony Amitrano, and Andy Tracy. Additionally, we would like to thank Roland Peschke for his excellent review and feedback.

Thanks also to the International Technical Support Organization, Raleigh and Poughkeepsie, for their invaluable support in this project, particularly Margaret Ticknor, Bob Haimowitz, David Bennin, Denice Sharpe, Eran Yona, Linda Robinson, Julie Czubik, and most importantly, Bill White—our ITSO mentor and guide.

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and clients.

Your efforts will help increase product acceptance and client satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYJ Building 662
  P.O. Box 12195
  Research Triangle Park, NC 27709-2195

1

Introduction

The z/OS Communications Server is the IBM implementation of the standard TCP/IP protocol suite on the z/OS platform. TCP/IP is a component product of the z/OS Communications Server that provides a multitude of technologies that collectively provide an Open Systems environment for the development, establishment, and maintenance of applications and systems.

The z/OS Communications Server product includes ACF/VTAM, in addition to TCP/IP.

This introduction to CS for z/OS TCP/IP presents a basic overview of the protocol suite as it is implemented in the z/OS environment. A more complete and comprehensive explanation of z/OS Communications Server IP may be obtained from the publications listed in “For additional information” on page 3.

This chapter discusses the following.

Section	Topic
1.1, “Overview” on page 2	Discusses the basic concepts of CS for z/OS IP
1.2, “Why CS for z/OS IP is important” on page 3	Discusses key characteristics of CS for z/OS IP and why it may be important in your environment
1.3, “How CS for z/OS IP is implemented” on page 4	Presents a functional overview of how CS for z/OS IP is implemented

1.1 Overview

CS for z/OS IP provides the industry-standard TCP/IP protocol suite, allowing z/OS environments to share data and computing resources with other TCP/IP computing environments, when authorized. CS for z/OS IP enables anyone in a non-z/OS TCP/IP environment to access resources in the z/OS environment and perform tasks and functions provided by the TCP/IP protocol suite. It provides the computer platform with the freedom desired by organizations to distribute workload to environments best suited to their needs. CS for z/OS IP, therefore, adds the z/OS environment to the list of environments in which an organization may share data and computer processing resources in a TCP/IP network.

CS for z/OS IP supports two environments:

- It provides a native MVS (z/OS) environment in which users may exploit the TCP/IP protocols in the z/OS applications environment. This includes batch jobs, started tasks, TSO, CICS® applications, and IMS™ applications.
- It also provides native TCP/IP support in the UNIX® Systems Services environment in which users may create and use applications that conform to the POSIX or XPG4 standard (a UNIX specification). The UNIX environment and services may also be exploited from the z/OS environment and vice versa.

1.1.1 Basic concepts

The TCP/IP address space is where the TCP/IP protocol suite is implemented for CS for z/OS IP. The TCP/IP address space is commonly referred to as a *stack*. In earlier versions Open Edition (OE) TCP/IP could run either as a stand-alone or in parallel with the TCP/IP for MVS. This was often necessary since the OE stack did not support all the functions and network connections available with TCP/IP for MVS. CS for z/OS IP now has a highly efficient direct communication between the UNIX System Services address space (OMVS) and a TCP/IP stack that was integrated in UNIX System Services. This communication path includes the UNIX System Services Physical File System (PFS) component for AF_INET and AF_INET6 (Addressing Family-Internet) sockets communication. The z/OS Communications Server has the following features:

- A process model that provides a full multiprocessing capability. It includes full duplex data paths of reduced lengths.
- An I/O process model that allows VTAM to provide the I/O device drivers. MultiPath Channel (MPC) Data Link Control (DLC) is shared between VTAM and TCP/IP. It executes multiple dispatchable units of work and is tightly integrated with the common storage management support.
- A storage management model handles expansion and contraction of storage resources as well as requests of varying sizes and types of buffers. Common Storage Management (CSM) manages communication between the Sockets PFS through the transport provider and network protocols to the network interface layer of CS for z/OS IP stack. The data that is placed in the buffers can be accessed by any function all the way down to the protocol stack.

In contrast to earlier versions of the product, CS for z/OS IP is integrated with OE and exploits UNIX Systems Services. CS for z/OS IP requires some UNIX Systems Services configuration. CS for z/OS IP runs as a single stack that serves both the traditional MVS (z/OS) environment and the z/OS UNIX (UNIX Systems Services) environment.

CS for z/OS IP offers two variants of the UNIX shell environment:

- The OMVS shell, which is much like a native UNIX environment

- The ISHELL, which is an ISPF interface with access to menu-driven command interfaces

The TCP/IP protocol suite is implemented by an MVS started task within the TCP/IP address space in conjunction with z/OS UNIX (UNIX System Services).

A CS for z/OS IP environment requires a Data Facility Storage Management Subsystem (DFSMS), a z/OS UNIX file system, and a security product such as Resource Access Control Facility (RACF). These resources must be defined and functional before the z/OS Communication Server can be started successfully and establish the TCP/IP environment. We later mention the manner in which these products impact a CS for z/OS IP environment.

1.1.2 For additional information

The following IBM publications provide further details for implementing a z/OS environment that supports the TCP/IP protocol suite:

- *z/OS V1R7.0 Communications Server: New Function Summary*, GC31-8771
- *z/OS V1R7.0 Communications Server: IP Configuration Reference*, SC31-8776
- *z/OS V1R7.0 Communications Server: IP Configuration Guide*, SC31-8775
- *z/OS V1R7.0 Communications Server: IP Sockets Application Programming Interface Guide and Reference*, SC31-8788

1.2 Why CS for z/OS IP is important

The z/OS Communications Server provides a high-performance, highly secure, scalable, and reliable platform on which to build and deploy networking applications.

CS for z/OS IP is important because it offers an environment that is accessible to the enterprise IP network and the Internet if so desired. It defines the z/OS environment as a viable platform by making z/OS applications and systems available to the non-z/OS environment, which are typically UNIX/Windows® centric. Consequently, it eliminates the issues and challenges of many large corporations to migrate or integrate with a more accessible platform and newer technologies. Many technologies have been implemented in the z/OS environment to complement TCP/IP. Some of these include:

- High-speed connectivity, such as:
  - OSA-Express Gigabit Ethernet or 1000BASE-T in QDIO mode
  - High-speed communication between TCP/IP stacks running in logical partitions using HiperSockets
- High availability for applications using Parallel Sysplex technology in conjunction with:
  - Dynamic Virtual IP Address (*VIPA*), which provides *TCP/IP application availability across z/OS systems* in a Sysplex and allows participating TCP/IP stacks to provide backup and recovery for each other, for planned and unplanned TCP/IP outages
  - *Sysplex Distributor*, which provides intelligent load balancing for *TCP/IP application servers* in a Sysplex, and along with *Dynamic VIPA* provides a single system image for client applications connecting to those servers
  - The Load Balancing Advisor (LBA), which provides z/OS Sysplex server application availability and performance data to *outboard load balancers via the Server Application State Protocol* (SASP)

- Enterprise connectivity support is offered through many features, such as:
  - TN3270 Server, which provides *workstation connectivity over TCP/IP networks* to access z/OS and enterprise SNA applications.
  - Enterprise Extender, which allows SNA Enterprise applications to communicate reliably over an IP network, using SNA HPR and UDP transport layer protocols.
  - IPv4 and IPv6 networking functions are provided by the TCP/IP stack operating in a standard dual-mode setup where IPv4 and IPv6 connectivity and applications are supported concurrently by a single TCP/IP stack instance.
  - Sockets programming interface support for traditional z/OS workloads provide IP connectivity to applications written in REXX, COBOL, PL/I. Sockets programming interfaces are supported in various environments, such as TSO, batch, CICS, and IMS.
- Network Security protects sensitive data and the operation of the TCP/IP stack on z/OS, by using:
  - IPsec/VPN functions that enable the secure transfer of data over a network using standards for encryption, authentication, and data integrity.
  - Intrusion Detection Services (IDS), which evaluates the stack for attacks that would undermine the integrity of its operation. Events to examine and actions to take (such as logging) at event occurrence are defined by the IDS policy, which is downloaded from an LDAP server.
  - *Transport Layer Security* (TLS) enablement ensures data is protected as it flows across the network.
  - Kerberos and GSSAPI support is provided for selected applications.
- Network Management support collects network topology, status, and performance information and makes it available to *network management tools*, including:
  - Local management applications that can access management data via a specialized high-performing network management programming interface that is known as *NMI*.
  - Support of remote management applications through the SNMP protocol. CS z/OS supports the latest SNMP standard, SNMPv3. CS z/OS also supports standard TCP/IP-based Management Information Base (MIB) data.
  - Additional MIB support is also provided by Enterprise-specific MIB, which supports management data for Communications Server TCP/IP stack-specific functions.

1.3 How CS for z/OS IP is implemented

CS for z/OS IP provides TCP/IP support for the native MVS and UNIX System Services environment. It is implemented within a z/OS address space and runs within the native MVS environment, and consequently it has RACF, DFSMS, and z/OS UNIX file system dependencies.

1.3.1 Functional overview

CS for z/OS IP takes advantage of Communications Storage Management (CSM) and of VTAM's Multi-Path Channel (MPC) and Queued Direct I/O (QDIO) capabilities in its TCP/IP protocol implementation. This tight coupling with VTAM provides enhanced performance and serviceability.

As illustrated in Figure 1-1 on page 5, there are many data link control (DLC) protocols provided the Communications Server. These DLCs are provided by the VTAM component.

In CS for z/OS IP, two worlds converge, providing access to the z/OS UNIX environment and the traditional MVS environment.

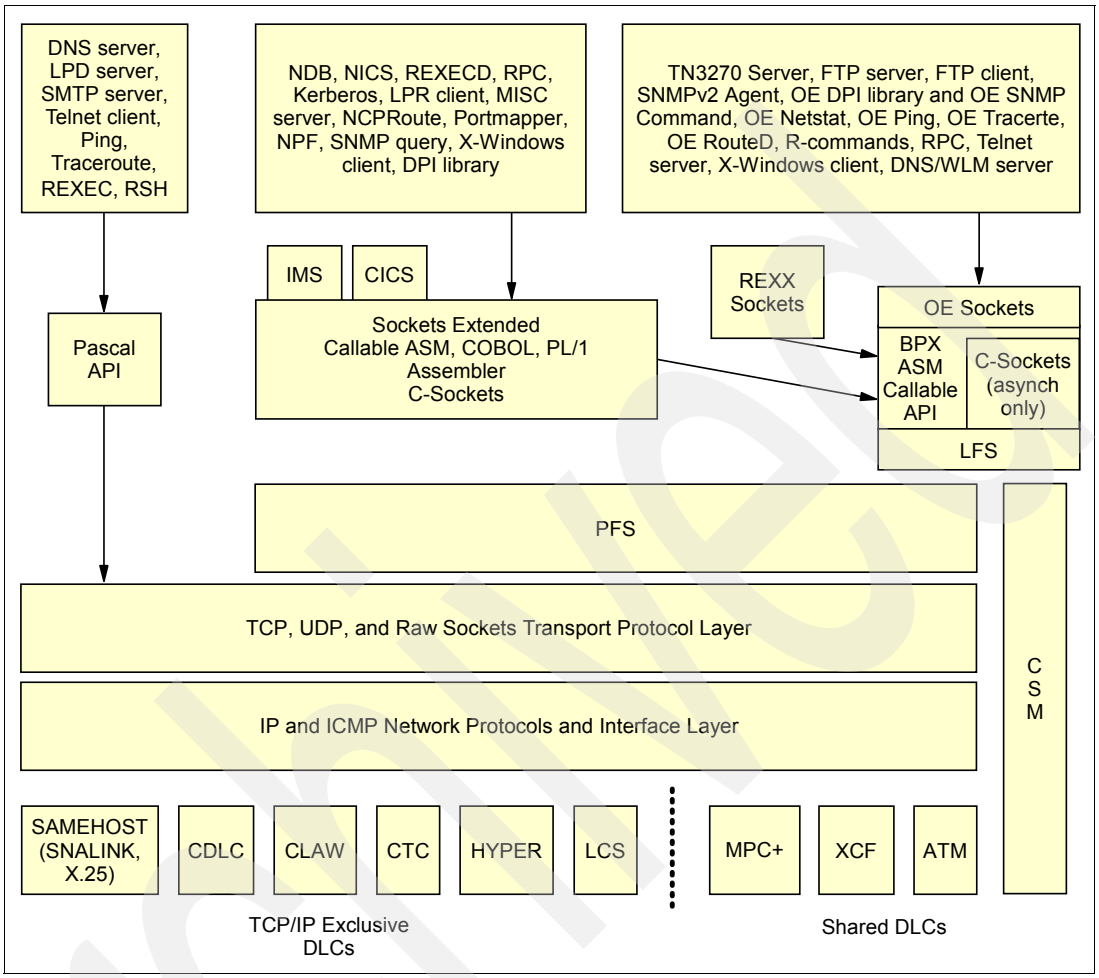


Figure 1-1 Functional overview

1.3.2 Operating environment

Since the z/OS UNIX environment is supported in the MVS environment there is no need to discuss the creation of an MVS environment. However, there are some customization requirements on the UNIX Systems Services side of the environment in order to start CS for z/OS IP successfully. This dependence on UNIX, of course, implies that z/OS UNIX administrators must also be familiar with both traditional MVS commands and interfaces.

Input/output flow process

Another feature of the operating environment illustrated in Figure 1-1 is the storage and input/output designs. The operating environment design features a tightly integrated storage and I/O model, known as CSM.

Communications Storage Management (CSM)

The CSM Facility is used by authorized programs to manage subsystem storage pools. It provides a flat storage model that is accessible by multiple layers of the process model, as Figure 1-1 illustrates. It is also accessible across z/OS address space boundaries, thereby reducing the data moves between processes/tasks that exchange data as they perform work.

VTAM and TCP/IP tasks are typical examples. The CSM facility also manages storage as it automates the addition and subtraction of the different types and sizes of storage requests.

1.3.3 Protocols and devices

As illustrated in Figure 1-1 on page 5, the Data Link Control (DLC) is a protocol layer that manages and provides communication between the file I/O subsystem and the I/O device driver of the particular device. It also shows two categories of DLCs: TCP/IP exclusive DLCs and shared DLCs.

Exclusive DLCs

TCP/IP exclusive DLCs are those only available for TCP/IP usage and are not shared with ACF/VTAM. Examples of TCP/IP exclusive DLCs supported by CS for z/OS IP are as follows.

LCS

LAN Channel Station (LCS) protocol is used by OSA-Express, some routers, and the 3746-9x0 MAE.

SAMEHOST

SAMEHOST is another TCP/IP exclusive DLC protocol that exists, although it does not make use of System z9 or zSeries channels. In the past, this communication was provided by IUCV. Currently, three such servers (SNALINK LU0, SNALINK LU6.2, X.25) exploit the SAMEHOST DLC.

Shared DLCs

Shared DLCs are those that can be simultaneously used by TCP/IP and ACF/VTAM. The shared DLCs are indicated in Figure 1-1 on page 5; however, we only focus on the most commonly used DLCs, such as the following.

Multipath Channel+ (MPC+)

MPC+ is an enhanced version of VTAM's Multipath Channel (MPC) protocol. The Multipath Channel I/O process (MPC) defines the implementation of the MPC protocols. It allows for the efficient use of multiple read and write channels. MPC handles protocol headers and data separately and executes multiple I/O dispatchable units of work. This, when used in conjunction with Communication Storage Management, creates efficient I/O throughput. High Performance Data Transfer uses MPC+ together with Communication Storage Manager (CSM) to decrease the number of data copies required to transmit data. This type of connection may be used in two ways:

- MPCPTP allows a CS for z/OS IP environment to connect to a peer IP stack in a point-to-point configuration. With MPCPTP a CS for z/OS IP stack may be connected to:
  - Another CS for z/OS IP stack
  - An IP router with corresponding support
  - A non-z/OS server
  - 3746-9x0 MAE

  PTP Samehost (MPCPTP), sometimes referred to IUTSAMEH: This connection type is used to connect two or more CS for z/OS IP stacks running on the same z/OS LPAR. In addition, it can be used to connect these CS for z/OS IP stacks to z/OS VTAM for the use of Enterprise Extender.
- MPCIPA allows an Open Systems Adapter-Express (OSA-Express) port to act as an extension of the CS for z/OS TCP/IP stack and not as a peer TCP/IP stack, as with MPCPTP.

– OSA-Express provides a mechanism for communication called Queued Direct I/O (QDIO). Although it uses the MPC+ protocol for its control signals, the QDIO interface is quite different from channel protocols. It uses Direct Memory Access (DMA) to avoid the overhead associated with channel programs. A partnership between CS for z/OS IP and the OSA-Express adapter provides compute-intensive functions from the System z9 or zSeries server to the adapter. Segmentation offload is a feature of OSA-Express2. This interface is called IP Assist (IPA). Offloading reduces System z9 and zSeries server cycles required for network interfaces and provides an overall improvement in the OSA-Express environment compared to existing OSA-2 interfaces. OSA-Express collaborates with CS for z/OS TCP/IP to support Gigabit Ethernet, 1000BASE-T, Fast Ethernet, Fast Token-Ring, and ATM LAN emulation.

– HiperSockets (Internal Queued Direct I/O, iQDIO) provides high-speed, low-latency IP message passing between Logical Partitions (LPARs) within a single System z9 or zSeries server. The communication is through processor system memory via Direct Memory Access (DMA). The virtual servers that are connected via HiperSockets form a virtual LAN. HiperSockets uses internal QDIO at memory speeds to pass traffic between virtual servers.

Cross-System Coupling Facility (XCF)

XCF allows communication between multiple CS for z/OS IP stacks within a Parallel Sysplex. The XCF DLC can be defined, as with traditional DLCs, but it also supports XCF Dynamics, in which the XCF links are brought up automatically.

If DYNAMICXCF is coded, z/OS images within the same server will use the HiperSockets DYNAMICXCF connectivity instead of the standard XCF connectivity for data transfer.

For more information about devices and connectivity options refer to Chapter 4, “Connectivity” on page 101.

1.3.4 Supported routing applications

z/OS Communications Server ships two routing applications, OrouteD and OMPROUTE. CS for z/OS IP V1R7 is the last release to support OrouteD.

OMPROUTE implements the Open Shortest Path First protocols (OSPF and OSPVFv3) and Routing Information Protocols (RIPv1, RIPv2, RIPng). It enables the CS for z/OS IP to function as an OSPF/RIP-capable router in a TCP/IP network. Either (or both) of these two routing protocols can be used to dynamically maintain the host routing table. Additionally, CS for z/OS IP provides an OMPROUTE subagent that implements the OSPF MIB variable containing OSPF protocol and state information for SNMP. This MIB variable is defined in RFC 1850. Refer to Chapter 5, “Routing” on page 139, for a detailed discussion on OMPROUTE and its function within the CS for z/OS IP environment.

1.3.5 Application programming interfaces

As Figure 1-1 on page 5 illustrates, all of the APIs provided by CS for z/OS IP, with the exception of the PASCAL API, interface with the Logical File System (LFS) layer. The APIs are divided into the following categories:

- Pascal
- TCP/IP socket APIs
- z/OS UNIX APIs

Pascal API

The Pascal application programming interface enables you to develop TCP/IP applications in Pascal language. Supported environments are normal MVS address spaces. Unlike the other APIs, the Pascal API does not interface directly with the LFS. It uses an internal interface to communicate with the TCP/IP protocol stack. The Pascal API only supports AF_INET.

TCP/IP socket APIs

The z/OS V1R7 Communications Server provides several APIs to access TCP/IP sockets. These APIs can be used in either or both integrated and common INET PFS configurations. In a common INET PFS configuration, however, they function differently from z/OS UNIX APIs. In this type of configuration, the z/OS Communications Server APIs always bind to a single PFS transport provider, and the transport provider must be the TCP/IP stack provided by the z/OS V1R7 Communications Server. The following TCP/IP socket APIs are included in the z/OS V1R7 Communications Server:

- The CICS socket interface enables you to write CICS applications that act as clients or servers in a TCP/IP-based network. CICS sockets only support AF_INET.
- The C sockets interface supports socket function calls that can be invoked from C programs. However, note that for C application development, IBM recommends the use of the UNIX C sockets interface. These programs can be ported between MVS and most UNIX environments relatively easily if the program does not use any other MVS-specific services. C sockets only support AF_INET.
- The Information Management System (IMS) IPv4 socket interface supports 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 IMS sockets API supports AF_INET.
- The Sockets Extended macro API is a generalized assembler macro-based interface to sockets programming. The Sockets Extended macro API supports AF_INET and AF_INET6.
- The Sockets Extended Call Instruction API is a generalized call-based, high-level language interface to sockets programming. The Sockets Extended Call Instruction API supports AF_INET and AF_INET6.

z/OS UNIX APIs

The following APIs are provided by the z/OS UNIX element of z/OS and are supported by the TCP/IP stack in the z/OS V1R7 Communications Server:

- z/OS UNIX C sockets is used in the z/OS UNIX environment. It is the z/OS UNIX version of the native MVS C sockets programming interface. Programmers use this API to create applications that conform to the POSIX or XPG4 standard (a UNIX specification). The z/OS UNIX C sockets support AF_INET and AF_INET6.
- z/OS UNIX assembler callable services is a generalized call-based, high-level language interface to z/OS UNIX sockets programming. The z/OS UNIX assembler callable services support AF_INET and AF_INET6.

Refer to the *z/OS V1R7.0 XL C/C++ Compiler and Run-Time Migration Guide for the Application Programmer CBCMG140 06/28/05 16:05:01*, GC09-4913-03, for complete documentation of the z/OS UNIX C sockets APIs and refer to *z/OS V1R7.0 UNIX System Services Programming Tools BPXZA640 05/03/05 16:19:37*, SA22-7805-04 .

REXX sockets

The REXX sockets programming interface implements facilities for socket communication directly from REXX programs by using an address rxsocket function. REXX socket programs

can execute in TSO, online, or batch. The REXX sockets programming interface supports AF_INET and AF_INET6.

Refer to *z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference: z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference Version 1 Release 7*, SC31-8788-05, for complete documentation of the TCP/IP Services APIs.

1.3.6 z/OS Communications Server applications

z/OS Communications Server TCP/IP support provides a number of standard client and server applications, including:

- SNA 3270 Logon Services (TN3270)
- z/OS UNIX logging services (syslogd)
- File Transfer Services (FTP)
- Network Management Services (SNMP Agents, Subagents, Trap forwarding)
- IP Printing (LPR, LPD, Infoprint Server)
- Internet Daemon Listener (INETD)
- Mail Services (SMTP and sendmail)
- z/OS UNIX logon services (otelnetd)
- Remote Execution (REXECD, RSHD, REXEC, RSH, orexecd, orshd, orexec, orsh)
- Domain Name Services (Caching DNS BIND9 server)

These applications are discussed in detail in *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 2 - Standard Applications*, SG24-7170, and the *z/OS V1R7.0 Communications Server: IP Configuration Reference*, SC31-8776.

1.3.7 UNIX Systems Services

UNIX System Services is the z/OS Communications Server implementation of UNIX as defined by X/Open in the XPG 4.2. UNIX System Services coexists with traditional MVS functions and traditional MVS file types (partitioned data sets, sequential files, and so on). It concurrently allows access to z/OS UNIX file system files and to UNIX utilities and commands by means of application programming interfaces and the interactive shell environment. CS for z/OS IP offers two variants of the UNIX shell environment: The z/OS shell (the default shell) and the tcsh shell (or Ishell) (an enhanced version of the Berkeley UNIX C shell). The Communications Server for z/OS IP requires that UNIX System Services be customized in full-function mode before the TCP/IP stack will successfully initialize. We will therefore conduct an overview of UNIX System Services to provide an appreciation for the coding and security considerations involved with UNIX System Services.

Customization levels of UNIX System Services

There are two levels of z/OS UNIX services:

- *Minimum mode*, indicating that although OMVS initializes, it provides few z/OS UNIX services, and there is no support for TCP/IP and the z/OS shell. In this mode there is no need for DFSMS or for a security product such as RACF.
- *Full-function mode*, indicating that the complete array of z/OS UNIX services is available. In this mode DFSMS, RACF, and the z/OS UNIX file system are required. TCP/IP and z/OS UNIX file system interaction with UNIX System Services is defined within the BPXPRMxx member of SYS1.PARMLIB.

z/OS V1R7.0 UNIX System Services Planning, SA22-7800, provides a good description of the UNIX System Services customization process. It also includes a chapter devoted to TCP/IP.

UNIX System Services concepts

z/OS UNIX enables two open systems interfaces on the z/OS operating system: An application program interface (API) and an interactive shell interface.

With the APIs, programs can run in any environment (including batch jobs, in jobs submitted by TSO/E interactive users, and in most other started tasks) or in any other MVS application task environment. The programs can request:

- Only MVS services
- Only z/OS UNIX services
- Both MVS and z/OS UNIX services

The shell interface is an execution environment analogous to TSO/E, with a programming language of shell commands analogous to the Restructured eXtended eXecutor (REXX) language. The shell work consists of:

- Programs that are run interactively by shell users
- Shell commands and scripts that are run interactively by shell users
- Shell commands and scripts that are run as batch jobs

In z/OS UNIX Systems Services, address spaces are provided by the fork() or spawn() functions of the Open Edition callable services.

For a fork(), the system copies one process, called the parent process, into a new process, called the child process, and places the child process in a new address space, the forked address space.

A Spawn() also starts a new process in a new address space. Unlike a fork(), in a spawn() call the parent process specifies a name of a program to be run in the child process.

The types of processes can be:

- User processes, which are associated with a user
- Daemon processes, which perform continuous or periodic system-wide functions, such as a Web server

  Daemons (a UNIX concept) are programs that are typically started when the operating system is initialized and remain active to perform standard services. Some programs are considered daemons that initialize processes for users even though these daemons are not long-running processes. Examples of daemons provided by z/OS UNIX are *cron*, which starts applications at specific times, and *inetd,* which provides service management for a network.

A process can have one or more threads. A thread is a single flow of control within a process. Application programmers create multiple threads to structure an application in independent sections that can run in parallel for more efficient use of system resources.

UNIX Hierarchical File System

Data sets and files are comparable terms. If you are familiar with MVS, you probably use the term *data set* to describe a unit of data storage. If you are familiar with AIX® or UNIX, you probably use the term *file* to describe a named set of records stored or processed as a unit. In the UNIX System Services environment, the files are arranged in a z/OS UNIX file system.

The Hierarchical File System allows you to set up a file hierarchy that consists of:

- Directories, which contain files, other directories, or both. Directories are arranged hierarchically, in a structure that resembles an upside-down tree, with the root directory at the top and branches at the bottom.

- z/OS UNIX file system files, which contain data or programs. A file containing a load module, shell script, or REXX program is called an executable file. Files are kept in directories.
- Additional local or remote file systems, which are mounted on directories of the root file system or of additional file systems.

To the MVS system, the UNIX file hierarchy appears as a collection of zSeries File System (zFS) data sets. Each z/OS UNIX file system data set is a mountable file system. The root file system is the first file system mounted. Subsequent file systems can be logically mounted on a directory within the root file system or on a directory within any mounted file system.

Each mountable file system resides in a z/OS UNIX file system data set on direct access storage. DFSMS/MVS® manages the z/OS UNIX file system data sets and the physical files.

For more information about the z/OS UNIX file system, refer to the *z/OS V1R7.0 CS: IP Migration,* GC31-8773, and *z/OS V1R7.0 UNIX System Services Planning*, SA22-7800.

z/OS UNIX file system definitions in BPXPRMxx

To get UNIX System Services active in full-function mode, you need the root file system defined in the BPXPRMxx member of SYS1.PARMLIB. The root file system is usually loaded or copied at z/OS installation time. The BPXPRMxx definition is detailed in a *z/OS V1R7.0 UNIX System Services Planning*, GA22-7800-08.

An important part of your z/OS UNIX file system is located in the /etc directory. The /etc directory contains some basic configuration files of UNIX System Services and most applications keep their configuration files in there as well. To avoid losing all of your configuration when you upgrade your operating system, it is recommended that you put the /etc directory in a separate z/OS UNIX file system data set and mount it at the /etc mountpoint. Refer to *z/OS V1R7.0 UNIX System Services Planning*, GA22-7800-08 for more information about the /etc directory.

z/OS UNIX user identification

All users of an MVS system, including users of z/OS UNIX functions, must have a valid MVS user ID and password. To use standard MVS functions, the user must have the standard MVS identity based on the RACF user ID and group name.

If a unit of work in MVS uses z/OS UNIX functions, this unit of work must have, in addition to a valid MVS identity, a z/OS UNIX identity. A z/OS UNIX identity is based on a UNIX user ID (UID) and a UNIX group ID (GID). Both UID and GID are numeric values ranging from 0 to 2147483647 (2^{31}-1). In a z/OS UNIX system, the UID is defined in the OMVS segment in the user's RACF user profile, and the GID is defined in an OMVS segment in the group's RACF group profile. What we in an MVS environment call the user ID is in a UNIX environment normally termed the user name or the login name. It is the name the user uses to present himself or herself to the operating system. In both a z/OS UNIX system and other UNIX systems, this user name is correlated to a numeric user identification, the UID, which is used to represent this user wherever such information has to be stored in the z/OS UNIX environment. One example of this is in the Hierarchical File System, where the UID of the owning user is stored in the file security portion of each individual file.

Access to resources in the traditional MVS environment is based on the MVS user ID, group ID, and individual resource profiles that are stored in the RACF database.

Access to z/OS UNIX resources is granted *only* if the MVS user ID has a valid OMVS segment with an OMVS UID or if a default user is configured as explained below. Access to resources in the Hierarchical File System is based on the UID, the GID, and file access permission bits that are stored with each file. The permission bits are three groups of three bits each. The groups describe:

- The owner of the file itself
- The users with the same GID as the owner
- The rest of the world

The three bits are:

- Read access
- Write access
- Search access if it is a directory or if it is a file that is executable

The superuser UID has a special meaning in all UNIX environments, including the z/OS UNIX environment. This user has a UID of zero and can access every resource.

In lieu of or in addition to RACF definitions for individual users, you may define a *default user*. The default user will be used to allow users without an OMVS segment defined to access UNIX System Services. The default user concept should be used with caution, since it could become a security exposure.

You will also find more information about the RACF security aspects of implementing the Communications Server for z/OS IP in *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 4 - Security*, SG24-7172.

Accessing the z/OS UNIX shells

The following ways are available to access the z/OS UNIX shells:

- The TSO/E **OMVS** command provides a 3270 interface in the z/OS UNIX shell.
- The TSO/E **ISHELL** or **ISH** command provides a 3270 interface that uses ISPF dialogs.
- The **rlogin** command provides an ASCII interface.
- The **Telnet** command provides an ASCII interface. This Telnet is into the UNIX Telnet daemon and not the TN3270E server in the z/OS system space.
- From a TCP/IP network, the TN3270(E) command, which provides a full-screen 3270 interface for executing the OMVS or ISHELL commands.

There are two shells, the z/OS shell and the Ishell. The login shell is determined by the PROGRAM parameter in the RACF OMVS segment for each user. The default is the z/OS shell.

Further information about the z/OS UNIX shells can be found in *z/OS V1R7.0 UNIX System Services User's Guide*, SA22-7801.

Operating mode

When a user first logs on to the z/OS UNIX shell, the user is operating in line mode. Depending on the method of accessing the shell, the user may then be able to use utilities that require raw mode (such as vi) or run an X-Windows application.

The different workstation operating modes are:

- Line mode

  Input is processed after you press Enter. This is also called canonical mode.

- Raw mode

  Each character is processed as it is typed. This is also called non-canonical mode.

- Graphical mode

  This is a graphical user interface for X-Windows applications.

UNIX System Services communication

A socket is the endpoint of a communication path; it identifies the address of a specific process at a specific computer using a specific transport protocol. The exact syntax of a socket address depends on the protocol being used, that is, on its *addressing family*. When you obtain a socket via the socket() system call, you pass a parameter that tells the socket library to which addressing family the socket should belong. All socket addresses within one addressing family use the same syntax to identify sockets.

Socket addressing families in UNIX System Services

In a z/OS UNIX environment, the most widely used addressing families are AF_INET and AF_UNIX. There is IPv6 support (AF_INET6 addressing family) in Communications Server for z/OS IP in a single transport driver environment configured in Dual-mode. Socket applications written to the IPv6 APIs can use the z/OS TCP/IP stack for IPv6 network connectivity. Throughout this discussion whatever reference is made for AF_INET (IPv4) also applies to AF_INET6 (IPv6).

The z/OS UNIX Systems Services implements support for a given addressing family through different physical file systems. There is one physical file system for the AF_INET addressing family and there is another for the AF_UNIX addressing family. A PFS is the part of the z/OS UNIX operating system that handles the storage of data and its manipulation on a storage medium.

AF_UNIX addressing family

The UNIX addressing family is also referred to as the UNIX domain. If two socket applications on the same MVS image want to communicate with each other, they may open a socket as an AF_UNIX family socket. In that case, the z/OS UNIX address space will handle the full communication between the two applications (see Figure 1-2). That is, the AF_UNIX physical file system is self-contained within z/OS UNIX and does not rely on other products to implement the required functions.

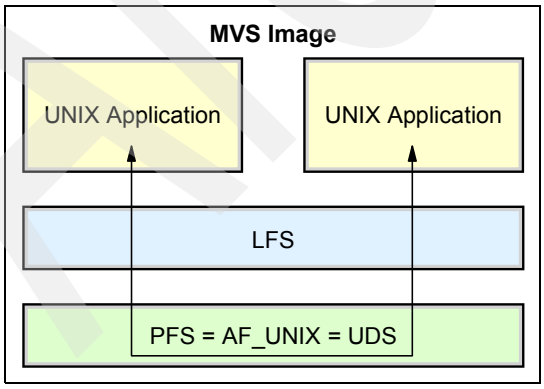


Figure 1-2 AF_UNIX sockets

AF_INET addressing family

This is the Internet addressing family, also referred to as the Internet domain. Socket programs communicate with socket programs on other hosts in the IP network using AF_INET family sockets which, in turn, use the AF_INET physical file system.

You may configure either AF_INET or both AF_INET and AF_INET6. You may not define the stack as IPv6 only. Although coding AF_INET6 alone is not prohibited, TCP/IP will not start since the master socket is AF_INET and the call to open it will fail.

For more on this subject refer to Chapter 3, “Base functions” on page 49, or *z/OS V1R7.0 UNIX System Services Planning,* SA22-7800.

The AF_INET physical file system relies on other products to provide the AF_INET transport services to interact with UNIX System Services and its sockets programs.

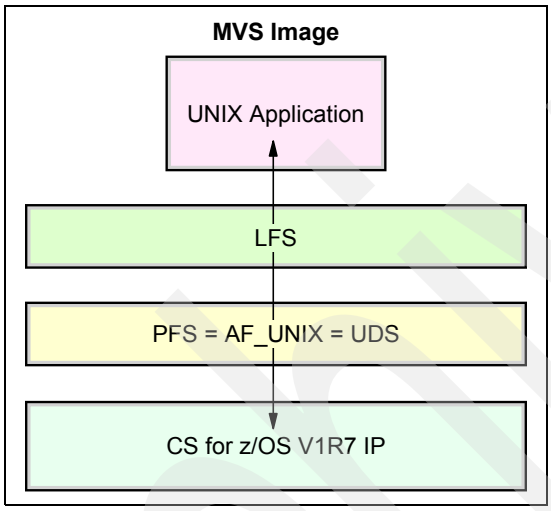


Figure 1-3 AF_INET sockets

For AF_INET/AF_INET6 sockets, the z/OS UNIX address space routes the socket request to the TCP/IP address space directly. As you see in Figure 1-3, the sockets/Physical File System layer is a transform layer between z/OS UNIX and the TCP/IP stack.

The sockets/PFS effectively transforms the sockets calls from the z/OS UNIX interface to the TCP/IP stack regardless of the version of MVS or TCP/IP. The sockets/PFS handles the communication between the TCP/IP address space and the z/OS UNIX address space in much the same manner as High Performance Native Socket (HPNS) handles the communication between the TCP/IP address space and the TCP/IP client and server address spaces.

Physical File System transport providers

TCP/IP requires the use of the Physical File System (AF_INET). The Physical File System may be configured in two ways: The Integrated Sockets File System type (INET) or the Common INET Physical File System type (CINET). INET is used in a single-stack environment and CINET is used in a multiple-stack environment.

A single Physical File System transport provider

If your background is in a UNIX environment, this may seem to be a strange question to ask (INET or CINET), since you are used to the TCP/IP protocol stack being an integral part of the UNIX operating system. This is not the case in a z/OS environment; it is very versatile. In this

environment you may start multiple instances of a TCP/IP protocol stack, each stack running on the same operating system, but each stack having a unique TCP/IP identity in terms of network interfaces, IP addresses, host name, and sockets applications.

A simple example of a situation where you have more TCP/IP stacks running in your z/OS system is if you have two separate IP networks, one production and one test (or one secure and one not); you do not want routing between them, but you want to give hosts on both IP networks access to your z/OS environment. In this situation you could implement two TCP/IP stacks, one connected to the production IP network and another connected to the test network.

This multi-stack implementation in which you share the UNIX System Services across multiple TCP/IP stacks provides challenges. Sockets applications that need to have an affinity to a particular stack need special considerations, in some cases including the coordination of port number assignments in order to avoid conflicts. This subject is discussed in Chapter 3, “Base functions” on page 49.

If a single AF_INET(6) transport provider is sufficient, use the Integrated Sockets physical file system (INET). If you need more than one AF_INET(6) transport provider (multiple TCP/IP stacks), you must use the Common INET physical file system (CINET).

You can customize z/OS to use the Common INET physical file system with just a single transport provider (AF_INET(6), but it is generally not recommended due to a slight performance decrease as compared to the Integrated Sockets Physical File System (INET). However, you may consider doing this if you expect to run multiple stacks in the future.

The PFS is also known under the name INET, and this appears in UNIX System Services definitions when a FILESYSTYPE and NETWORK TYPE need to be defined in the BPXPRMxx member of SYS1.PARMLIB.

Common INET Physical File System (CINET)

If you have two or more AF_INET transport providers on an MVS image, such as a production TCP/IP stack together with a test TCP/IP stack, you must use the Common INET Physical File System. Figure 1-4 on page 16 shows a multiple stack environment with Common INET (CINET).

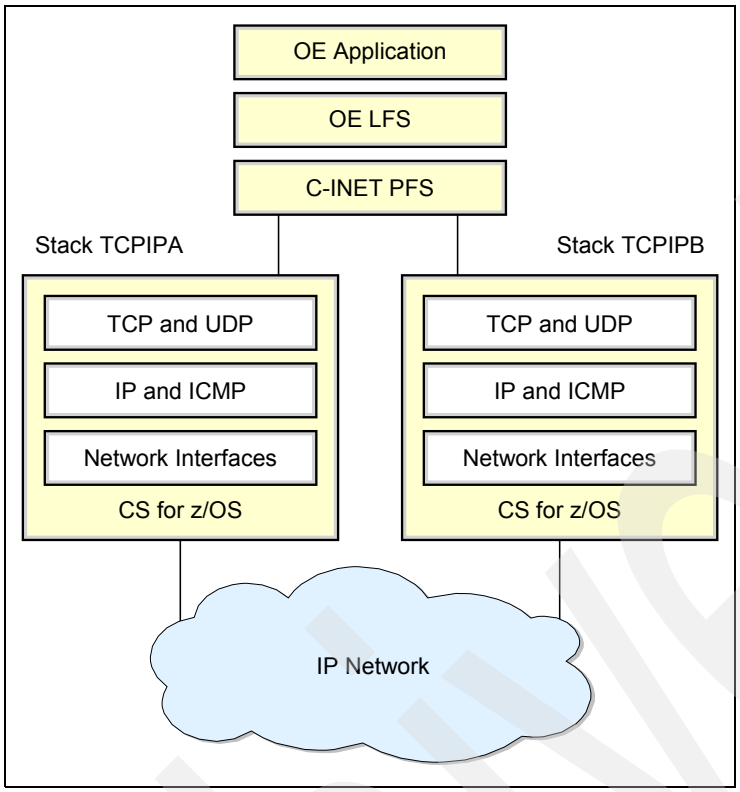


Figure 1-4 Multiple INET transport providers: CINET PFS

2

The Resolver

TCP/IP protocols rely upon a strict system of addressing in order to reach a host in a network. IPv4 addresses are represented in dotted-decimal format. The 32-bit address is divided along 8-bit boundaries. Each set of 8 bits is converted to its decimal equivalent and separated by periods. In contrast, IPv6 addresses are 128 bits divided along 16-bit boundaries. Each 16-bit block is converted to a 4-digit hexadecimal number and separated by colons. The resulting representation is called colon-hexadecimal. The major drawback of this addressing system is that, for most people, numbers are difficult to remember.

As a result, today's IP-based networks use a mapping of host names to host numbers or addresses. The obvious advantage of this name-to-address mapping is that we can assign easy-to-remember names to hosts in the network.

The Resolver is used to help applications establish a connection to a service that translates host names and IP addresses.

This chapter discusses the following.

Section	Topic
2.1, “Overview” on page 18	Discusses the basic concepts of the Resolver and address space
2.2, “Why the Resolver address space is important” on page 23	Discusses key characteristics of the Resolver address space and why it may be important in your environment
2.3, “The common design scenarios for the Resolver” on page 24	Presents commonly implemented Resolver design scenarios, their dependencies, advantages, considerations, and our recommendations
2.4, “How the Resolver address space is implemented” on page 31	Presents selected implementation scenarios, tasks, configuration examples, and problem determination suggestions

2.1 Overview

A Resolver is a set of routines that act as a client on behalf of an application to read a local host file or to access one or more Domain Name System (DNS) for name-to-IP address or IP address-to-name resolution.

Where the Resolver looks for name-address resolution and how it handles requests, is defined in a file called the Resolver configuration file. The Resolver can exist on any system that supports sockets programs, using name resolution.

On a z/OS system, the Resolver configuration data is located either in an z/OS dataset (TCPIP.DATA) or in a file (resolv.conf) in the z/OS UNIX file system.

Note: The *z/OS V1R7.0 Communications Server: IP Configuration Guide*, SC31-8775, contains useful information about the characteristics that are required for the z/OS data sets or file system files that contain resolver setup and configuration statements. The Guide also points out the security characteristics and file system permission settings that are needed.

The Resolver configuration on z/OS can be somewhat complex, because of the flexibility provided to control how the Resolver behaves based on the socket environment and local preferences. The Resolver function in a z/OS environment is performed by a component called the *Resolver address space*, which was introduced to simplify the configuration tasks.

2.1.1 Basic concepts

In most systems, for the application to reach a remote partner, it uses two commands to ask the Resolver what the IP address is for a host name, or vice versa. The commands are **`gethostbyname(xxx)`** and **`gethostbyaddress(nnn.nnn.nnn.nnn)`**. Figure 2-1 shows the information request and response flows as the Resolver gets a request, and based on its own configuration file will either look at a local hosts file or send a request to a DNS server. Once the relationship between the host name and IP address is made, the Resolver returns the response to the application.

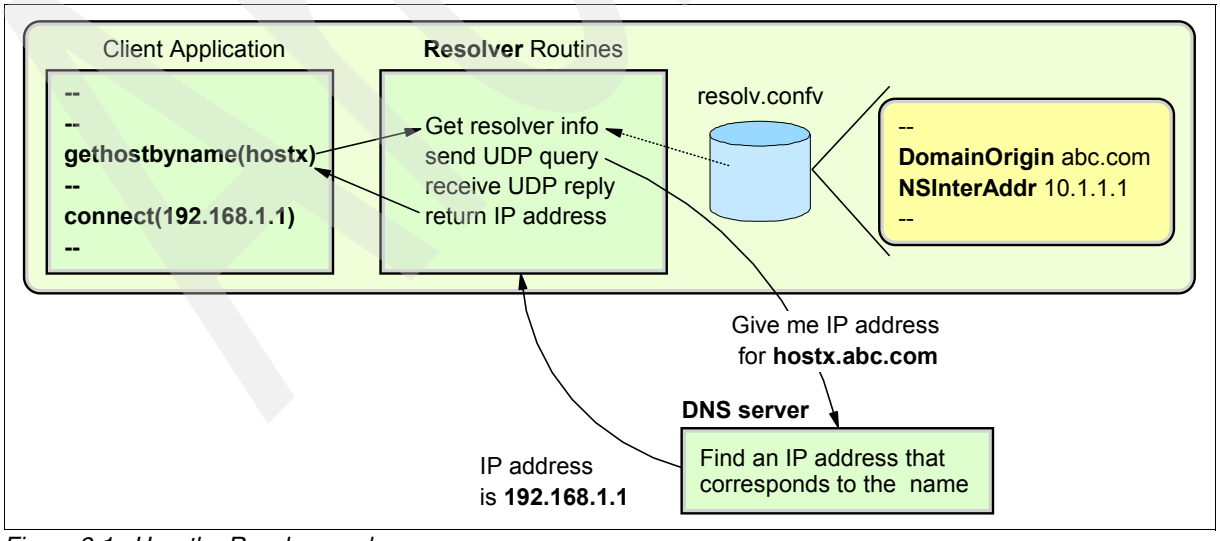


Figure 2-1 How the Resolver works

In a z/OS system, this task is more complex, because the applications can be built using one of three main groups of sockets API environments.

- Native TCP/IP sockets
- UNIX System Services callable sockets (BPX1xxxx calls)
- Language Environment® C/C++ sockets

To initiate a request to the Resolver in z/OS, an application executes a set of commands based on the Sockets API Library the application used to generate the socket to connect to the TCP/IP stack.

Each one of these environments interacts with the Resolver in a different way, which might change how the search for various Resolver configuration data sets are done, so if we open an application that uses LE C/C++ sockets programs it will include the resolv.conf (z/OS UNIX file system file) location in the search, while the native TCP/IP sockets programs will not. An application that uses USS sockets will use resolv.conf (z/OS UNIX file system file).

This flexibility allows an application to choose its own environment to work with, but also increases the complexity of the tasks needed to implement the right Resolver configuration in each environment.

To deal with this complexity a component called *Resolver address space* was introduced. The Resolver address space offers a potential for significant simplification of the Resolver configuration on z/OS.

The various Resolver libraries supported by the TCP/IP and LE APIs were consolidated into a single Resolver component. This allowed for consistent name resolution processing across all applications using the TCP/IP and LE socket APIs. The consolidated Resolver is automatically enabled on z/OS and runs in a separate address space that is automatically started during UNIX System Services initialization. This consolidated Resolver implements global parameters that can be configured to be used either as the first parameters to be used in a LPAR, or to be the last file to be used.

To implement these parameters, the Resolver procedure has a setup data set that contains the directives to define which files will be used as global files.

- GLOBALTCPIPDATA allows an installation to enforce a single Resolver configuration for the entire LPAR and fully under the administrator's control. The need for individual user-specific Resolver configurations is significantly reduced and in many cases eliminated when using GLOBALTCPIPDATA.
- DEFAULTTCPIPDATA also creates a Resolver configuration for the entire LPAR, but it is the last Resolver configuration to be used if the application does not define which Resolver it wants to use.

IPv6 changes to Resolver processing

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

- Resolver APIs were introduced for IPv6-enabled applications. Refer to “Name and address resolution functions” on page 20.
- DNS resource records are defined to represent hosts with IPv6 addresses, and therefore network flows between Resolvers and name servers (instead of DNS IPv4 A records).
- An 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 20 for more information.

Note: A much more comprehensive discussion of IPv6 support is in Chapter 6, “IPv6 support” on page 187.

Name and address resolution functions

These APIs allow applications to resolve host names to IP addresses and vice versa for IPv6. The primary APIs are getaddrinfo, getnameinfo, and freeaddrinfo. The APIs are designed to work with both IPv4 and IPv6 addressing. The use of these APIs should be considered if an application is being designed for eventual use in an IPv6 environment.

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

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 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 the entire list of addresses. These are the rules being applied:

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.

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.

Terminology: Deprecated, in this context, means that an IPv6 address has changed his state from *preferred* (the address has been leased to an interface for a fixed (possibly infinite) length of time) state to *deprecated*. (When a lifetime expires, the binding (and address) may become invalid and the address may be reassigned to another interface elsewhere on the Internet.) While in a deprecated state, the use of an address is discouraged, but not strictly forbidden.

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 the longest matching prefix. If one destination address has a longer CommonPrefixLength with its associated source address than the other destination address has with its source address, then the address with the longer CommonPrefixLength is placed before the other address.

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.

IPv6 Resolver statements

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

ETC.IPNODES is a 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. 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 search order is used for selecting ETC.IPNODES (local host files) for IPv6 searches:

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

The IPv6 search order is simplified, but to minimize migration concerns, the IPv4 search order continues to be supported as in previous releases. The side effect of this is that, by default, you 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 COMMONSEARCH statement in the Resolver setup data set. By specifying COMMONSEARCH, the user indicates that only the IPv6 search order should be used, regardless of whether the search is for IPv6 or IPv4 resources. This means that only one data set (ETC.IPNODES) has to be managed for the system, and that all the APIs utilize the same single data set. The use of COMMONSEARCH not only reduces IPv6 and IPv4 searching to a single search order, but also reduces the UNIX and native z/OS environments to a single search order as well.

To support the IPv6/IPv4 ipnodes file, the following statements are needed:

- 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 z/OS and 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.

Consolidating the Resolver address space environment

To consolidate this environment and to locate the Resolver configuration file each application wants to use, the Resolver address space uses a file search order where the first configuration file to be looked at will be the file defined by the directive GLOBALTCPIPDATA, and the last one the file defined by the directive DEFAULTTCPIPDATA. The remaining search order files are dependent of the sockets programs being used, as shown in Figure 2-2.

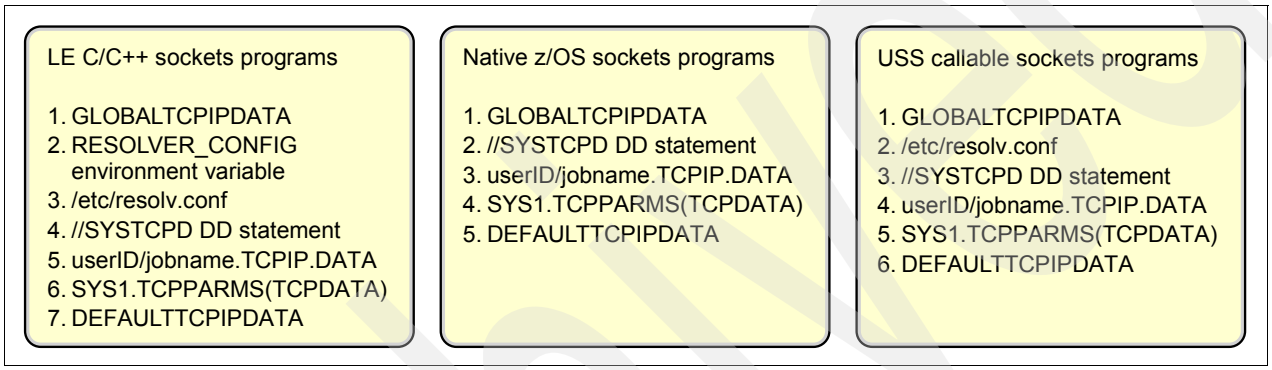


Figure 2-2 Search order for the Resolver configuration files

After the Resolver gets the right configuration file, the next thing to do is to define where to look for the name/address relationship. Inside the Resolver configuration file, it will look at the directives and, based on them, either look at local file or send a request to the defined name server in the network. The basic directives that should be defined are these:

- DOMAIN (equivalent to DOMAINORIGIN): Use this statement to specify the domain origin that is appended to the host name to form the fully qualified domain name of a host.
- HOSTNAME: Use this statement to specify the TCP host name of this z/OS Communications Server server.
- LOOKUP: Use this statement to specify the order in which the DNS or local host files are to be used for name resolution.
- NSINTERADDR (equivalent to NAMESERVER): Use this statement to define the IP address of a name server in dotted decimal format.
- TCPIPJOBNAME (equivalent to TCPIPUSERID): Use this statement to specify the member name of the procedure used to start the TCPIP address space. This parameter is used in a multiple stack environment.

The lookup statement defines if the Resolver address space will do the name resolution only in the local files, or using the defined name server or both in any specified order. If the local files are chosen, the Resolver has been enhanced to provide a new local file, called the ipnodes file, which can be used instead of the hosts file. It also provides a statement in the Resolver setup data set, COMMONSEARCH, used to indicate the search order for the local host files, as shown Figure 2-3 on page 23.

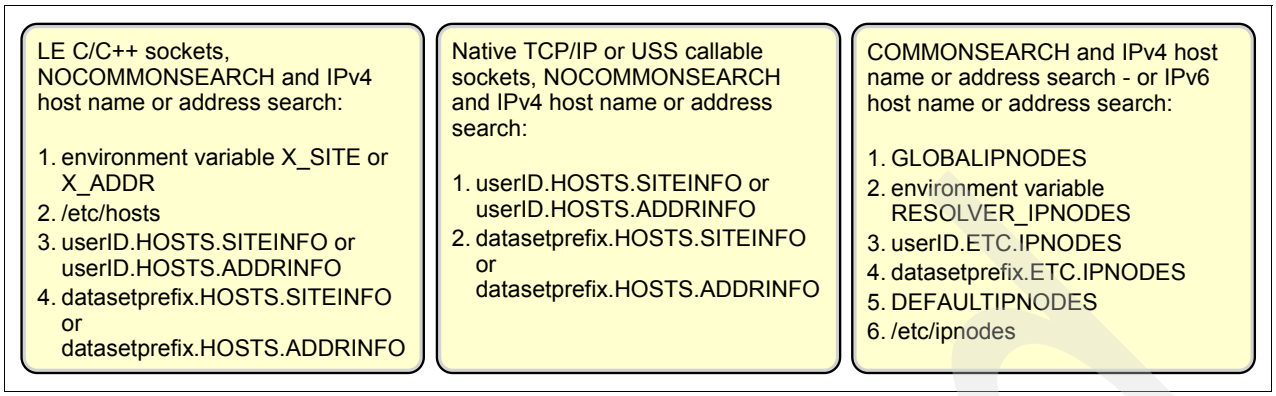


Figure 2-3 local host files search order using COMMONSEARCH

Once the correct Resolver configuration files and local files have been defined, the Resolver address space will execute the next step to find the relationship between the name and the IP address that has been requested by the application, which is based on what has been defined in the Resolver configuration file, send a query to the name server, or look at the local ipnode file to get the name/IP address being requested.

2.1.2 For additional information

For more specific information regarding the Resolver address space, refer to *z/OS V1R7.0 Communications Server: IP Configuration Guide*, SC31-8775.

2.2 Why the Resolver address space is important

The Resolver function allows applications to use names instead of IP addresses to connect to other partners. Although the use of an IP address might seem to provide an easy method to the connection, picture applications that must connect themselves with a larger number of partners, or an application that is accessed by thousands of clients. In these cases, the use of names is certainly a much easier and more reliable form for establishing access than IP addressing.

Another important reason to use names instead of IP addressing is that a user or an application is "insulated" from underlying network address changes.

To compare and decide which method would be more useful in your environment, refer to Table 2-1.

Table 2-1 Comparing the use of direct addressing with name resolution

	Hard-coded IP addresses	**Local hosts file**	**Domain Name System (DNS)**
Technology	None - Use the entered IP address directly on the connect() or sendto() socket call.	Use gethostbyname() and let the Resolver find an IP address in the locally configured hosts file.	Use gethostbyname() and let the Resolver contact the configured name server for an IP address.

	Hard-coded IP addresses	Local hosts file	Domain Name System (DNS)
Benefits	Fast (no name resolution). Good in some debugging situations (you know exactly which IP address is being used).	Fast (local name resolution).	IP address changes can be done without any local changes. All host names (in the entire network) can be resolved. A hierarchical name space.
Drawbacks	Difficult to remember IP addresses. Very inconvenient in case an IP address change occurs. Just think about IPv6.	If an IP addressing change is needed, all the local hosts files have to be updated. Only locally configured host names can be resolved.	Additional packets (requests) flow to resolve host name before destination can be reached.

2.3 The common design scenarios for the Resolver

When thinking about how to implement the Resolver in a z/OS environment, it is important to understand and review the environment variables and their purpose.

To demonstrate how this is done, we are going to show two basic scenarios:

- Using the Resolver address space in a single stack environment, where we are going to design a Resolver address space with specific global settings and a default Resolver config, and how it applies to a single stack environment.
- Using the Resolver address space in a multiple stack environment, setting up a Resolver global statement in a multiple stack environment, allowing applications to use an specific stack affinity or use all stacks to receive or send connection requests.

Recommendation: Although there are specialized cases where multiple stacks per LPAR can provide value, we in general recommend implementing only one TCP/IP stack per LPAR. The reasons for this recommendation are as follows:

- A TCP/IP stack is capable of exploiting all available resources defined to the LPAR in which it is running. Therefore, starting multiple stacks will not yield any increase in throughput.
- When running multiple TCP/IP stacks, additional system resources, such as memory, CPU cycles, and storage, are required.
- Multiple TCP/IP stacks add a significant level of complexity to TCP/IP system administration tasks.
- It is not necessary to start multiple stacks to support multiple instances of an application on a given port number, such as a test HTTP server on port 80 and a production HTTP server also on port 80. This type of support can instead be implemented using BIND-specific support where the two HTTP server instances are each associated with port 80 with their own IP address, via the BIND option on the PORT reservation statement.

One example where multiple stacks can have value is when an LPAR needs to be connected to multiple isolated security zones in such a way that there is no network level connectivity between the security zones. In this case, a TCP/IP stack per security zone can be used to provide that level of isolation, without any network connectivity between the stacks.

2.3.1 Using the Resolver address space in a single stack environment

To set up an environment with global definitions, it is important to understand how these definitions affect the entire environment, and also which statements must always be defined once we create this global environment.

To implement the global settings, we use a setup data set that has parameters to address the Resolver config file to be used as global or as a default file; so in our first scenario, we use the following steps to include the global definitions in our system:

1. Describe the environment that is going to be created. In this step, we define which statements we want to be defined as global parameters, and which ones we want to define as default, if any applications do not have it defined.
2. Define the names and location of the global and default Resolver configuration files.
3. Define how the Resolver procedure will be started.

Describe the environment

In this step, the global TCPIP.DATA data set will become the first TCPIP.DATA data set read regardless of the Socket API library being used. Any parameters found in this data set will be global settings for the z/OS LPAR. If a global TCPIP.DATA data set has been specified then all Resolver statements defined in it will only be obtained from this data set.

The search continues beyond the file specified by GLOBALTCPIPDATA, but any of the Resolver statements specified in files lower in the search order will only be used if not specified explicitly in the global statements.

In our scenario we want our system to have a common set of parameters that will not be changed by any application. We also want all queries to be searched on the local files first using the ETC.IPNODES data set. If it is not there, then query a name server. Figure 2-4 on page 26 shows the relationship between the Resolver procedure, which defines the setup data set, rssetup, where we will find the pointer to the Global Resolver configuration, called GLOBAL, and the global ipnodes, called IPNODES.

We also defined the global ipnodes in the setup data set so all queries will address this data set locally, and defined the statement lookup in the global definitions to search the local files first and, if it is not there, then query the DNS.

Figure 2-4 Resolver definitions with global setup

Using local settings in a single stack environment

After we implement the global settings, we might want to allow an application to use its own Resolver configuration file (local settings). This can be done for all statements that are not already defined in the GLOBALTCPIPDATA data set.

The setup of these statements is done for each application, and to accomplish this, these applications must include in their start procedures a pointer to a Resolver configuration file where their local settings are defined, as shown in Figure 2-5 on page 27. Each directive defined to RESOLVER_CONFIG will only be used if it is not defined in the GLOBALTCPIPDATA data set.

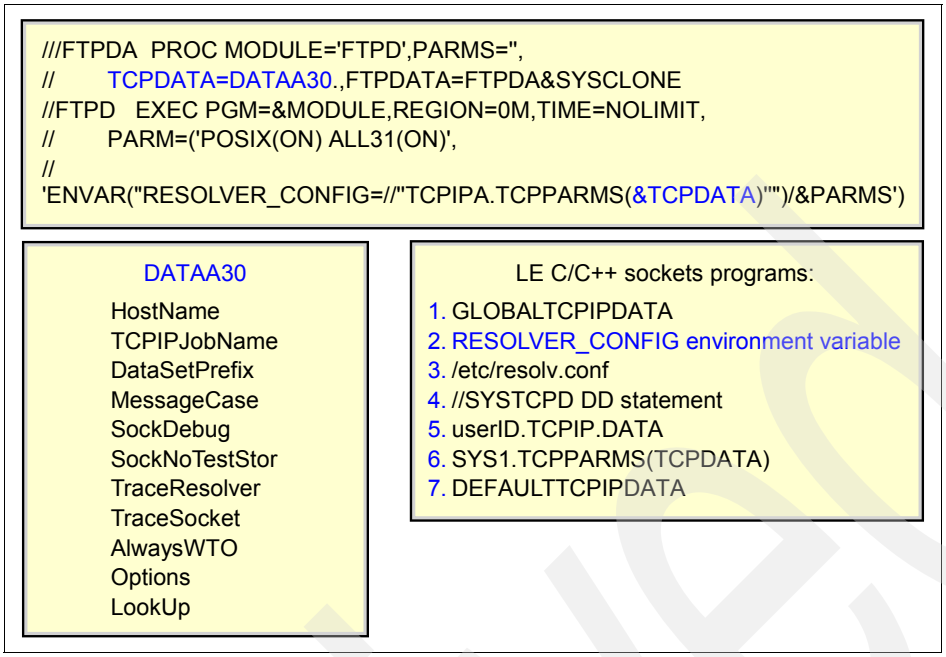


Figure 2-5 Local Resolver definitions

Advantages

The advantages of setting up a Resolver using global and default statements in a single stack environment are that it allows the administrator to retain control of which Resolver statements are used for name resolution, and to eliminate the complexity of attempting to merge Resolver statements from multiple files or data sets in a predictable and useful manner.

The main advantage of using local configuration files for each application is the relative independence that allows the application owner to use specific parameters only related to his own environment.

Considerations

One thing that must be taken into consideration when using a GLOBALTCPIPDATA file is that some statements must be coded or defaults will be used by the Resolver address space. If you try to define any of these statements in any of the following TCP/IP DATA data sets, they will not be used:

- Domain or DomainOrigin
- NameServer or NSInterAddr
- NSPortAddr
- ResolverTimeOut
- ResolverUDPRetries
- ResolveVia
- Search
- Sortlist

When setting up a local Resolver configuration defining statements that are meant to be used only by the application using these statements, we must be sure to point to a specific TCPIP.DATA data set.

Important: In some Resolver environments, the use of the trace functions (such as SockDebug or TraceResolver) may affect performance. Therefore, we recommend using the method described in “CTRACE - RESOLVER (SYSTCPRE)” on page 46.

When considering allowing applications to use their own set of statements, it is important to set up a DEFAULTTCPIPDATA file, so the environment will provide the application owners with the possibility of using their own settings, while it will maintain the basic settings defined for those who do not want to define their own definitions.

2.3.2 Using the Resolver address space in a multiple stack environment

When implementing a Resolver address space with global settings in a multiple stack environment, we must be very careful. The Resolver address space is a single process within the z/OS LPAR, which means the global environment (GLOBALTCPIPDATA) statements will have an impact on all stacks within the z/OS LPAR.

What should be taken into consideration when deciding whether to use GLOBALTCPIPDATA is to verify that all stacks will use common Resolver statements, such as NameServer and domainorigin. If they cannot use common Resolver statements, then you must use distinct TCPIP.DATA data sets for setting up a DEFAULTTCPIPDATA file to define a default TCP/IP stack for applications that do not define their own Resolver statements.

In our scenario, we will implement a global environment defining a common set of statements. This will allow each stack to use its own TCPIP.DATA data set to define the local environment and will create a default TCPIP.DATA defining stack TCPIPA as the default TCPIPJOBNAME statement (see Figure 2-6).

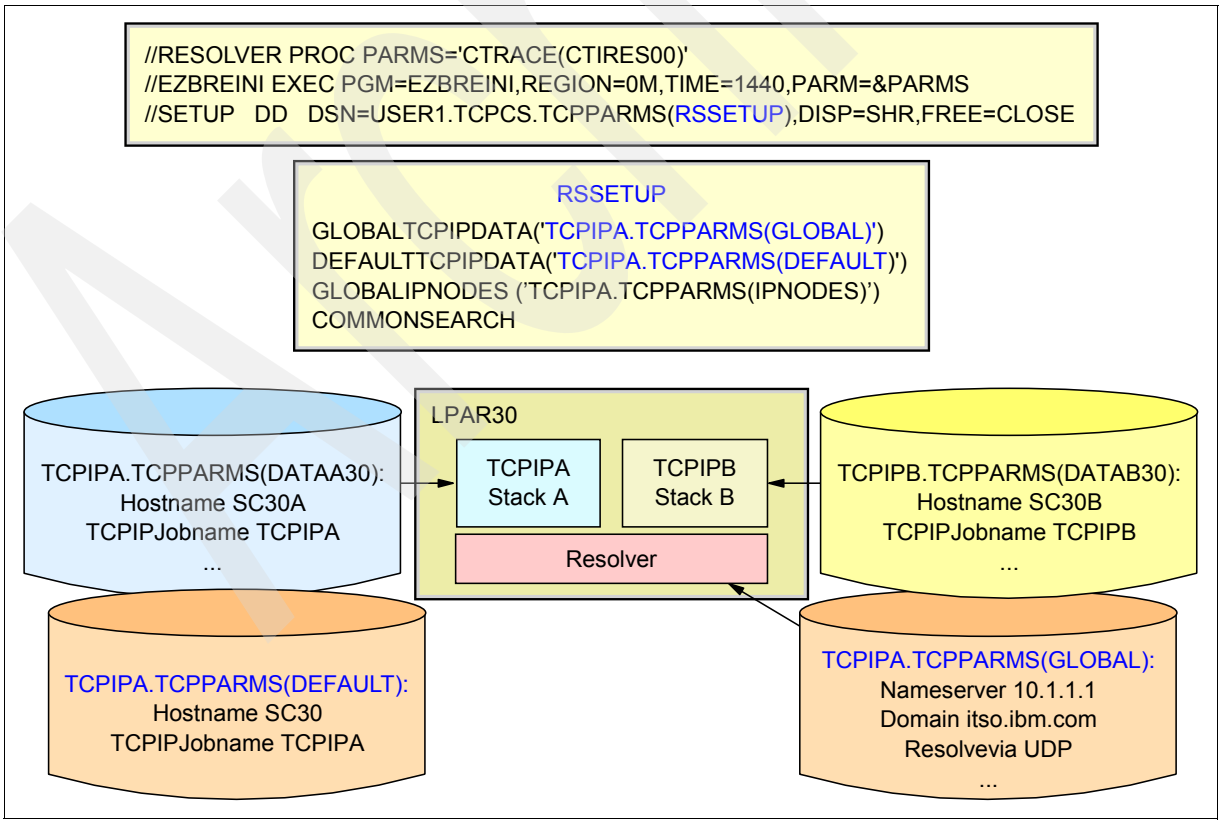


Figure 2-6 Multiple stacks with global Resolver environment

Using local settings in a multiple stack environment

When designing a multiple stack scenario, it is important to check each application that will be used and how it will be implemented in the environment. Each type of application has its own relationship with a TCP/IP stack, for example:

- A native TCP/IP sockets program will always use one stack only—by default the stack that is identified in the TCPIPJOBNAME option in the chosen Resolver configuration file. However, the stack can also be chosen via the program configuration and API calls to associate the program with a chosen stack, as shown in Figure 2-7.

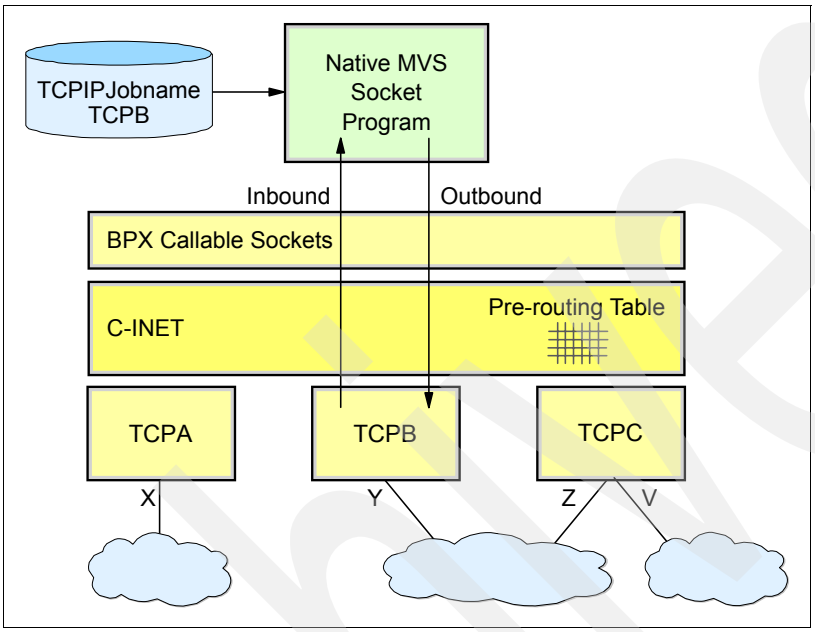


Figure 2-7 Native TCP/IP applications in a multiple stack environment

- Applications using UNIX System Server callable APIs or LE C/C++ sockets APIs can be implemented using a generic bind to open the same port in all TCP/IP stacks. By doing so, the application will accept incoming connections or UDP datagrams over any interface of all connected stacks, as shown in Figure 2-8 on page 30.

  Outbound connections or UDP datagrams are processed by the C-INET pre-router, and the stack with the best route to the destination is chosen.

  When using a generic bind, the server port number must be reserved in all stacks. If one has it reserved to another address space, the bind() call fails.

  When using the generic bind, it does not matter if the chosen Resolver configuration file has a TCPIPJOBNAME; it is not used when the server is a pure generic server.

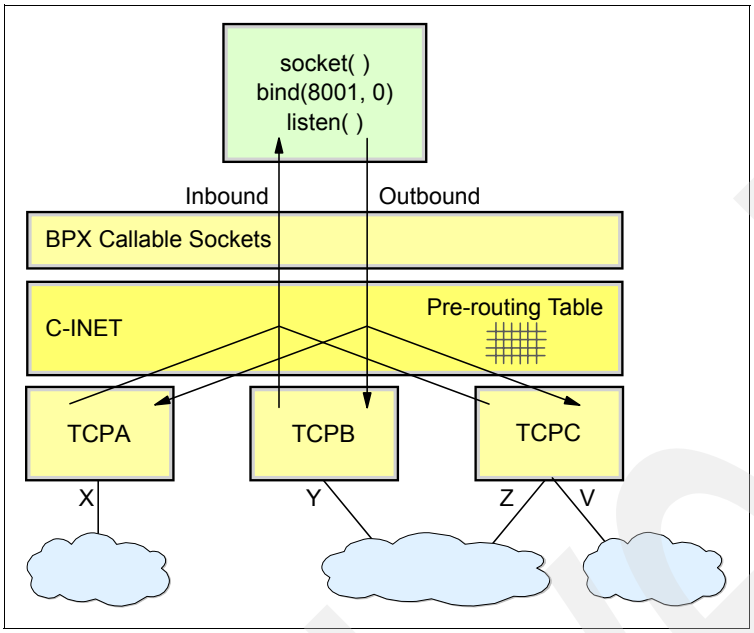


Figure 2-8 UNIX System Server callable APIs or LE C/C++ sockets APIs using Generic bind in a multiple stack environment

- Applications using UNIX System Server callable APIs or LE C/C++ sockets APIs can also use a specific bind to open a socket. A bind-specific server socket will only receive connections from the stack that owns the IP address to which the socket is bound. Outbound connections or UDP datagrams will be handled by the stack that offers the best route to the destination IP address, as shown in Figure 2-9.

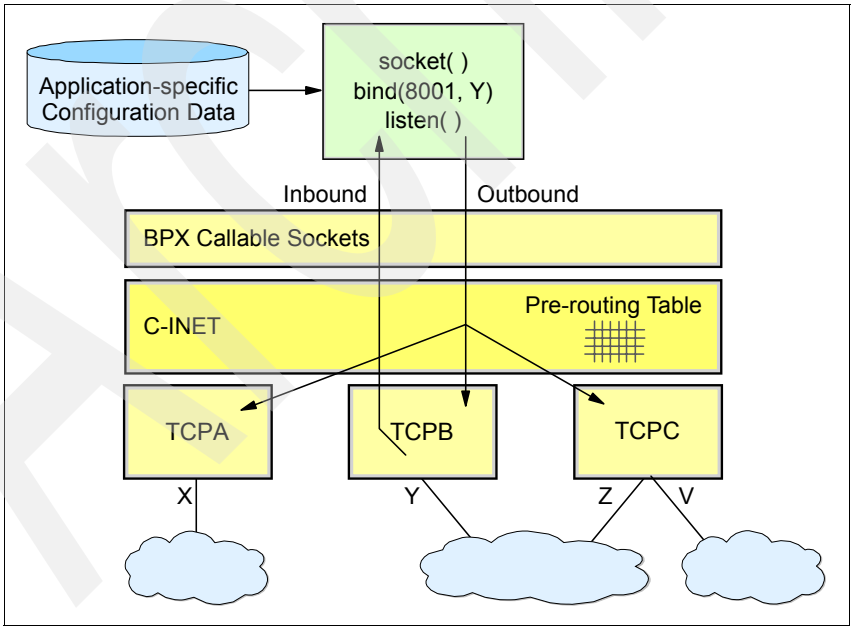


Figure 2-9 UNIX System Server callable APIs or LE C/C++ sockets APIs using specific bind in a multiple stack environment

Advantages

When using a multiple stack environment, setting up a global environment with a minimum set of common statements has the advantage of preserving the entire image from being changed inadvertently. It has also the advantage of creating a default environment that could be used for those who are not planning to create their own Resolver definitions.

Considerations

In a multiple stack environment, it is important to review how the applications will be implemented. Some of these applications might be implemented to open their socket with a specific bind and to do this they will need to code TCPIPJOBNAME to direct them to the different stacks. Therefore, when designing the global definitions in the Resolver address space, do not code a TCPIPJOBNAME in GLOBALTCPIPDATA, but allow it to be coded in local TCPIP.DATA.

2.3.3 Recommendations

To implement the Resolver address space, it is important to first determine whether your environment will require a single TCP/IP stack or multiple TCP/IP stacks.

In both cases the Resolver is an independent address space and has to be up and running before the TCP/IP stack is started. However, when implementing for a multiple stack environment, you must determine how the global files will be defined and which global statements will be used. This will depend on how independent each stack will be within the z/OS LPAR. The recommendation is to create a global TCPIP.DATA data set for a single stack environment, and a default TCPIP.DATA data set if a multiple stack environment is required.

In the next session we will configure the Resolver address space to create the global and default definitions to provide the same environment for all applications, which do not have specific needs. This will allow specific local statements to be defined at the stack level or application level. To be able to do that, each stack (if more than one is desired) or each application that needs to have its own resolver statements, has to define to its own TCPIP.DATA data set for local statements.

It is also recommended to start the Resolver address space during the initialization of the UNIX System Services via the RESOLVER_PROC() statement within BPXPRMxx, as we will show in the next section.

2.4 How the Resolver address space is implemented

Based upon our recommendations, we will show the implementation details for the Resolver address space.

2.4.1 Implementing the Resolver address space

In this section we are going to show the necessary tasks to implement and configure the Resolver address space in our test environment. Figure 2-10 on page 32 shows the data sets that are need to configure the Resolver address space.

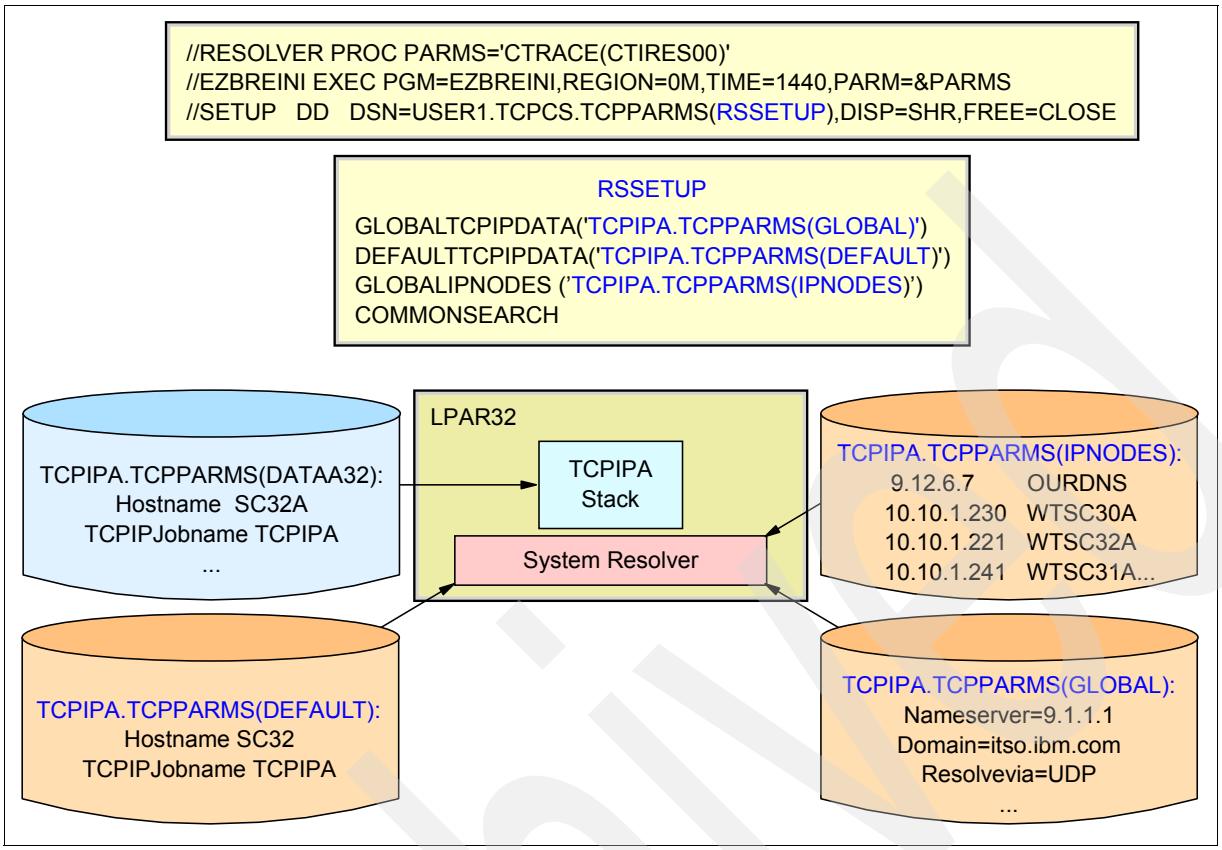


Figure 2-10 Resolver data sets

Implementation tasks

To implement the Resolver address space in our test environment we followed these steps:

1. Set up the Resolver procedure.
2. Configure the setup data set.
3. Configure the global TCPIP.DATA data set statements.
4. Configure the default TCPIP.DATA data set statements.
5. Create the ETC.IPNODES data set.

Set up the Resolver procedure

In this step, we prepare the resolver procedure to be started during the UNIX System Services initialization as recommended. To do so, we used the BPXPRMxx statement, RESOLVER_PROC, to specify the procedure name we want to use to start the Resolver address space. Figure 2-11 on page 33 show the partial contents of BPXPRMxx (member of SYS1.PARMLIB).

```
/*********************************************************************/
/* RESOLVER_PROC is used to specify how the resolver address space  */
/* is processed during Unix System Services initialization.         */
/* The resolver address space is used by Tcp/Ip applications        */
/* for name-to-address or address-to-name resolution.               */
/* In order to create a resolver address space, a system must be    */
/* configured with an AF_INET or AF_INET6 domain.                   */
/*  RESOLVER_PROC(procname|DEFAULT|NONE)                            */
/*     procname - The name of the address space for the resolver.   */
/*                In this case, this is the name of the address     */
/*                space as well as the procedure member name        */
/*                in SYS1.PROCLIB. procname is 1 to 8 characters    */
/*                long.                                             */
/*     DEFAULT -  An address space with the name RESOLVER will      */
/*                be started. This is the same result that will     */
/*                occur if the RESOLVER_PROC statement is not       */
/*                specified in the BPXPRMxx profile.                */
/*                                                                  */
/*     NONE    -  Specifies that a RESOLVER address space is        */
/*                not to be started.                                */
/*                                                          @DAA*/
/*********************************************************************/
RESOLVER_PROC(RESOLV32)
```

Figure 2-11 Specifying the Resolver procedure to be started

After we defined the procedure that will be started during the UNIX System Services initialization, the next step is to create and define the Resolver procedure, which in our scenario is called RESOLV32.

To create the procedure, we copied the sample procedure, EZBREPRC (alias RESOPROC), located in SEZAINST, and customized it to our environment, as shown in Figure 2-12. The procedure has only one DD card that has to be configured, the SETUP DD card **1**, which describes where the setup data set is located.

```
//RESOLV32 PROC PARMS='CTRACE(CTIRES00)'
//*
//*   IBM Communications Server for z/OS
//*   SMP/E distribution name: EZBREPRC
//*
//*   5694-A01 (C) Copyright IBM Corp. 2001, 2005.
//*   Licensed Materials - Property of IBM
//*
//*   Function: Start Resolver
//EZBREINI EXEC PGM=EZBREINI,REGION=0M,TIME=1440,PARM=&PARMS
//*   SETUP contains Resolver setup parameters.
//*     See the section on "Understanding Resolvers" in the
//*     IP Configuration Guide for more information. A sample of
//*     Resolver setup parameters is included in member RESSETUP
//*     of the SEZAINST data set.
//*
//*SETUP   DD   DSN=TCPIP.SETUP.RESOLVER,DISP=SHR,FREE=CLOSE
//*SETUP   DD   PATH='/etc/setup.resolver',PATHOPTS=(ORDONLY)
//SETUP   DD   DSN=TCPIPA.TCPPARMS(SETUP),DISP=SHR,FREE=CLOSE 1
```

Figure 2-12 Resolver procedure

Important: When the Resolver is started by UNIX System Services, you must pay attention to the following:

- The Resolver address space is started with SUB=MSTR. This means that JES services are not available to the Resolver address space. Therefore, no DD cards with SYSOUT can be used.
- The Resolver start procedure needs to reside in a data set that is specified by the MSTJCLxx PARMLIB member's IEFPDSI DD card specification. If not, the procedure will not be found and the Resolver will not start. SYS1.PROCLIB is usually one of the libraries specified there.

Configure the setup data set

The next step to implement the resolver address space is to configure our setup data set. This data set defines the location of the global and default data sets containing the parameters we want to be defined in the z/OS environment.

In our test environment, we copy the setup sample data set RESSETUP from SEZAINST and change its contents to meet our requirements, as shown in Figure 2-13.

```
;   IBM Communications Server for z/OS
;   SMP/E distribution name: EZBRECNF
;   5694-A01 (C) Copyright IBM Corp. 2002, 2005,
;   Licensed Materials - Property of IBM
;   Function: Sample Resolver setup file
;
DEFAULTTCPIPDATA('TCPIPA.TCPPARMS(DEFAULT)') 1
;
GLOBALTCPIPDATA('TCPIPA.TCPPARMS(GLOBAL)') 2
;
GLOBALIPNODES('TCPIPA.TCPPARMS(IPNODES)') 3
;
; DEFAULTIPNODES('TCPCS.SYS.TCPPARMS(IPNODES)') 4
;
; DEFAULTIPNODES('TCPCS.ETC.IPNODES')
;
; DEFAULTIPNODES(/etc/ipnodes)
;
; NOCOMMONSEARCH
;
COMMONSEARCH 5
```

Figure 2-13 Resolver address space SETUP data set

Provided is a brief explanation of each statement:

- 1 This statement defines the final search location for TCPIP.DATA statements. It will replace TCPIP.TCPIP.DATA. It may be an z/OS data set or z/OS UNIX file system file.
- 2 This statement defines the first search location for TCPIP.DATA statements. It may be an z/OS data set or z/OS UNIX file system file.
- 3 This statement defines the first search location for IPNODES statements. It may be an z/OS data set or z/OS UNIX file system file.
- 4 This statement defines the final search location for IPNODES statements. It may be an z/OS data set or z/OS UNIX file system file.

- 5 This statement defines if the common search order in which the local files should be used or not.

Attention: Plan carefully to create these global parameters. They affect the entire z/OS Communications Server TCP/IP implementation.

Configure the global TCPIP.DATA data set statements

In this step we provide the global statements that all stacks (if more than one is created) and applications will use in our z/OS environment. To define these statements we copied the sample TCPIP.DATA data set provided in SEZAINST, called TCPDATA, and changed the statements to achieve our desired configuration, as shown in Figure 2-14.

```
;DOMAINORIGIN  ITSO.IBM.COM     1
SEARCH  ITSO.IBM.COM IBM.COM    1 2
NSINTERADDR  10.12.6.7            1
NSPORTADDR 53                   1
RESOLVEVIA UDP                  1
RESOLVERTIMEOUT 10              1
RESOLVERUDPRETRIES 1            1
LOOKUP LOCAL DNS                3
```

Figure 2-14 Global TCP.DATA data set

Important: If GLOBALTCPIPDATA is specified:

- Any TCPIP.DATA statements contained in it will take precedence over any TCPIP.DATA statements found by way of the appropriate environment's (Native z/OS or z/OS UNIX) search order.
- The TCPIP.DATA statements in Figure 2-14 marked with 1 can only be specified in GLOBALTCPIPDATA. If the resolver statements are found in any of the other search locations for TCPIP.DATA they are ignored. If the resolver statements are not found in GLOBALTCPIPDATA, their default value will be used.

The SEARCH statement (2) is used instead of DOMAINORIGIN because it allows us to specify a list of domains to be appended to the host name when passed to the Resolver. The first domain in the list is defined as the DOMAINORIGIN.

We defined the LOOKUP (3) statement in the global TCPIP.DATA data set, because we want the name resolution to be attempted using the local files first.

Configure the default TCPIP.DATA data set statements

The next step in our Resolver address space implementation was to create the default TCPIP.DATA data set pointed to by statement DEFAULTTCPIPDATA in our SETUP data set, as shown in Figure 2-13 on page 34, marked by (1).

In this data set we included the statements that have not been already defined in the Global TCPIP.DATA data set. In our scenario, we want to define a default TCP/IP job name and a default host name, as shown in Figure 2-15.

```
TCPIPJOBNAME TCPIP 1
HOSTNAME WTSC30 2
```

Figure 2-15 Default TCPIP.DATA data set contents

TCPIPJOBNAME (1)specifies the name of the started procedure that is being used to start the TCP/IP address space.

Attention: Applications that use Language Environment services without a TCPIPJOBNAME statement cause applications that issue __iptcpn() to receive a jobname of NULL, and some of these applications will use INET instead of TCP/IP. Although this presents no problem when running in a single-stack environment, this can potentially cause errors in a multi-stack environment.

HOSTNAME (2) specifies the TCP host name of this system as it is known in the IP network.

Create the ETC.IPNODES data set

The last step of our implementation was to create our ETC.IPNODES data set, which will have our local hosts name-to-address relationship and vice versa. The data set contains the Internet protocol (IP) host names and addresses for the local host and other hosts in the network. This data set is used to resolve a name into an IP address (that is, to translate) or resolve an IP address into a name.

We chose to use the COMMONSEARCH, because it would allow us to have a common local search environment with IPv4 or IPv6 hosts. Figure 2-16 shows the contents of the ETC.IPNODES data set. When using COMMONSEARCH, only the IPNODES data set is used.

```
;  Function: Sample ETC.IPNODES file
;
;  Entries in the hosts file have the following format:
;
;  Address HostName
;
;  Address HostName1 HostName2 HostName3 ..... HostName35
;
;  Address: is an IP address, it can be IPV4 or IPV6 address.
;           Note: IPv4-mapped IPv6 address is not allowed.
;  HostName: the length of the hostname is up to 128 characters,
;            and each IP address can have up to 35 hostnames.
;
   10.12.6.7    OURDNS
   10.10.1.230 WTSC30A
   10.10.1.221 WTSC32A
   10.10.1.241 WTSC31A
   10.10.2.1   router1
   10.10.3.2   router2
   1::2 TESTIPV6ADDRESS1
   1:2:3:4:5:6:7:8 TESTIPV6ADDRESS2
```

Figure 2-16 ETC.IPNODES data set contents

Managing the Resolver address space

As we mentioned earlier, use a BPXPRMxx statement, RESOLVER_PROC, to specify the procedure name and to start the Resolver address space. If the RESOLVER_PROC statement is not in the BPXPRMxx parmlib member or is specified with a procedure name of DEFAULT, z/OS UNIX will start a Resolver address space with the assigned name of

RESOLVER. This name is used with the following z/OS system commands to manage the Resolver address space:

- STOP (P) - Use the STOP command if you must stop and restart the Resolver Address space to update the Resolver code, change the procedure, or there is a problem that demands that you recycle the Resolver address space. To stop the default Resolver use the following command:

  ```
  STOP RESOLVER
  ```

  To stop a customized Resolver address space simply use the procedure name defined in the RESOLVER_PROC statement instead of RESOLVER.

- START (S) - Use this command to restart the Resolver address space if needed. Use the START command to start the default procedure, as follows:

  ```
  START IEESYSAS.RESOLVER,PROG=EZBREINI,SUB=MSTR
  ```

  If you have customized the Resolver address space, issue the following command:

  ```
  START RESOLV32,SUB=MSTR
  ```

  Important: Stopping and restarting of the resolver should only be used if a new level of the resolver code has been installed.

- FORCE - Use the FORCE command to kill the process in case of a hang.
- MODIFY (F) - Use the MODIFY command to dynamically change resolver setup statements, update the resolver's usage of TCPIP.DATA statements, or update the resolver's usage of local host and services tables.

Configuration examples

In this section we show the resulting messages when starting, stopping, or modifying the Resolver address space, using the procedure we created in “Set up the Resolver procedure” on page 32.

To implement our Resolver address space, we first stopped the running Resolver using the STOP command, as shown in Example 2-1.

Example 2-1 Stopping the Resolver address space

```
STOP RESOLVER
EZZ9292I RESOLVER ENDING
IEF196I IEF142I IEESYSAS RESOLVER - STEP WAS EXECUTED - COND CODE 0000
IEF196I IEF373I STEP/IEFPROC /START 2005270.1315
IEF196I IEF374I STEP/IEFPROC /STOP 2005285.1723 CPU    0MIN 00.02SEC
IEF196I SRB    0MIN 00.12SEC VIRT    44K SYS   176K EXT     220K SYS
IEF196I 26984K
IEF196I IEF375I JOB/RESOLVER/START 2005270.1315
IEF196I IEF376I JOB/RESOLVER/STOP  2005285.1723 CPU    0MIN 00.02SEC
IEF196I SRB    0MIN 00.12SEC
IEF352I ADDRESS SPACE UNAVAILABLE
IEF196I IEF352I ADDRESS SPACE UNAVAILABLE
```

Next, we restarted the Resolver address space using our procedure, RESOLV32, as shown in Example 2-2.

Example 2-2 Starting a configured Resolver address space

```
S RESOLV32,SUB=MSTR
IRR812I PROFILE ** (G) IN THE STARTED CLASS WAS USED 366
```

```
        TO START RESOLV32 WITH JOBNAME RESOLV32.
EZZ9298I DEFAULTTCPIPDATA - TCPIPA.TCPPARMS(DEFAULT)
EZZ9298I GLOBALTCPIPDATA - TCPIPA.TCPPARMS(GLOBAL)
EZZ9298I DEFAULTIPNODES - None
EZZ9298I GLOBALIPNODES - TCPIPA.TCPPARMS(IPNODES)
EZZ9304I COMMONSEARCH
EZZ9291I RESOLVER INITIALIZATION COMPLETE
$HASP395 BPXAS    ENDED
```

Note: In Example 2-2, during the startup process, we are able to see what was defined in the setup data set.

It is possible to change the Resolver address space configuration by changing the SETUP data set contents and using the MODIFY command and pointing to the changed SETUP data set to refresh Resolver. To show how this is done, we created a new SETUP data set, named NEWSETUP, without the global definitions, then used the MODIFY command to refresh the Resolver to reflect the changes, as shown in Example 2-3.

Example 2-3 Modifying the Resolver address space

```
F RESOLV32,REFRESH,SETUP=TCPIPA.TCPPARMS(NEWSETUP)
IEF196I IEF237I 803B ALLOCATED TO SYS00009
IEF196I IEF285I   TCPIPA.TCPPARMS                              KEPT
IEF196I IEF285I   VOL SER NOS= COMCAT.
EZZ9298I DEFAULTTCPIPDATA - TCPIPA.TCPPARMS(DEFAULT)
EZZ9298I GLOBALTCPIPDATA - None
EZZ9298I DEFAULTIPNODES - TCPIPA.TCPPARMS(IPNODES)
EZZ9298I GLOBALIPNODES - None
EZZ9304I COMMONSEARCH
EZZ9293I REFRESH COMMAND PROCESSED
```

2.4.2 Verification

After our new Resolver address space was up and running, we wanted to verify it was working as expected. The verification steps we used are:

1. Display the Resolver address space setup configuration.
2. Check if the Resolver is establishing the correct name-to-address relationship by executing a **ping** command.

To display the Resolver address space configuration, use the MODIFY command with the display option, as shown in Example 2-12 on page 42.

Example 2-4 Modify Resolver with display option

```
F RESOLV32,DISPLAY
EZZ9298I DEFAULTTCPIPDATA - TCPIPA.TCPPARMS(DEFAULT)
EZZ9298I GLOBALTCPIPDATA - TCPIPA.TCPPARMS(GLOBAL)
EZZ9298I DEFAULTIPNODES - None
EZZ9298I GLOBALIPNODES - TCPIPA.TCPPARMS(IPNODES)
EZZ9304I COMMONSEARCH
EZZ9293I DISPLAY COMMAND PROCESSED
```

The next step is to check if the Resolver is able to execute the expected name-to-address relationship using the **ping** command, as shown in Example 2-5 on page 39. We were able to see that the name *router1* was resolved to address 10.10.2.1.

Example 2-5 Ping command results

```
CS03 @ SC32:/u/cs03>ping router1
CS V1R7: Pinging host router1 (10.10.2.1)
Ping #1 response took 0.000 seconds.
```

It is also possible to verify where the resolver is looking by using the TRACE RESOLVER parameter in the stack's or application's TCPIP.DATA data set. How this is done and the contents of this trace will be explained in "Diagnosing the Resolver address space environment" on page 43.

2.4.3 Implementing local settings in the Resolver environment

Once our Resolver address space is activated, all our global TCPIP.DATA statements will be in place and cannot be overrided unless we change the setup data set and execute a MODIFY command to use a new set of statements.

The TCP/IP stack or application is still able to use some local TCPIP.DATA statements to suit its own requirements, such as the HOSTNAME and TCPIPJOBNAME. In this section we start an FTP server (FTPDA), which establishes affinity with stack TCPIPA, and uses the TCPIP.DATA data set (DATAA32) to create its own statements. We also create a user local IPNODES data set for TSO user CS03, which will take precedence over the stack's IPNODES data set and over the default IPNODES data set pointed to by the Resolver address space setup data set, as shown in Figure 2-17 on page 40.

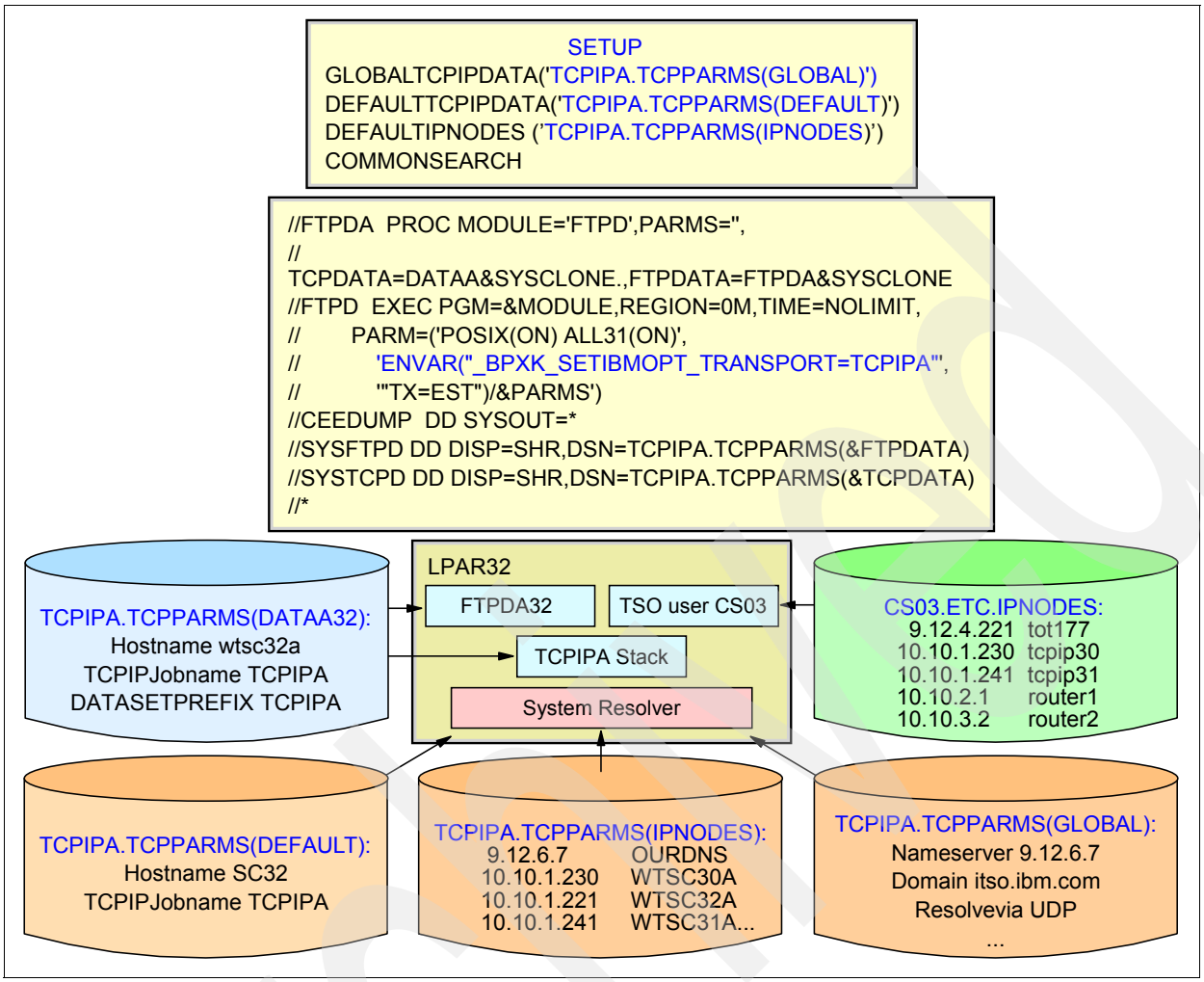


Figure 2-17 Local Resolver configuration

Implementation tasks

To implement these local changes in our Resolver environment we did the following:

- Created a TCPIP.DATA data set named DATAA32 with the TCPIPA stack's statements, as shown in Example 2-6.

Example 2-6 TCPIP.DATA data set DATAA32

```
;OPTIONS DEBUG
TCPIPJOBNAME TCPIPA
HOSTNAME WTSC32A
DOMAINORIGIN  ITSO.IBM.COM
DATASETPREFIX TCPIPA
MESSAGECASE MIXED
```

- Customized the TCPIPA stack procedure (RESOLVER_CONFIG) to point to DATAA32, using the &sysclone variable to simplify our implementation to allow for a single procedure to be used by any z/OS image in our sysplex environment, as shown in Example 2-7.

Example 2-7 TCPIPA procedure

```
//TCPIPA    PROC PARMS='CTRACE(CTIEZB00),IDS=00',
//             PROFILE=PROFA&SYSCLONE.,TCPDATA=DATAA&SYSCLONE
```

```
//TCPIPA    EXEC PGM=EZBTCPIP,REGION=0M,TIME=1440,
//             PARM=('&PARMS',
//         'ENVAR("RESOLVER_CONFIG=//''TCPIPA.TCPPARMS(&TCPDATA)''")')
//SYSPRINT DD SYSOUT=*,DCB=(RECFM=VB,LRECL=132,BLKSIZE=136)
//ALGPRINT DD SYSOUT=*,DCB=(RECFM=VB,LRECL=132,BLKSIZE=136)
//CFGPRINT DD SYSOUT=*,DCB=(RECFM=VB,LRECL=132,BLKSIZE=136)
//SYSOUT   DD SYSOUT=*,DCB=(RECFM=VB,LRECL=132,BLKSIZE=136)
//CEEDUMP  DD SYSOUT=*,DCB=(RECFM=VB,LRECL=132,BLKSIZE=136)
//SYSERROR DD SYSOUT=*
//PROFILE  DD DISP=SHR,DSN=TCPIPA.TCPPARMS(&PROFILE.)
```

- Customized a FTPDA procedure to establish affinity with the TCPIPA stack and to use the TCPIPDATA data set (DATAA32), as shown Example 2-8.

Example 2-8 FTPDA procedure

```
//FTPDA  PROC MODULE='FTPD',PARMS='',
//        TCPDATA=DATAA&SYSCLONE.,FTPDATA=FTPDA&SYSCLONE
//FTPD   EXEC PGM=&MODULE,REGION=0M,TIME=NOLIMIT,
//        PARM=('POSIX(ON) ALL31(ON)',
//            'ENVAR("_BPXK_SETIBMOPT_TRANSPORT=TCPIPA"',
//            '"TX=EST")/&PARMS')
//CEEDUMP  DD SYSOUT=*
//SYSFTPD DD DISP=SHR,DSN=TCPIPA.TCPPARMS(&FTPDATA)
//SYSTCPD DD DISP=SHR,DSN=TCPIPA.TCPPARMS(&TCPDATA)
//*
```

- After establishing the affinity of FTPDA to work only with stack TCPIPA, we reserved ports 21 and 20 for the FTP server in PROFILE.TCPIP, as shown in Example 2-9.

Example 2-9 Reserving ports 20 and 21 to FTPDA

```
20 TCP omvs  NOAUTOLOG     ; FTP Server
21 TCP ftpda1  BIND 10.10.1.221; control port
```

- Created a TCPIPA.ETC.IPNODES data set to define the host names for stack TCPIPA, which takes precedence over the default IPNODES data set when using this stack, as shown in Example 2-10.

Example 2-10 TCPIPA.IPNODES data set

```
;  Entries in the hosts file have the following format:
;  Address HostName
;  Address HostName1 HostName2 HostName3 ..... HostName35
;  Address: is an IP address, it can be IPV4 or IPV6 address.
;           Note: IPv4-mapped IPv6 address is not allowed.
;  HostName: the length of the hostname is up to 128 characters,
;            and each IP address can have up to 35 hostnames.
10.12.4.221  wscs01
   10.10.1.230 tcpipa30
   10.10.1.221 tcpipa32
   10.10.1.241 tcpipa31
   10.10.2.1   router1
   10.10.3.2   router2
   1::2 TESTIPV6ADDRESS1
   1:2:3:4:5:6:7:8 TESTIPV6ADDRESS2
```

- Created a CS03.ETC.IPNODES data set to define host names to be used by TSOUSER CS03 taking precedence over any other IPNODES data sets in our environment when using this TSO user (see Example 2-11 on page 42).

Example 2-11 CS03 local IPNODES data set

```
;  Entries in the hosts file have the following format:
;  Address HostName
;  Address HostName1 HostName2 HostName3 ..... HostName35
;  Address HostName1 HostName2 HostName3 ..... HostName35
;  Address: is an IP address, it can be IPV4 or IPV6 address.
;           Note: IPv4-mapped IPv6 address is not allowed.
;  HostName: the length of the hostname is up to 128 characters,
;            and each IP address can have up to 35 hostnames.
    10.12.4.221  tot177
   10.10.1.230 tcpip30
   10.10.1.221 tcpip32
   10.10.1.241 tcpip31
   10.10.2.1   router1
   10.10.3.2   router2
   1::2 TESTIPV6ADDRESS1
   1:2:3:4:5:6:7:8 TESTIPV6ADDRESS2
```

2.4.4 Verification

To verify whether the local implementation is working as expected, we checked whether FTPDA was started with the correct stack affinity by looking at its startup messages, as shown in Example 2-12.

Example 2-12 Startup messages

```
m  STARTING  FTPDA
m  FTPDA PROC
m  FTPD   FTPD:  STARTINGi  OMl  NOLIMITk  POSIX(ON) ALL31(ON)
ENVAR("_BPXK_SETIBMOPT_TRANSPORT=TCPIPA" "TX=EST")/
>  CEEDUMP.  *¢  TCPIP.FTPDA.STCO6903.DOOOO101.?
>  SYSFTPD   SHR¢ b TCPIPA.TCPPARMS FTPDA32
>  SYSTCPD   SHR¢ b TCPIPA.TCPPARMS DATAA32
```

The next step was to check if port 21 has been assigned to FTPDA1 as expected, by displaying the ports in stack TCPIPA, as shown in Example 2-13.

Example 2-13 Displaying the FTP port 21

```
D TCPIP,TCPIPA,N,CONN
EZDO101I NETSTAT CS V1R7 TCPIPA 525
USER ID  CONN     STATE
FTPDA1   OOOOO1DA LISTEN
  LOCAL SOCKET:   ::FFFF:10.10.1.221..21
  FOREIGN SOCKET: ::FFFF:0.0.0.0..0
```

To verify whether the user CS03 was using the correct IPNODES data set, a **ping** command to a host name defined in CS03.ETC.IPNODES was issued (see Example 2-14).

Example 2-14 Resolving the name TOT177 through local IPNODES data set

```
CSO3 @ SC32:/u/cs03>ping tot177
CS V1R7: Pinging host tot177 (10.12.4.221)
Ping #1 response took 0.001 seconds.
```

It is also possible to verify where the Resolver is looking, by using the TRACE RESOLVER parameter in the stack's or application's TCPIP.DATA data set. How this is done and the

contents of this trace will be explained in Diagnosing the Resolver address space environment.

2.4.5 Diagnosing the Resolver address space environment

To diagnose Resolver problems we can use two kinds of trace tools: The TRACE RESOLVER, which provides information that can be helpful in debugging problems an application program could have with using Resolver facilities (for example, GetHostByName or GetHostByAddr); and the Component Trace RESOLVER (SYSTCPRE) for diagnosing Resolver problems that cannot be isolated to one particular application.

In this section we provide a brief explanation of when to debug, which trace has to be used, and how to use these trace facilities. To find more information about Resolver diagnosis, refer to the *z/OS Communications Server: IP Diagnosis Guide, Version 1 Release 7*, GC31-8782-06.

Deciding which tool to use to diagnose a Resolver problem

The first thing to do when diagnosing a possible Resolver problem is to check the symptoms to verify if it is indeed a Resolver problem (see Table 2-2).

Table 2-2 What to do if the host name cannot be reached

When we ping a host name, the ping command:	What is the problem?	Solution
Succeeds, but another application fails when resolving the same host name.	The problem is with the Resolver configuration for the application in the users environment.	Use the Trace Resolver statement on the local TCPIP.DATA used by the application, which has the problem.
Fails, but the host name is converted to an IP address.	The resolution is successful but the host is not reachable or active.	This problem is related to connectivity, not a resolver problem.
Fails to convert the name to an IP address.	The problem might be with the resolver configuration, searching local host files, or using DNS.	Use Trace Resolver to solve the problem.

Tip: If the problem seems to be related to the DNS, use the LOOKUP statement to define to check the local files first.

TRACE RESOLVER

The Trace Resolver tells what the Resolver looked for (the questions) and where it looked (name server's IP addresses or local host file names).

The following situations can be checked in the trace output:

- Fix or check any problems reported at the top of the trace. These are errors in the Resolver data sets.
- Are the data sets being used by the Resolver the ones you expected? If not, see the search orders for data sets in the *z/OS Communications Server: IP Configuration Guide, Version 1 Release 7*, SC31-8775-07.
- Check if the data sets defined to be used are authorized by RACF and can be read by the application, TCP/IP stack, or user.

- Check the TCPIP.DATA parameter values, especially Search, NameServer, NSINTERADDR, and NsPortAddr.
- Check the questions posed by the Resolver to DNS or in searching the local host files. Are these the queries you expected?
- Look for errors or failures in the trace.
- Did DNS respond (if you expected it to)? If not, see whether DNS is active at the IP address you specified for NameServer and NSINTERADDR and what port it is listening on. Also, DNS logs can be helpful. Ask the DNS administrator for help.

To set up the trace Resolver, define the statement TRACE RESOLVER or OPTIONS DEBUG in the first line of the TCPIP.DATA data set being used by the application that has the problem you want to diagnose (see Example 2-15).

Example 2-15 Using the OPTIONS DEBUG to get a trace of the resolver

```
OPTIONS DEBUG
TCPIPJOBNAME TCPIPA
HOSTNAME WTSC32A
DOMAINORIGIN  ITSO.IBM.COM
DATASETPREFIX TCPIPA
MESSAGECASE MIXED
NSINTERADDR  10.12.6.7
NSPORTADDR 53
```

The resulting messages for a failing request are shown in Example 2-16.

Example 2-16 Failing local search

```
Resolver Trace Initialization Complete -> 2005/10/14 17:22:44.795904
res_init Resolver values:
Global Tcp/Ip Dataset  = TCPIPA.TCPPARMS(GLOBAL)
   Default Tcp/Ip Dataset = None
   Local Tcp/Ip Dataset   = //DD:SYSTCPD
                          ==> TCPIPA.TCPPARMS(DATAA32)
   Translation Table      = Default
   UserId/JobName         = CSO3
   Caller API             = TCP/IP Sockets Extended
   Caller Mode            = EBCDIC
   (G) DataSetPrefix = TCPIP
   (L) HostName      = WTSC32A
   (L) TcpIpJobName  = TCPIPA
   (G) Search        = ITSO.IBM.COM
                       IBM.COM
   (*) NameServer(s) = None
   (G) NsPortAddr    = 53            (G) ResolverTimeout    = 10
   (G) ResolveVia    = UDP           (G) ResolverUdpRetries = 1
   (*) Options NDots = 1             (L) Options Debug
   (*) SockNoTestStor
   (*) AlwaysWto     = NO            (G) MessageCase        = MIXED
   (G) LookUp        = LOCAL
GetAddrInfo Started: 2005/10/14 17:22:44.810254
 GetAddrinfo Invoked with following inputs:
    Host Name:  HOST1
    No Service operand specified
    Hints parameter supplied with settings:
        ai_family = 0, ai_flags = 0x00000062
        ai_protocol = 0, ai_socktype = 0
 No NameServers specified, no DNS activity
```

```
 GetAddrInfo Opening Socket for IOCTLs
  BPX1SOC:  RetVal = 0, RC = 0, Reason = 0x00000000
 GetAddrInfo Opened Socket 0x00000000
 GetAddrInfo Only IPv4 Interfaces Exist
 GetAddrInfo Searching Local Tables for IPv4 Address
 Global IpNodes Dataset  = None
 Default IpNodes Dataset = TCPIPD.TCPPARMS(IPNODES)
 Search order            = CommonSearch
  SITETABLE from userid/jobname CS03.ETC.IPNODES
  - Lookup for HOST1.ITSO.IBM.COM
  - Lookup for HOST1.IBM.COM
- Lookup for HOST1
  GetAddrInfo Closing IOCTL Socket 0x00000000
   BPX1CLO:  RetVal = 0, RC = 0, Reason = 0x00000000
  GetAddrInfo Failed:  RetVal = -1, RC = 1, Reason = 0x78AE1004
 GetAddrInfo Ended: 2005/10/14 17:22:44.818959
 ************************************************************************
 Unknown host 'HOST1'
 ***
```

In this sample we tried to ping HOST in a local search only, because we specified LOOKUP LOCAL statement in our TCPIP.DATA data set.

Our next TRACE RESOLVER output shows a successful search (see Example 2-17), showing only the relevant messages.

Example 2-17 Successful local search

```
Resolver Trace Initialization Complete -> 2005/10/14 17:30:07.701382
res_init Resolver values:
Global Tcp/Ip Dataset  = TCPIPA.TCPPARMS(GLOBAL)
   Default Tcp/Ip Dataset = None
   Local Tcp/Ip Dataset   = //DD:SYSTCPD
                          ==> TCPIPA.TCPPARMS(DATAA32)
   Translation Table      = Default
   UserId/JobName         = CS03
   Caller API             = TCP/IP Sockets Extended
   Caller Mode            = EBCDIC
   (G) DataSetPrefix = TCPIP
   (L) HostName      = WTSC32A
   (L) TcpIpJobName  = TCPIPA
   (G) Search        = ITSO.IBM.COM
                       IBM.COM
   (*) NameServer(s) = None
   (G) NsPortAddr    = 53            (G) ResolverTimeout    = 10
   (G) ResolveVia    = UDP           (G) ResolverUdpRetries = 1
   (*) Options NDots = 1             (L) Options Debug
   (*) SockNoTestStor
   (*) AlwaysWto     = NO            (G) MessageCase        = MIXED
   (G) LookUp        = LOCAL
 GetAddrInfo Started: 2005/10/14 17:30:07.711177
 GetAddrinfo Invoked with following inputs:
    Host Name:  TOT177
 No NameServers specified, no DNS activity
 GetAddrInfo Opening Socket for IOCTLs
 GetAddrInfo Only IPv4 Interfaces Exist
 GetAddrInfo Searching Local Tables for IPv4 Address
 Global IpNodes Dataset  = None
 Default IpNodes Dataset = TCPIPD.TCPPARMS(IPNODES)
 Search order            = CommonSearch
```

```
  SITETABLE from userid/jobname CS03.ETC.IPNODES
  - Lookup for TOT177.ITSO.IBM.COM
  - Lookup for TOT177.IBM.COM
  - Lookup for TOT177
  ADDRTABLE from userid/jobname CS03.ETC.IPNODES
  - Lookup for 10.12.4.221
GetAddrInfo Succeeded:  IP Address(es) found:
  IP Address(1) is 10.12.4.221
GetAddrInfo Ended: 2005/10/14 17:30:07.723637
********************************************************************************
FreeAddrInfo Started: 2005/10/14 17:31:23.043887
FreeAddrInfo Called to free addrinfo structures
FreeAddrInfo Succeeded, Freed 1 Addrinfos
FreeAddrInfo Ended: 2005/10/14 17:31:23.043929
********************************************************************************
CS V1R7: Pinging host tot177 (10.12.4.221)
Ping #1 response took 0.001 seconds.
***
```

If you do not know or you think you might be using the wrong TCPIP.DATA data set, there is another way to activate the TRACE RESOLVER, by specifying the z/OS UNIX RESOLVER_TRACE environment variable or a SYSTCPT DD allocation.

Specifying the RESOLVER_TRACE environment variable or allocating the SYSTCPT DDname dynamically activates Trace Resolver output regardless of the TCPIP.DATA being used.

CTRACE - RESOLVER (SYSTCPRE)

Component Trace (CTRACE) is used for the RESOLVER component (SYSTCPRE) to collect debug information. The TRACE RESOLVER traces information on a per-application basis and directs the output to a unique file for each application. The CTRACE shows Resolver actions for all applications (although it might be filtered).

The CTRACE support allows for JOBNAME, ASID filtering, or both. The trace buffer is located in the Resolver private storage. The trace buffer minimum size is 128 K, maximum 128 M, default 16 M. Trace records can optionally be written to an external writer.

The Resolver CTRACE can be started anytime you need by using the TRACE CT command, or it can be activated during Resolver procedure initialization.

The Resolver CTRACE initialization PARMLIB member can be specified at Resolver start time. To activate the Resolver CTRACE during resolver initialization, follow these steps:

1. Create a CTWTR procedure in your SYS1.PROCLIB, as shown in Example 2-18.

Example 2-18 CTWTR procedure

```
//CTWTR    PROC
//IEFPROC  EXEC PGM=ITTTRCWR
//TRCOUT01 DD  DSNAME=CS03.CTRACE1,VOL=SER=COMST2,UNIT=3390,
//         SPACE=(CYL,10),DISP=(NEW,KEEP),DSORG=PS
//TRCOUT02 DD  DSNAME=CS03.CTRACE2,VOL=SER=COMST2,UNIT=3390,
//         SPACE=(CYL,10),DISP=(NEW,KEEP),DSORG=PS
//*
```

2. Using the sample Resolver procedure shipped with the product, enter the following console command:

```
S RESOLV32,PARMS='CTRACE(CTIRES xx)'
```

Where *xx* is the suffix of the CTIRESxx PARMLIB member to be used. To customize the parameters used to initialize the trace, you can update the SYS1.PARMLIB member CTIRES00, as shown in Example 2-19.

Example 2-19 Trace options

```
/*********************************************************************/
TRACEOPTS
/* ---------------------------------------------------------------- */
/*   Optionally start external writer in this file (use both       */
/*   WTRSTART and WTR with same wtr_procedure)                     */
          WTRSTART(CTWTR)
/* ---------------------------------------------------------------- */
/*   ON OR OFF: PICK 1                                             */
/* ---------------------------------------------------------------- */
          ON
/*        OFF                                                     */
/*   BUFSIZE: A VALUE IN RANGE 128K TO 128M                        */
          BUFSIZE(16M)
/*        JOBNAME(jobname1,...)                                   */
/*        ASID(Asid1,...)                                         */
          WTR(CTWTR)
/* ---------------------------------------------------------------- */
/*   OPTIONS: NAMES OF FUNCTIONS TO BE TRACED, OR "ALL"            */
/* ---------------------------------------------------------------- */
/*        OPTIONS(            */
/*                'ALL     '  */
/*               ,'MINIMUM '  */
/*                )           */
```

3. Use the TRACE CT command to define the options (see Example 2-20).

Example 2-20 TRACE CT command flow

```
  TRACE CT,ON,COMP=SYSTCPRE,SUB=(RESOLV32)
 *189 ITT006A SPECIFY OPERAND(S) FOR TRACE CT COMMAND.
  R 189,OPTIONS=(ALL),END
  IEE600I REPLY TO 189 IS;OPTIONS=(ALL),END
  ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
  WERE SUCCESSFULLY EXECUTED.
  IEE839I ST=(ON,0256K,00512K) AS=ON  BR=OFF EX=ON  MT=(ON,024K) 497
          ISSUE DISPLAY TRACE CMD FOR SYSTEM AND COMPONENT TRACE STATUS
          ISSUE DISPLAY TRACE,TT CMD FOR TRANSACTION TRACE STATUS
```

4. Reproduce the problem.
5. Save the trace contents into the trace file created by the CTWRT procedure, executing the the commands shown in Example 2-21.

Example 2-21 Saving the trace contents

```
  TRACE CT,ON,COMP=SYSTCPRE,SUB=(RESOLV32)
 *190 ITT006A SPECIFY OPERAND(S) FOR TRACE CT COMMAND.
  R 190,WTR=DISCONNECT,END
  IEE600I REPLY TO 190 IS;WTR=DISCONNECT,END
  ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
  WERE SUCCESSFULLY EXECUTED.
  IEE839I ST=(ON,0256K,00512K) AS=ON  BR=OFF EX=ON  MT=(ON,024K) 503
          ISSUE DISPLAY TRACE CMD FOR SYSTEM AND COMPONENT TRACE STATUS
          ISSUE DISPLAY TRACE,TT CMD FOR TRANSACTION TRACE STATUS
```

6. Stop the CTRACE by issuing the command shown in Example 2-22.

Example 2-22 Stopping CTRACE

```
TRACE CT,OFF,COMP=SYSTCPRE,SUB=(RESOLV32)
ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
WERE SUCCESSFULLY EXECUTED.
IEE839I ST=(ON,0256K,00512K) AS=ON  BR=OFF EX=ON  MT=(ON,024K) 506
        ISSUE DISPLAY TRACE CMD FOR SYSTEM AND COMPONENT TRACE STATUS
        ISSUE DISPLAY TRACE,TT CMD FOR TRANSACTION TRACE STATUS
```

After these steps, we will have a trace file to be formatted using the IPCS command:

```
CTRACE COMP(SYSTCPRE) TALLY
```

The resulting display will show the Resolver process entries, as shown in Example 2-23.

Example 2-23 Resolver formatted trace entries

```
COMPONENT TRACE TALLY REPORT
SYSNAME(SC32)
COMP(SYSTCPRE)
    TRACE ENTRY COUNTS AND AVERAGE INTERVALS (IN MICROSECONDS)

FMTID    COUNT       Interval     MNEMONIC DESCRIBE
-------- ----------- ------------ -------- --------------------------------
00000001           0              CTRACE    CTrace Initialized
00000002           0              CTRACE    Status changed or displayed
00000003           0              CTRACE    CTrace Terminated
00000004           0              CTRACE   !CTrace has abended
00000005           0              CTRACE    CTrace Stopped - Buffers Retain
00010001           0              API       GetHostByAddr Entry Parameters
00010002           0              API       GetHostByAddr Stack Affinity
00010003           0              API       GetHostByAddr Failure
00010004           0              API       GetHostByAddr Success
00010005           0              API       GetHostByAddr GetLocalHostName
00010006           0              API       GetHostByName Entry Parameters
00010007           0              API       GetHostByName Stack Affinity
00010008           0              API       GetHostByName Failure
00010009           0              API       GetHostByName Success
```

For more information about Resolver CTRACE analysis, refer to *z/OS V1R7.0 Communications Server: IP Diagnosis Guide*, GC31-8782.

3

Base functions

The term *base functions* implies the minimum configuration required for the proper operation of a z/OS TCP/IP environment. The base functions described in this chapter are considered necessary for any useful deployment of the TCP/IP stack and commonly used applications.

This chapter discusses the following.

Section	Topic
3.1, “The base functions” on page 50	Discusses the basic concepts of base functions
3.2, “Why base functions are important” on page 51	Discusses key characteristics of base functions and why they may be important in your environment
3.3, “Common design scenarios for base functions” on page 51	Presents commonly implemented base functions design scenarios, their dependencies, advantages, and considerations and our recommendations
3.4, “How the base functions are implemented in a single stack” on page 55	Presents selected implementation scenarios, tasks, configuration examples, and problem determination suggestions

3.1 The base functions

Base functions are those functions considered to be standard in TCP/IP environments regardless of the implementation. Most of these functions are implemented at the lower layers. There are some base functions that are implemented at the application layer (such as Telnet and FTP). However, the details of the standard applications can be found in *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 2 - Standard Applications*, SG24-7170. Here we discuss the configuration that provides the infrastructure of the TCP/IP protocol suite in the z/OS Communications Server environment.

3.1.1 Basic concepts

The z/OS TCP/IP stack (a TCP/IP instance) is a full functional implementation of the standard RFC protocols that are fully integrated and tightly coupled between z/OS and UNIX System Services. It provides the environment that supports the base functions as well as the many traditional TCP/IP applications. The two environments that need to be created and customized to support the z/OS Communications Server for TCP/IP are:

- A native z/OS environment in which users can exploit the TCP/IP protocols in a standard z/OS application environment such as batch jobs (with JES interface), started tasks, TSO, CICS, and IMS applications.
- A z/OS UNIX System Services environment that lets you develop and use applications and services that conform to the POSIX or XPG4 standards (UNIX specifications). The z/OS UNIX environment also provides some of the base functions to support the z/OS environment and vice versa.

Since the z/OS Communications Server exploits z/OS UNIX services even for traditional z/OS environments and applications a full-function mode z/OS UNIX environment—including a Data Facility Storage Management Subsystem (DFSMS), a z/OS UNIX file system and a security product (such as Resource Access Control Facility, or RACF) are required before the z/OS Communications Server can be started successfully and the TCP/IP environment initialized.

3.1.2 For additional information

When you install and customize the z/OS Communications Server IP, it will be very helpful to have the following documentation and product publications available:

- Implementation and migration plans, fallback plans, and test plans that you have created and customized for your environment
- Printouts of procedures and data sets that you will be using for the implementation
- *z/OS V1R7.0 Program Directory, Program Number 5694-A01*, GI10-0670
- *z/OS V1R7.0 XL C/C++ Run-Time Library Reference*, SA22-7821
- *z/OS V1R7.0 Migration*, GA22-7499
- *z/OS V1R7.0 Communications Server: IP Configuration Guide*, SC31-8775
- *z/OS V1R7.0 Communications Server: IP Configuration Reference*, SC31-8776
- *z/OS V1R7.0 Communications Server: IP Messages Volume 1 (EZA)*, SC31-8783
- *z/OS V1R7.0 Communications Server: IP Messages Volume 2 (EZB, EZD)*, SC31-8784
- *z/OS V1R7.0 Communications Server: IP Messages Volume 3 (EZY)*, SC31-8785
- *z/OS V1R7.0 Communications Server: IP Messages Volume 4 (EZZ, SNM)*, SC31-8786
- *z/OS V1R7.0 Communications Server: IP and SNA Codes*, SC31-8791

- *z/OS V1R7.0 UNIX System Services Planning*, GA22-7800
- *z/OS V1R7.0 UNIX System Services User's Guide*, SA22-7801
- *z/OS V1R7.0 UNIX System Services Messages and Codes*, SA22-7807
- *z/OS V1R7.0 MVS System Messages, Vol 1 (ABA-AOM)*, SA22-7631
- *z/OS V1R7.0 MVS System Messages, Vol 2 (ARC-ASA)*, SA22-7632
- *z/OS V1R7.0 MVS System Messages, Vol 3 (ASB-BPX)*, SA22-7633
- *z/OS V1R7.0 MVS System Messages, Vol 4 (CBD-DMO)*, SA22-7634
- *z/OS V1R7.0 MVS System Messages, Vol 5 (EDG-GFS)*, SA22-7635
- *z/OS V1R7.0 MVS System Messages, Vol 6 (GOS-IEA),* SA22-7636
- *z/OS V1R7.0 MVS System Messages, Vol 7 (IEB-IEE)*, SA22-7637
- *z/OS V1R7.0 MVS System Messages, Vol 8 (IEF-IGD)*, SA22-7638
- *z/OS V1R7.0 MVS System Messages, Vol 9 (IGF-IWM)*, SA22-7639
- *z/OS V1R7.0 MVS System Messages, Vol 10 (IXC-IZP)*, SA22-7640

3.2 Why base functions are important

Base functions are important because they establish a functional working environment that may be exploited by other features or upon which many other functions may be implemented or validated. When the base functions are implemented, they exercise the most commonly used code and features of a TCP/IP environment, providing an effective way to perform integrity tests and validate the TCP/IP environment before embarking on the more complex features, configurations, and implementations of the stack.

3.3 Common design scenarios for base functions

Because base functions are primarily setting up the *primitives* in the TCP/IP environment, we deal with very basic scenarios, which can be built upon at a later time. For the base functions we consider two scenarios:

- Single TCP/IP stack environment
- Multiple TCP/IP stack environment

3.3.1 Single stack

A single stack environment is the existence of one TCP/IP system address space in a single z/OS image (LPAR) providing support for the functions and features of the TCP/IP protocol suite. The tasks to complete the configuration of a single stack may be achieved using the IBM z/OS IP Configuration Wizard or msys for Setup configuration tools. We will explore using those IBM offerings to perform the configuration task to set up the single stack environment. However, since these tools provide just the basic setup to get a minimally configured environment up and running, we additionally use conventional z/OS commands to add features to the configuration as we create our single stack environment.

Dependencies

In order to achieve a successful implementation of the z/OS Communications Server - TCP/IP component we identified certain dependencies:

- Implement a *full-function* UNIX System Services system on z/OS. Details are available in the *z/OS V1R7.0 UNIX System Services Planning*, GA22-7800-08;*z/OS V1R7.0 Program Directory*, GI10-0670-07 (http://publibz.boulder.ibm.com/epubs/pdf/i1006707.pdf); and *z/OS V1R7.0 MVS Initialization and Tuning Reference*. SA22-7592-11.
- Define a RACF environment for the z/OS Communications Server - TCP/IP component. This includes defining RACF groups to z/OS UNIX groups to manage resources, profiles, user groups, and user IDs. An OMVS UID must be defined with UID (0) and assigned to the started task name of the CS for z/OS IP system address space. Details are available in *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 4 - Security*, SG24-7172; *z/OS V1R7.0 Security Server RACF Security Administrator's Guide*, SA22-7683-07; *z/OS V1R7.0 Security Server RACF System Programmer's Guide*, SA22-7681-06; *z/OS V1R7.0 Security Server RACF Command Language Reference*, SA22-7687-08.
- Customize SYS1.PARMLIB members with special reference to BPXPRMxx to use the integrated sockets INET with the AF_INET and AF_INET6 physical file system. Details are available in the *z/OS V1R7.0 MVS Initialization and Tuning Reference*, SA22-7592-11; *z/OS V1R7.0 UNIX System Services Planning*, GA22-7800; and *z/OS V1R7.0 Program Directory*, GI10-0670.
- Customize the TCP/IP configuration data sets:
  - PROFILE.TCPIP
  - TCPIP.DATA
  - Other configuration data sets
- Fully functional VTAM to support the interfaces used by TCP/IP.

Advantages

The advantages of a single stack are:

- Less CPU cycles spent on processing TCP/IP traffic, since there is only one logical instance of each physical interface in a single stack environment versus a multiple stack environment.
- Servers use fewer CPU cycles when certain periodic updates arrive (OMPROUTE processing routing updates). Multiple stacks mean multiple copies of OMPROUTE.
- Each stack requires a certain amount of storage, the most significant being virtual storage.

Considerations

When creating a TCP/IP stack we need to consider the other requirements upon which the successful initialization of the stack depends. Very often the initial problems encountered are related to the omission of tasks that were not performed by other disciplines such as RACF administration. *CS for z/OS TCP/IP* exploits the tightly coupled design of the z/OS Communications Server, the integration of z/OS and UNIX System Services, and the provision of RACF services. Coordination is the key to a successful implementation the TCP/IP stack.

3.3.2 Multiple stacks

A multiple stack environment consists of more than one stack running concurrently in a single LPAR. They exist independent of each other, with the ability to be uniquely configured. Each stack can support different features and provide different functions. Each stack is configured

in its own address space and can communicate with the other stacks in the LPAR if so desired.

Dependencies

The dependencies for the multiple stack environment are exactly the same as for the single stack environment, as well as:

- Additional storage, especially virtual storage
- Additional CPU cycles for processing subsequent interfaces and services performing periodic functions, such as OMPROUTE routing updates

Advantages

There are some advantages for running a multiple stack environment, since it gives us the flexibility to partition our networking environment. Here are some advantages to consider:

- You might wish to establish separate stacks to separate workloads based on availability and security. For example, you might have different requirements for a production stack, a system test stack, and a secure stack.

  This approach could, for example, be used to establish a test TCP/IP stack, where new socket applications are tested before they are moved into the production system. You may also want to apply maintenance to a non-production stack so it can be tested before you apply it to the production stack.
- Your strategy might be to separate workload onto multiple stacks based on the functional characteristics of applications, as with UNIX (OpenEdition) applications and non-UNIX (z/OS) applications.
- You may be running z/OS servers and UNIX (OpenEdition) servers on the same well-known port (TN3270 and otelnet on port 23). An alternative to this is approach is the BIND for INADDR_ANY function.

Whatever the reason, the ability to configure multiple stacks and have them fully functional, independently and concurrently, can be exploited in many different ways.

Considerations

The consideration are primarily the same as they are for a single stack environment. We therefore indicate only the differences and the additional considerations regarding the multiple stack environment.

Sharing Resolver between multiple stacks

The general recommendation is that you use separate DATASETPREFIX values per stack and create separate copies of configuration data sets or at the very least Resolver datasets. Refer to Chapter 2, “The Resolver” on page 17, for further details.

Selecting the correct configuration data sets

The Resolver needs access to all Resolver data sets if there are multiple stacks in multiple z/OS LPARs. Refer to Chapter 2, “The Resolver” on page 17, for further details.

TSO clients

TSO client functions may be directed against any number of TCP/IP stacks. Obviously the client must be able to find the TCPIP.DATA data set appropriate for the stack of interest. You may modify your TSO logon procedure with a SYSTCPD DD statement or use a common TSO logon procedure without the SYSTCPD DD statement and allocate the TCPIP.DATA dataset to the appropriate stack of interest.

Stack affinity

Any server or client needs to reference the appropriate stack if the desired stack is not the default stack defined in the BPXPRMxx member of SYS1.PARMLIB. Servers may use the BPXK_SETIBMOPT_TRANSPORT environment variable to override the choice of the default stack. There may also be applications that have affinity to the wrong stack and do not have the option of establishing stack affinity. In those instances, you can execute BPXTCAFF prior to the application execution step. For example:

```
//AFFINITY EXEC PGM=BPXTCAFF,PARM='TCPIPA'
```

This assumes TCPIPA is not the default stack.

Port management

When there is a single stack and the relationship of server to stack is 1:1, port management is relatively simple. Using the PORT statement, the port number can be reserved for the server in the PROFILE.TCPIP for that given stack.

Port management becomes more complex in an environment where there are multiple stacks and a potential for multiple combinations of the same server (for example, UNIX System Services and TN3270/TN3270E).

So, in a multiple stack environment, you need to answer some questions based on the following concepts:

- Generic server

  A generic server is a server without affinity for a specific stack, and it provides service to any client on the network. FTP is an example, since the stack is merely a connection linking client and server. The service File Transfer is not related to the internal functioning of the stack, and the server can communicate concurrently over any number of stacks.

- Servers with an affinity for a specific stack

  There must be an explicit binding of the server application to the chosen stack when the service is related to the internal functioning of the stack. You can take UNIX System Services DNS, OSNMP, and ONETSTAT as examples for that.

  This bind is made via the setibmopt() socket call to specify which stack they have chosen, or via the C function _iptcpn() that allows applications to search in the TCPIP.DATA file to find the name of a specific stack.

- Ephemeral ports

  As well as synchronizing PORT reservations for specific applications across all stacks, you have to synchronize reservations for port numbers that will be dynamically assigned across all stacks when running with multiple stacks.

  Those ports are called ephemeral ports, which are all above 1024, and are assigned by the stack when none is specified on the application bind(). You have the PORTRANGE statement in the PROFILE.TCPIP to reserve a group of ports, and you should specify the same port range for every stack. You also need to let CINET know which ports are guaranteed to be available on every stack, using the BPXPRMxx parmlib member through INADDRANYPORT and INADDRANYCOUNT statements.

CPU resources

Provisions need to be made for additional CPU cycles and storage (especially virtual storage). These increases in resources are just for the existence of the additional stacks running concurrently.

3.3.3 Recommendation

Although there are specialized cases where multiple stacks per LPAR can provide value, we in general recommend implementing only one TCP/IP stack per LPAR. The reasons for this recommendation are as follows:

- A TCP/IP stack is capable of exploiting all available resources defined to the LPAR in which it is running. Therefore, starting multiple stacks will not yield any increase in throughput.
- When running multiple TCP/IP stacks additional system resources, such as memory, CPU cycles, and storage are required.
- Multiple TCP/IP stacks add a significant level of complexity to TCP/IP system administration tasks.
- It is not necessary to start multiple stacks to support multiple instances of an application on a given port number, such as a test HTTP server on port 80 and a production HTTP server also on port 80. This type of support can instead be implemented using BIND-specific support where the two HTTP server instances are each associated to port 80 with their own IP address, via the BIND option on the PORT reservation statement.

One example where multiple stacks can have value is when an LPAR needs to be connected to multiple isolated security zones in such a way that there is no network level connectivity between the security zones. In this case, a TCP/IP stack per security zone can be used to provide that level of isolation, without any network connectivity between the stacks.

3.4 How the base functions are implemented in a single stack

There are several areas of interest that require your attention and action in order to implement a TCP/IP stack successfully.

3.4.1 z/OS tasks for UNIX Systems Services

In Chapter 1, “Introduction” on page 1, we reviewed the UNIX concepts in the z/OS environment. We made specific references to the BPXPRMxx member in SYS1.PARMLIB. We now need to identify the parameters in BPXPRMxx that warrant our attention and action. However, first we need to discuss important security considerations for the UNIX environment.

RACF actions for UNIX

Security is an important consideration for most z/OS installations and there are a few features we need to mention here for the base functions of any TCP/IP environment. TCP/IP has some built-in internal security mechanisms and relies on the services of a security manager, such as the IBM Resource Access Control Facility (RACF).

A security manager is a requirement in the z/OS Communications Server IP environment. As an online application, it is important that TCP/IP undergo security checks to eliminate possible security exposures. Some basic security concepts are included in the following sections, but for a more detailed explanation refer to *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 7: Security*, SG24-6840.

APF authorization

The TCP/IP system program libraries must be APF authorized. Authorized Program Facility (APF) means that z/OS built-in security may be bypassed by programs that are executed

from such libraries. CS for z/OS IP data sets have to be protected with RACF. Special attention has to be given to the APF authorized libraries defined in PROGxx.

We used the LNKAUTH=LNKLST specification in SYSx.PARMLIB member IEASYSxx, which means that all libraries in the LNKLST concatenation will be APF authorized. If these libraries are accessed through STEPLIB or JOBLIB, they will not be APF authorized unless they have been specifically defined in the IEAAPFxx or PROGxx member.

SEZALOAD is one library that *must* be made part of your LNKLST concatenation. Because of the LNKAUTH=LNKLST specification, it will be APF authorized when it is accessed through the LNKLST concatenation. The SEZALOAD library holds the TCP/IP system code used by both servers and clients.

In addition to the LNKLST libraries, there are some libraries that are not accessed through the LNKLST concatenation, but have to be APF authorized. The SEZATCP library holds the TCP/IP system code used by servers. The library is normally placed in the STEPLIB or JOBLIB concatenation, which is part of the server JCL.

The following libraries may have to be APF authorized, depending on the choices you make during the installation of z/OS:

SEZALPA This library holds the TCP/IP modules that must be made part of your system's LPA. If you choose to add the library name to your LPALSTxx member in SYSx.PARMLIB, you also have to make sure the library is APF authorized. If you copy the load modules in the library to an existing LPALSTxx data set, you do not need to authorize the SEZALPA data set.

SEZADSIL This library holds the load modules used by the SNMP command processor running in the NetView® address space. If you choose to concatenate this library to STEPLIB in the NetView address space, you may have to APF authorize it, if other libraries in the concatenation are already APF authorized.

Every APF-authorized online application may have to be reviewed to ensure that it matches the security standards of the installation. A program is a “well-behaved program” if:

- Logged-on users cannot access or modify system resources for which they are not authorized.
- The program does not require any special credentials to be able to execute.

  Or, in the case of RACF, the program does not need the RACF authorization attribute OPERATIONS for execution.

> **Note:** User IDs with the RACF attribute OPERATIONS have ALTER access to all data sets in the system. The access authority to single data sets may be specifically lowered or excluded.

RACF environment

RACF is very flexible and can be set up and tailored to meet almost all security requirements of large enterprises. All RACF implementations are based on the following key elements:

- User IDs
- Groups
- RACF resources
- RACF profiles

- RACF facility classes
- The hierarchical owner principle, which is applicable for all RACF definitions of user IDs, groups, and RACF resources

RACF implementation

Each unit of work in the z/OS system that requires UNIX System Services must be associated with a valid UNIX System Services identity. A valid identity refers to the presence of a valid UNIX user ID (UID) and a valid UNIX group ID (GID) for each such user. The UID and the GID are defined through the OMVS segment in the user's RACF user profile and in the group's RACF group profile.

Each functional RACF access group must be authorized to access a specific TCP/IP RACF resource with a specific access attribute. The details of this process are discussed in *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 4 - Security*, SG24-7172.

Assigning user IDs to started tasks

In some cases, the user ID and started task must be associated with the UNIX superuser. In other cases, you can associate the user ID and started task with the default user.

RACF offers you two techniques to assign user IDs and group IDs to started tasks:

- The started procedure name table (ICHRIN03).
- The RACF STARTED resource profiles.

  By using the STARTED resources, you can add new started tasks to RACF, and immediately make those new definitions active.

```
IEF695I START TO3DNS   WITH JOBNAME TO3DNS   IS ASSIGNED TO USER TCPIP3, GROUP OMVSGRP
```

The user ID and default group must be defined in RACF, which then treats the user ID as any other RACF user ID for its resource access checking. RACF allows multiple started procedure names to be assigned to the same RACF user ID. We used this method to assign RACF user IDs to *all* TCP/IP started tasks.

Started task user IDs

The UNIX System Services tasks OMVS and BPXOINIT need to execute in an z/OS system space and have the special user ID OMVSKERN assigned to them. OMVSKERN has to be defined as superuser with UID 0, program /bin/sh, and home directory.

TCP/IP tasks need RACF user IDs with the OMVS segment defined. The user ID associated with the main TCP/IP address space must be defined as a superuser; the requirements for the individual servers vary but most need to be a superuser as well.

z/OS VARY TCPIP commands

Access to VARY TCPIP commands can be controlled by RACF. This places restrictions on this command, which may be used to alter and disrupt the TCP/IP environment.

NETSTAT command

Access to the TSO **NETSTAT** command, the UNIX shell command **onetstat**, and command options can be controlled by RACF, by defining NETSTAT resources to the RACF generic class SERVAUTH. This command may also need to be restricted, as it can be used to alter/drop connections or stop the TN3270 server.

Establish RACF security environment

The notes that follow are merely an overview of the steps in the process. You should consult the instructions in *z/OS Security Server RACF Callable Services,* SA22-7691, to accomplish these tasks.

1. Defining commands for CS for z/OS IP in the RACF OPERCMDS class.
2. Establishing a group ID for a default OMVS group segment:

```
ADDGROUP OEDFLTG OMVS(GID(9999))
```

3. Defining a user ID for a default OMVS group segment:

```
RDEFINE FACILITY BPX.DEFAULT.USER APPLDATA('OEDFLTU/OEDFLTG')
ADDUSER OEDFLTU DFLTGRP(OEDFLTG) NAME('OE DEFAULT USER') PASSWORD(xg18ej)
OMVS(UID(999999) HOME('/') PROGRAM('/bin/sh'))
```

4. Activating or refreshing appropriate facility classes:

```
SETROPTS CLASSACT(FACILITY)
SETROPTS RACLIST(FACILITY)
SETROPTS RACLIST(FACILITY) REFRESH
```

5. Defining one or more superuser IDs to be associated with certain UNIX System Services users and TCP/IP started tasks:

```
ADDGROUP OMVSGRP OMVS(GID(1))
ADDUSER TCPIP3 DFLTGRP(OMVSGRP) OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
```

6. Defining other UNIX System Services users.

   You may already have defined RACF groups and users. If this is the case, you can set up a z/OS UNIX file system home directory for each user, add an OMVS identity by altering the group to include a GID (ALTGROUP), and then using the ISHELL utility add OE segments for UNIX System Services users (associating them with the altered group and giving each user a distinct UID).

   Otherwise, you will have to perform these tasks in a more painstaking manner, for example:

```
ADDGROUP usergrp OMVS(GID(10))
ADDUSER user01 DFLTGRP(usrgrp) OMVS(UID(20) HOME('/u/user01') PROGRAM('/bin/sh/'))
```

More information about RACF with CS for z/OS TCP/IP

RACF can be used to protect many TCP/IP resources, such as the TCP/IP stack itself and ports. Further information about securing your TCP/IP implementation can be found in *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 4 - Security*, SG24-7172.

SYS1.PARMLIB

As we noted, the z/OS environment consists of the traditional MVS and UNIX Systems Services (USS) environment. Since the USS environment is implemented within a z/OS system space, there are definitions in the z/OS environment upon which the USS environment depends.

SYS1.PARMLIB is the single most important data set in the z/OS environment. It contains most of the parameters that define z/OS as well as many other subsytems. The SYS1.PARMLIB data set definition parameters are critical to the proper initialization and functioning of USS and, therefore, to the TCP/IP implementation. Some of the members of interest (discussed below) include:

- IEASYS00
- BPXPRMxx
- Integrated Sockets PFS definitions

IEASYS00

Since the z/OS Communications Server exploits z/OS UNIX services even for traditional MVS environments and applications, a full-function mode z/OS UNIX environment—including a Data Facility Storage Management Subsystem (DFSMS) and z/OS File Systems (including z/OS UNIX file system)—is required before the z/OS Communications Server can be started and the TCP/IP environment successfully established.

The IEASYS00 parmlib definitions that we used that are relevant to TCP/IP are:

```
OMVS=7A,
SMS=00,
```

OMVS=7A specifies that BPXPRM7A is used to configure the z/OS UNIX environment at system initialization time. SMS=00 specifies that IGDSMS00 is to be used for definitions of the Data Facility Storage Management Subsystem at z/OS UNIX initialization time.

BPXPRMxx

All the parameters defined in BPXPRMxx should be reviewed and tailored to individual installation specification and resource utilization. The *z/OS V1R7.0 UNIX System Services Planning*, GA22-7800-08, and the *z/OS MVS Initialization and Tuning Guide*, SA22-7591, explain the details and significance of each parameter in the BPXPRMxx member.

The *z/OS V1R7.0 UNIX System Services Planning*, GA22-7800-08; *z/OS V1R7.0 UNIX System Services User's Guide*, SA22-7802-07; and *z/OS V1R7.0 Program Directory*, GI10-0670-07 (http://publibz.boulder.ibm.com/epubs/pdf/i1006707.pdf), detail the structure, design, installation, and implementation of the z/OS UNIX environment. Concepts such as Logical and Physical File Systems (PFS) are design components of z/OS UNIX and are not discussed here.

Integrated Sockets PFS definitions

We need to define the desired file systems to support the communication provided by the stack. Example 3-1 illustrates how support for IPv4 and IPv6 (dual mode) is defined for a single stack environment.

Example 3-1 BPXPRMxx definitions for a single stack supporting dual mode

```
FILESYSTYPE TYPE(UDS)
            ENTRYPOINT(BPXTUINT)
 NETWORK DOMAINNAME(AF_UNIX)
         DOMAINNUMBER(1)
         MAXSOCKETS(10000)
         TYPE(UDS)

 FILESYSTYPE TYPE(INET)
             ENTRYPOINT(EZBPFINI)
/* IPv4 support
 NETWORK DOMAINNAME(AF_INET)  1
         DOMAINNUMBER(2)
         MAXSOCKETS(25000)
         TYPE(INET)
         INADDRANYPORT(10000)
         INADDRANYCOUNT(2000)
/* IPv6 support
 NETWORK DOMAINNAME(AF_INET6) 2
        DOMAINNUMBER(19)
        TYPE(INET)
```

INET specifies a single stack with TCP/IP (by default) as the stack name.

EZBPFINI identifies a TCP/IP stack (this is the only valid value).

AF_INET specifies IPv4 support for the physical file type for socket address used by this stack (TCPIP) **1**.

AF_INET6 specifies IPv6 support for the physical file type for socket address used by this stack (TCPIP) **2**. With both IPv4 and IPv6 defined, dual-mode is supported.

Specifying NETWORK definitions for both AF_NET and AF_INET6 provides dual-support. If IPv6 support is not desired then you can omit the second NETWORK DOMIAINNAME statement and subsequent parameters.

Additional z/OS customization for USS

Updating the MVS system libraries must be done with great care. Follow the instructions in the *z/OS V1R7.0 Program Directory, Program Number 5694-A01,* GI10-0670, and check the PSP bucket to ensure that all required PTFs and modifications are done as required. You may need to make changes to some or all of the following members, depending on the features you are installing. Use the following checklist as an introduction to the process of modifying the z/OS system:

SYSx.PARMLIB updates

The updates are:

1. LNKLSTxx

   Add the following CS for z/OS IP link libraries to the z/OS system link list:

   – hlq.SEZALOAD
   – hlq.SEZALNK2

2. LPALSTxx

   Add the following CS for z/OS IP LPA modules to the LPA during IPL of z/OS:

   – hlq.SEZALPA

   Note: hlq.SEZALPA must be cataloged into the MVS master catalog. hlq.SEZALOAD and hlq.SEZALNK2 can be cataloged into the MVS master catalog. You may omit them from the MVS master catalog if you identify them to include a volume specification as in:

   ```
   TCPIP.SEZALOAD(WTLTCP),
   TCPIP.SEZALNK2(WTLTCP)
   ```

   If the three data sets mentioned were renamed during the installation process, then use these names instead.

3. PROGnn or IEAAPFxx

   Add the following TCP/IP libraries for APF authorization:

   – hlq.SEZATCP
   – hlq.SEZADSIL
   – hlq.SEZALOAD
   – hlq.SEZALNK2
   – hlq.SEZALPA
   – SYS1.MIGLIB

4. IEFSSNxx

TNF and VMCF are required for CS for z/OS IP. Add the subsystem definitions for the MVS address spaces of TNF and VMCF as follows:

- If you choose to use restartable VMCF and TNF:
  - TNF
  - VMCF
- If you will not be using restartable VMCF and TNF:
  - TNF,MVPTSSI
  - VMCF,MVPXSSI,*nodename*

  The *nodename* should be set to the MVS NJE node name of this MVS system. It is defined in the JES2 parameter member of SYSx.PARMLIB:

```
NJEDEF   ....
         OWNNODE=03,
         ....

NO3    NAME=SC30,SNA,NETAUTH
```

  Make sure that the hlq.SEZALOAD definition has been added to LNKLSTxx and the library itself has been APF authorized before you make this update. z/OS initializes the address spaces of the TNF and VMCF subsystems during IPL as part of the master scheduler initialization.

5. SCHEDxx

   We need to specify certain CS for z/OS IP modules as privileged modules in MVS. The entries below are present in the IBM-supplied program properties table (PPT). However, if your installation has a customized version of the PPT, ensure these entries are present:

   - For CS for z/OS IP

```
PPT PGMNAME(EZBTCPIP) KEY(6) NOCANCEL PRIV NOSWAP SYST LPREF SPREF
```

   - If you use restartable VMCF and TNF

```
PPT PGMNAME(MVPTNF) KEY(0) NOCANCEL NOSWAP PRIV SYST
PPT PGMNAME(MVPXVMCF) KEY(0) NOCANCEL NOSWAP PRIV SYST
```

   - For NPF

```
PPT PGMNAME(EZAPPFS) KEY(1) NOSWAP
PPT PGMNAME(EZAPPAAA) NOSWAP
```

   - For SNALINK

```
PPT PGMNAME(SNALINK) KEY(6) NOSWAP SYST
```

6. COMMNDxx

   VMCF and TNF are required for CS for z/OS IP. If you use restartable VMCF and TNF, procedure EZAZSSI must be run during your IPL sequence (EZAZSSI starts VMCF and TNF). Either use your operation's automation software to start EZAZSSI, or add a command to your COMMNDxx member in SYSx.PARMLIB:

```
COM='S EZAZSSI,P=your_node_name'
```

   The value of variable P defaults to the value of the MVS symbolic &SYSNAME. If your node name is the same as the value of &SYSNAME, then you can use the following command instead:

```
COM='S EZAZSSI'
```

   When the EZAZSSI address space starts, a series of messages is written to the MVS log indicating the status of VMCF and TNF; then the EZAZSSI address space terminates.

Once VMCF and TNF have initialized successfully, you can start your TCP/IP system address spaces.

7. IKJTSOxx

   We also need to specify CS for z/OS IP modules as authorized for TSO commands. Update the IKJTSOxx member by adding the following to the AUTHCMD section: MVPXDISP, NETSTAT, TRACERTE, RSH, LPQ, LPR, and LPRM.

8. IEASYSxx

   Review your CSA and SQA specifications and verify that the numbers allocated are sufficiently large enough to prevent getmain errors.

```
IEASYSxx:  CSA(3000,250M)
IEASYSxx:  SQA(8,448)
```

9. IVTPRMxx

   Review the computed CSM requirements to reflect z/OS VTAM and CS for z/OS IP usage:

   – IVTPRMxx: FIXED MAX(120M)
   – IVTPRMxx: ECSA MAX(30M)

10. CTIEZBxx

    Copy the following member to SYSx.PARMLIB from hlq.SEZAINST for use with CTRACE:

    CTIEZB00 can be customized to include a different sized buffer. The default buffer size is 8 MB. This should be increased to 32 MB to allow the capture of debugging information. We made a new member, CTIEZB01, with the buffer size change. You can read more about the use of component tracing (CTRACE) in *z/OS V1R2.0 CS: IP Diagnosis*, GC31-8782, in *z/OS V1R7.0 CS: IP Migration*, GC31-8773, and in Appendix A, “Component trace (CTRACE)” on page 209.

11. BPXPRMxx

    – In addition to defining the UNIX Physical File Systems you must ensure that the ports enabled on the system are consistent with what is defined in the PROFILE.TCPIPdata set. This is illustrated in Example 3-2.

Example 3-2 BPXPRMxx member with port range provided by a single stack environment

```
/* IPv4 support
 NETWORK DOMAINNAME(AF_INET)
         DOMAINNUMBER(2)
         MAXSOCKETS(25000)
         TYPE(INET)
         INADDRANYPORT(10000) 8
         INADDRANYCOUNT(2000) 8
* IPv6 support
 NETWORK DOMAINNAME(AF_INET6)
        DOMAINNUMBER(19)
        TYPE(INET)
```

Ensure that the INADDRANYPORT 8 assignment does not conflict with PORT assignments in the TCPIP.PROFILE data set.

Note: The OpenEdition ENTRYPOINT for CS for z/OS IP is EZBPFINI. If you have the incorrect value in BPXPRMxx member, you may see messages such as EZZ4203I or abend codes such as S806.

– Review the values specified in BPXPRMxx for MAXPROCSYS, MAXPROCUSER, MAXUIDS, MAXFILEPROC, MAXPTYS, MAXTHREADTASKS, and MAXTHREADS.

12. IFAPRDxx or PROGxx

    To add product and feature information in a z/OS environment.

SYS1.PROCLIB or the PROCLIB you use for your TCP/IP JCL procedures

These are:

1. If you choose to use restartable VMCF and TNF, add procedure EZAZSSI:

```
//EZAZSSI PROC P=''
//STARTVT EXEC PGM=EZAZSSI,PARM=&P
//STEPLIB DD   DSN=hlq.SEZATCP,DISP=SHR
```

2. Update your TCP/IP startup JCL procedure. The sample for the CS for z/OS IP procedure is in hlq.SEZAINST(TCPIPROC).

The PROCLIB you use for your TSO logon procedures

Update your TSO logon procedures by adding the TCP/IP help data set SYS1.HELP to the //SYSHELP DD concatenation. Optionally, add the //SYSTCPD DD statement to your logon procedures.

Add hlq.SEZAMENU to the //ISPMLIB DD concatenation and hlq.SEZAPENU to the //ISPPLIB DD and the //ISPTLIB DD concatenations.

3.4.2 TCP/IP server functions

Each CS for z/OS IP server relies on the use of a security manager such as RACF. Several servers provide some built-in security functions for additional security. These servers are described in *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 2 - Standard Applications*, SG24-7170.

3.4.3 TCP/IP client functions

The client functions of z/OS Communications Server IP are executed in a TSO environment or a UNIX shell environment. Some functions are also available in other environments, such as batch or started task address spaces.

Any TSO user may execute any TCP/IP command and use a TCP/IP client function to access any other TCP/IP server host via the attached TCP/IP network. If these TCP/IP servers have not implemented adequate password protection, then any TSO client user may log on to these servers and access all data.

3.4.4 UNIX client functions

Certain client functions executed from the UNIX shell environment require superuser authority. The user ID accessing the shell must have an OMVS segment associated with it. RACF considerations for UNIX Client functions in CS for z/OS IP are covered in detail in *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 2 - Standard Applications*, SG24-7170, and *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 4 - Security*, SG24-7172.

Common errors implementing UNIX System Services

In this section, we discuss some implementation problems frequently encountered.

Superuser mode

Certain commands and operations from OMVS or from the ISHELL are authorized only for superusers. There are two alternatives for running as a superuser:

- The user ID may have permanent superuser status. This means that the ID has been created with a UID value of 0. TCP/IP started tasks and some of its servers are also defined with a UID of zero.
- The user ID may have temporary authority for the superuser tasks. The defined UID will have been set up as a non-zero value in RACF, but the user will have been granted READ access to the RACF facility class of BPX.SUPERUSER. Also, RACF provides superuser granularity enhancements to assign functions to users that need them.

If you need only temporary authority to enter superuser mode, then the granting of simple READ permission to the BPX.SUPERUSER facility class will allow the user to switch back and forth between superuser mode and standard mode. You enter **'su'** from the OMVS shell, or you may select SETUP OPTIONS from the ISHELL and specify Option #7 to obtain superuser mode.

The user is then authorized to enter commands authorized for the superuser function from the ISHELL, or he can switch to an OMVS shell he has already signed onto. The basic prompt level, indicated by the $ prompt is changed when in superuser mode to a pound sign (#). The **exit** command takes the user out of superuser mode as well as the OMVS (UNIX) shell. Use of the **whoami** command shows the change of user IDs.

Problems with the home directory

In Figure 3-1, the TSO user attempted unsuccessfully to enter the OMVS shell interface from ISPF; the user has an OMVS segment defined but another problem occurs. The user entered the TSO OMVS command to enter the UNIX environment and received the response shown in Figure 3-1.

```
FSUM2078I No session was started.  The home directory for this TSO/E
user does not exist or cannot be accessed, +
FSUM2079I Function = sigprocmask, return value = FFFFFFFF, return code
code = 9C reason code = 0507014D
```

Figure 3-1 Error executing the TSO OMVS command

This error occurred because we neglected to define or authorize the home directory that is associated with the user in to the OMVS segment. You can see what the home directory is supposed to be with the RACF command **listuser** (if you have the RACF authorization to use the command). However, you still get access to the z/OS file even though the message was displayed.

A similar problem occurs when trying to access the ISHELL environment. See Figure 3-2.

```
Errno=9Cx Process Initialization error; Reason=0507014D The dub fail
due to an error with the initial home directory.  Press Enter to
continue.
```

Figure 3-2 Error executing in the ISHELL

In both cases the user had an OMVS segment defined in RACF. However, we neglected to define or authorize the home directory that had been associated with the user in his OMVS segment. (You can see what this home directory is supposed to be with the RACF command **listuser**.) Authorization is provided with the permission bits.

The same symptom shows up for users without an OMVS segment defined if the BPX.DEFAULTUSER facility has been activated with an inaccessible home directory.

UNIX permission bits

You have already read something about setting up appropriate UNIX permission bits. Figure 3-3 shows an example of incorrect permission bits set for a user.

```
ICH408I USER(CS01 ) GROUP(WTCRES ) NAME(SHERWIN LAKE     ) 703
  /u/CS01 CL(DIRSRCH) FID(01E2D7D3C5E7F34E2B0F000000000003)
  INSUFFICIENT AUTHORITY TO LOOKUP
  ACCESSINTENT(--X)  ACCESS ALLOWED(OTHER ---)
ICH408I USER(CS01 ) GROUP(WTCRES ) NAME(SHERWIN LAKE     ) 704
```

Figure 3-3 Error with UNIX permission bit settings

Although the user had the UNIX permission bit settings of 755 on the /u/cs01/ directory, the permission bits were set at 600 for the /u/ directory. This means you must ensure that all directories in the entire path are authorized with suitable permission bits. After the settings were changed to 755 for the /u/ directory, access to the subdirectory was allowed.

You can display UNIX permission bits from the ISHELL environment or by issuing the command **`ls -alF`** from the shell.

The **`ls -alF`** options indicate that all files should be listed (including hidden files), that the long format should be displayed, and that the flags about the type of file (link, directory, etc.) should be given.

Default search path and symbolic links

The directory search path is specified in the environment variable $PATH. Normally this environment variable is set system-wide in /etc/profile and can be further customized for individual users in $home/.profile. The sample for /etc/profile sets $PATH to:

```
/bin:.
```

It should be expanded to:

```
/bin:/usr/sbin:.
```

or:

```
.:/bin:/usr/sbin
```

depending on whether you want the current directory searched first or last.

The instructions for setting up this user profile are contained in *z/OS V1R7.0 UNIX System Services User's Guide,* SA22-7801, and *z/OS V1R7.0 UNIX System Services Planning*, SA22-7800.

Note: To view the search path that has been established for you, issue **`echo $PATH`** from the shell environment.

A user may attempt to run a simple TCP/IP command such as **`oping`** and receive an error that the command is not found.

```
          BROWSE -- /tmp/cs01
Command ===>
         ******************************** Top of Data ***
         oping: FSUM7351 not found
         ******************************** Bottom of Data *
```

Figure 3-4 Command not found error

In this case you must preface the command with the directory path necessary to locate it:

```
/usr/lpp/tcpip/bin/oping
```

If you experience such a problem, check that the symbolic links are correct. Part of the installation is to run the UNIX MKDIR program to set up the symbolic links for the various commands and programs from their real path to /bin or /usr/sbin, where they can be found using the default search path.

3.5 Verification checklist

The following checklist will help ensure that all z/OS and USS related setup tasks have been completed for the base functions:

1. Have TNF and VMCF initialized successfully?

   Check the console log for a successful start of EZAZSSI, TNF, VMCF.

2. Has the TCP/IP feature of z/OS been enabled or registered in IFAPRDxx?
3. Has a *full-function* OMVS (DFSMS, RACF, zFS) started successfully?
   - Is OMVS active when you issue D OMVS?
   - Is SMS active when you issue D SMS?
   - Have z/OS UNIX file systems been mounted? Verify with D OMVS,F.
   - Is RACF enabled on the system?
4. Have the definitions in BPXPRMxx of SYS1.PARMLIB been made to reflect:
   - The correct stack for the stack or stacks you will be running?
   - The support for dual-mode is defined to support IPv4 and IPv6 (AF_INET and AF_INET6)?
   - The correct CS for z/OS IP proc names?
   - The correct use of INET versus CINET?
   - The correct ENTRYPOINT name for z/OS Communications Server IP versus earlier versions of OE function in TCP/IP (z/OS IP ENTRYPOINT = EZBPFINI)?
   - The mounting of file systems for users? (You can verify with D OMVS,F.)
   - Appropriate values for MAXPROCSYS, MAXPROCUSER, MAXUIDS, MAXFILEPROC, MAXPTYS, MAXTHREADTASKS, and MAXTHREADS?
5. Have z/OS UNIX file systems and directories been created and mounted for the users of the system?
6. Have RACF definitions been put in place? For example:
   - OMVS user IDs and group IDs for your CS for z/OS IP procedures
   - OMVS user IDs and group IDs for your other users, for superusers, for a default user, with definitions for appropriate Facility classes, like BPX.SUPERUSER
   - TCP/IP **VARY** commands

- NETSTAT commands

7. Have you placed the correct definitions in the z/OS data sets? For example:

```
SYSx.LNKLSTxx
SYSx.LPALSTxx
SYSx.SCHEDxx
SYSx.PROGxx
SYSx.IEASYSxx
SYSx.IEFSSNxx
SYSx.IKJTSOxx
SYSx.IVTPRMxx
```

8. Raw sockets require authorization; they run from SEZALOAD and are usually already authorized; if you have moved applications and functions to another library (not recommended), ensure that this library is authorized.
9. The loopback address is now 127.0.0.1 for IPv4 and ::1 for IPv6. However, If you require 14.0.0.0, have you added this to the HOME list?
10. Have you computed CSA requirements to include not only z/OS VTAM, but also CS for z/OS IP?
    - IEASYSxx: CSA(3000,250M) (need to review)
    - IEASYSxx: SQA(8,448) (need to review)
11. Have you computed CSM requirements to include not only z/OS VTAM, but also CS for z/OS IP?
    - IVTPRMxx: FIXED MAX(120M)
    - IVTPRMxx: ECSA MAX(30M)
12. Have you modified the CTRACE initialization member (CTIEZB00) to reflect 32 MB of buffer storage?
13. Have you created CTRACE Writer procedures for taking traces?
14. Have you updated your TCP/IP procedure?
15. Have you updated your other procedures, for example, the FTP server procedure?
16. Have you revamped your TCP/IP Profile to use the new statements and to comment out the old?
    - Have you made provisions to address device connections that are no longer supported?
    - Have you investigated all your connections to ensure to what extent they are still supported? (In some cases, definitions will have changed.)
17. Have your applications that relied on VMCF and IUCV sockets been converted now that those APIs are no longer supported?
18. If you are migrating from a previous release, have you reviewed the Planning and Migration checklist in *z/OS V1R7.0 CS: IP Migration*, GC31-8773, and made appropriate plans to use the sample data sets?
19. Have you reviewed the list and location of configuration data set samples in *z/OS V1R7.0 CS: IP Configuration Reference?*

3.6 Configuring z/OS TCP/IP

A z/OS TCP/IP environment can be very complex. It is controlled using a large variety of settings, including parmlib members, and /etc files for UNIX System Services. Each of which

has a different interface and requires special knowledge to configure. IBM has provided tools to get you started and running with a very basic setup.

z/OS IP Configuration Tools

The z/OS IP Wizard and msys Setup are IBM offerings that may help you configure your z/OS Communications Server for IP environment. Both may be used to generate configuration files. We will use these tools to configure a single stack environment. However, since these tools provide a very basic configuration, additional z/OS statements and commands will be used to provide additions to the TCP/IP configuration.

z/OS IP Configuration Wizard

The z/OS IP Configuration Wizard sets up basic IP configuration for a single stack, including options to include simple instances of OMPROUTE, FTP, and TN3270 servers if selected.

The wizard takes you through a series of screens where you enter information about your z/OS IP configuration: General information such as the host name, the domain name, and the IP address of a name server; information about the type of routing you will use, details about your network connections, and whether you want to configure FTP and TN3270 servers and use dynamic routing protocols. Wherever possible, the wizard uses defaults so they need to be reviewed to ensure they fit your requirements. At the end of the exercise you can save and transfer the output datasets to the z/OS system. We used the wizard and built a basic customized PROFILE and TCPIP.DATA files, based on the information entered on the screens. We then used cut and paste to create the customized files on the z/OS system. The wizard also provides a checklist of tasks that must be completed when configuring z/OS IP. The preparation and completion of the worksheet is recommended. It simplifies the process.

Here are examples of a few screens generated by the wizard and used in the configuration of our single stack environment. The Wizard steps you through the entire configuration process of a basic single stack.

Figure 3-5 indicates the various areas in which the wizard provides assistance for the configuration of a basic single stack environment. You may save the completed process and reload to update at a later date.

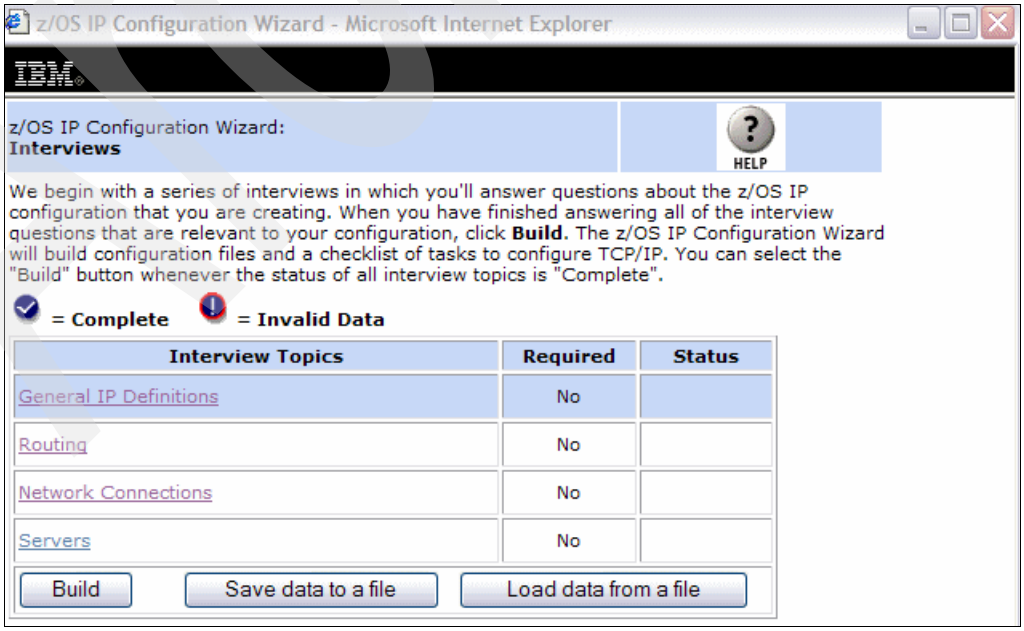


Figure 3-5 Wizard interview topics

When you select **General IP Definitions**, you will be asked to specify your host by a fully qualified name, IP addresses, and (optionally) subnet masks and IP address for the name server.

Once you provide that information, you can use the Routing topic to define your IP routes. Since we are implementing an *application server* in a small network, we selected **Direct routing only** (see Figure 3-6). For more information about routing, refer to Chapter 5, “Routing” on page 139.

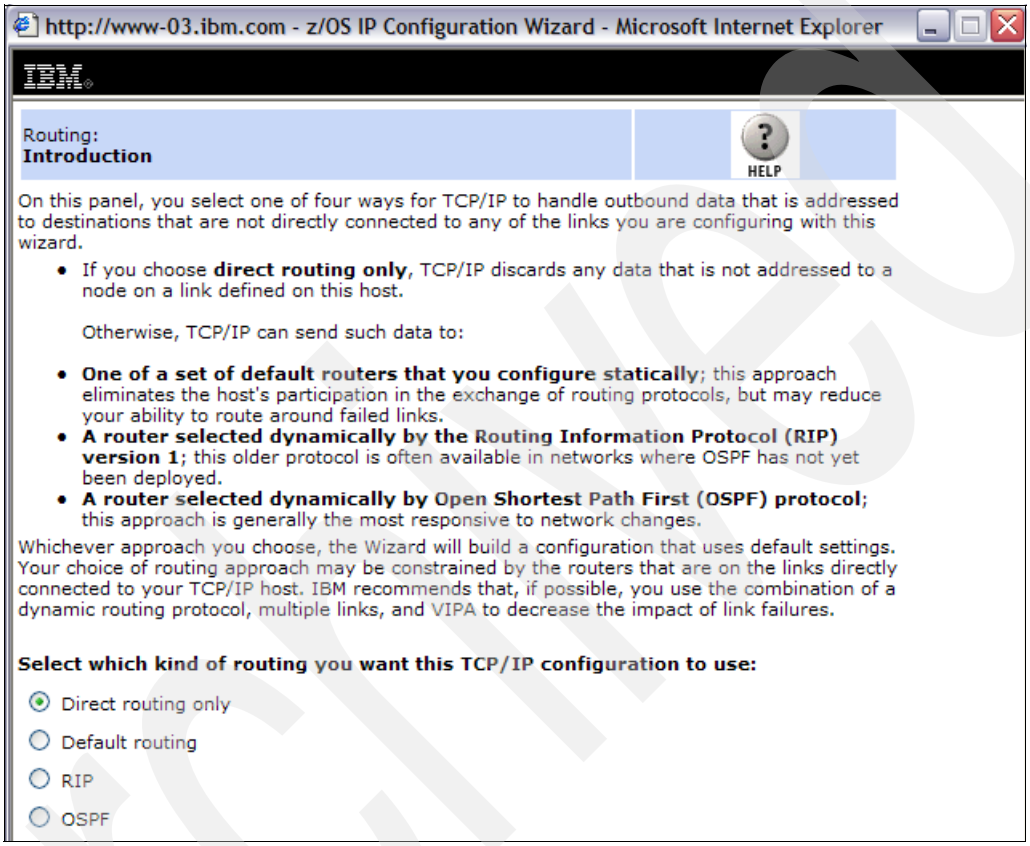


Figure 3-6 Routing definitions

Figure 3-7 on page 70 shows how the LINK and DEVICE for a given connection is defined. In our case, it is for an OSA-Express connection running in QDIO mode.

Figure 3-7 Defining the characteristics of a link and device

Figure 3-8 shows the two server (application) options provided by the wizard. Review the defaults, as they may need further adjustments, because the default resource values may be inadequate for your environment.

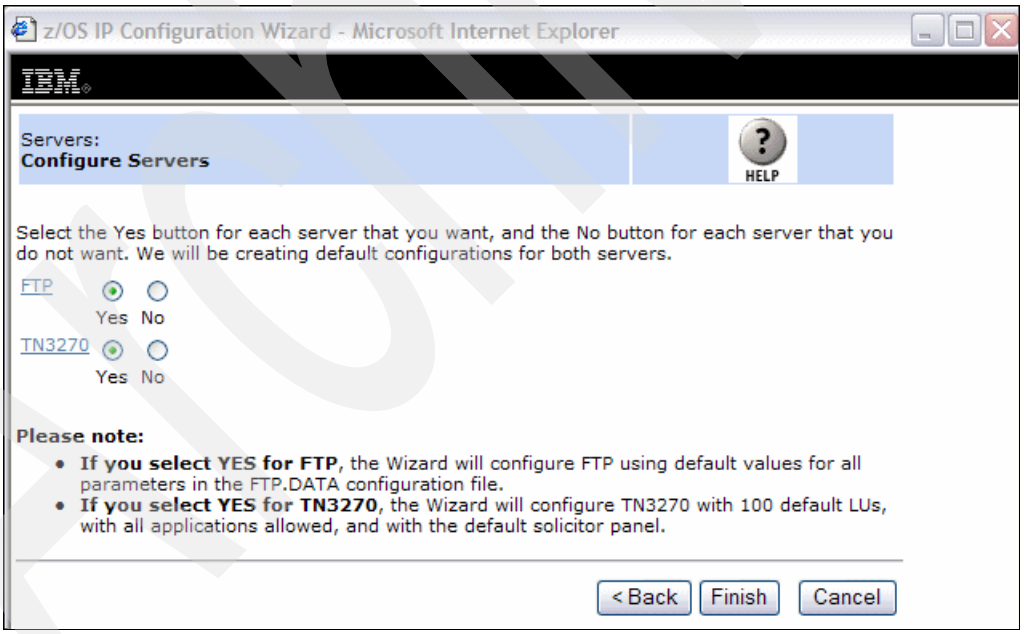


Figure 3-8 Defining servers

The wizard gives the option to save the configuration data so they may be transferred or transcribed to the z/OS host.

The z/OS IP Configuration Wizard can be found at:

`http://www.ibm.com/eserver/zseries/zos/wizards`

Below are two ways you may transfer the output of the wizard to your z/OS system.

Save the file on your workstation as a text (.txt) file, following these directions:

1. With your browser, view the file you want to save.
2. At the top of the Browser panel, click **File**, then choose **Save As**.
3. When the window appears, define the workstation location (Save in) where you want to save the file.
4. Enter the name of the file (File name), using .txt as the file extension.
5. Use the down arrow (Save as type) to pull down the list of possible file types and select **Plain Text**. Click **Save** to finish.
6. Use any file transfer process (NetView ftp or your 3270 emulator) to transfer the saved files. For example, the SEND or RECEIVE function may be used to transfer the data between the z/OS system and desktop in certain scenarios (such as ours).
7. Use the copy-and-paste function on your workstation to move the data to your z/OS system by creating (and saving) a skeletal file and copying and pasting pieces into it.

Note that some symbols you copy and paste might change depending on the fonts installed on the z/OS system.

z/OS msys for Setup

Managed System Infrastructure for Setup (msys for Setup) establishes a central directory for product configuration data and a single interface to that directory.

msys for Setup code runs on a z/OS system. Product configuration data is held in a directory on an LDAP server running on z/OS. Msys parses existing z/OS configuration files and stores the information in the LDAP directory.

The configuration data stored in the LDAP directory can be viewed and updated, and new configuration data added, via the windows of the msys Windows NT® application. To enable this, the Windows NT workstation has IP connections to the LDAP server and the z/OS system. Many products, such as CS for z/OS IP, can be configured using the same msys application.

The current version of msys for Setup allows you to set up a basic TCP/IP system. The windows prompt you for information about your z/OS IP system, which is then stored in the LDAP directory. msys for Setup can then be used to generate TCP/IP configuration files PROFILE.TCPIP, TCPIP.DATA, and OMPROUTE.CONF based on the information in the LDAP directory. DB2® and LDAP are prerequisites.

For further information about msys for Setup see *z/OS V1R4.0 msys for Setup User's Guide*, SC33-7985.

3.6.1 Configuration features

z/OS Communications Server IP continues to be enhanced with new features, enhancements, and defaults, so if you are migrating from a previous release you should consult with the migration guide for your particular release from which you are migrating. Our TCP/IP scenarios will include some of the more current features.

For further details, refer to *z/OS Communications Server: New Function Summary, Version 1 Release 7*, GC31-8771-01; and the *z/OS Communications Server: IP Configuration Guide, Version 1 Release 7*, SC31-8775-07.

PROFILE.TCPIP

Before you start your TCP/IP stack, you must configure the operational and address space characteristics.

These definitions are entered into a PROFILE configuration data set that is read by the TCP/IP address space during initialization.

A sample PROFILE.TCPIP configuration file is provided in hlq.SEZAINST(SAMPPROF).

The PROFILE data set contains the following major groups of TCP/IP configuration parameters:

- Operating characteristics
- Reserved port number definitions
- Network connectivity definitions
- Telnet definitions
- Network routing definitions

For more information about TCP/IP Telnet definitions, see *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 2 - Standard Applications*, SG24-7170.

Detailed information about TCP/IP connectivity and routing definitions can be found in Chapter 4, “Connectivity” on page 101, and Chapter 5, “Routing” on page 139, respectively.

3.6.2 VTAM Resource

As mentioned in the introduction, VTAM provides the Data Link Control layer (Layer 2 of the OSI model) for TCP/IP, including support of the Multi-Path Channel (MPC) interfaces. MPC protocols are used to define the DLC layer for OSA-Express devices in QDIO. OSA-Express QDIO connections are configured via a TRLE definition. All TRLEs are defined as VTAM major nodes. For further information about MPC-related devices/interfaces refer to Chapter 4, “Connectivity” on page 101.

A TRLE definition we used for our OSA-Express in QDIO mode is shown in Example 3-3.

Example 3-3 TRLE VTAM major node definition for device OSA2080

```
OSA2080  VBUILD TYPE=TRL
OSA2080P TRLE  LNCTL=MPC,
               READ=2080,
               WRITE=2081,
               DATAPATH=(2082-2087),
               PORTNAME=OSA2080,    1
               MPCLEVEL=QDIO
```

SInce VTAM provides the DLC layer for TCP/IP, then VTAM must be started before TCP/IP. The major node (in our case, OSA2080) should be activated when VTAM is initializing. This will ensure the TRLE is active when the TCP/IP stack is started. This is accomplished by placing an entry for OSA2080 in the VTAM startup list ATCCONxx. The portname 1 (Example 3-3) must also be the same name as the device name defined in TCP/IP PROFILE data set on the DEVICE and LINK statements.

3.6.3 PROFILE.TCPIP parameters

The syntax for the parameters in the PROFILE can be found in *z/OS V1R7.0 CS: IP Configuration Reference,* SC31-8776.

Most PROFILE parameters required in a basic configuration have default values that will allow the stack to be initialized and ready for operation. There are, however, a few parameters that must be modified or must be unique to the stack.

AUTOLOG considerations

The purpose of the AUTOLOG statement is to start all procedures specified. AUTOLOG also monitors procedures started under its auspices, and will restart a procedure that terminates for any reason unless NOAUTOLOG is specified on the PORT statement.

For UNIX servers some special rules apply. If the procedure name on the AUTOLOG statement is eight characters long, no jobname needs be specified. If the procedure name on the AUTOLOG statement is less than eight characters long and the job spawns listener threads with different names, you may have to specify the JOBNAME parameter and ensure that the jobname matches that coded on the PORT statement. In the following example, jobname FTPDE1 on the PORT statement matches JOBNAME on the AUTOLOG statement:

```
PORT
    20   TCP  OMVS
    21   TCP  FTPDE1

AUTOLOG 1
    FTPDE1 JOBNAME FTPDE1 ; FTP Server
ENDAUTOLOG
```

HOME

The TCP/IP stack uses an IP address of 127.0.0.1 for IPv4 and ::1 for Pv6 as the loopback interfaces. If there is a requirement to represent the loopback IP address of 14.0.0.0 for compatibility with earlier TCP/IP versions, you must code an entry in the HOME statement. The link label specified is LOOPBACK and you can define multiple IP addresses with the LOOPBACK interface. For example:

```
HOME
   14.0.0.0   LOOPBACK
```

The **onetstat -h** command displays the home address assignments of the currently running stack. The display results are similar to the NETSTAT HOME command in TSO and the z/OS command D TCPIP,procname,NETSTAT,HOME. There is an additional field, called the Flg field, that indicates which interface is the primary interface. The primary interface is the address that is inserted as the source address in an IP header when communicating to a destination through an indirect route. The primary interface is the first entry in the HOME list in the PROFILE.TCPIP definitions unless the PRIMARYINTERFACE parameter is specified.

Example 3-4 Netstat home display

```
D TCPIP,TCPIPE,N,HOME
EZD0101I NETSTAT CS V1R7 TCPIPE 751
HOME ADDRESS LIST:
LINKNAME:   STAVIPA1LNK
  ADDRESS:  10.10.10.210
    FLAGS:  PRIMARY
LINKNAME:   OSA2080LNK
  ADDRESS:  10.10.2.212
    FLAGS:
LINKNAME:   OSA20A0LNK
  ADDRESS:  10.10.3.213
    FLAGS:
LINKNAME:   OSA20C0LNK
  ADDRESS:  10.10.2.214
    FLAGS:
```

```
LINKNAME:   OSA2OEOLNK
  ADDRESS:  10.10.3.215
    FLAGS:
LINKNAME:   EZASAMEMVS
  ADDRESS:  10.10.20.130
    FLAGS:
LINKNAME:   IQDIOLNK0A0A1482
  ADDRESS:  10.10.20.130
    FLAGS:
LINKNAME:   LOOPBACK
  ADDRESS:  127.0.0.1
    FLAGS:
INTFNAME:   LOOPBACK6
  ADDRESS:  ::1
    TYPE:   LOOPBACK
    FLAGS:
12 OF 12 RECORDS DISPLAYED
END OF THE REPORT
```

INTERFACE statement

The HOME statement in combination with the DEVICE and LINK statements are used to define IPv4 links. The INTERFACE statement is used to define IPv6 interfaces. This statement combines the definitions of the DEVICE, LINK, and HOME into a single statement for IPv6. Refer to the *z/OS Communications Server: IP Configuration Guide, Version 1 Release 7*, SC31-8775-07; and *z/OS Communications Server: IP Configuration Reference, Version 1 Release 7*, SC31-8776-08, for further details.

IPCONFIG

All IPv4 features are defined within IPCONFIG. There is a separate configuration section for IPv6 parameters. Refer to the *z/OS Communications Server: IP Configuration Reference, Version 1 Release 7*, SC31-8776-08, for details.

IPCONFIG6

All IPv6 features are defined within IPCONFIG6. The same parameters provided for IPv6 exist for IPv4. Refer to the *z/OS Communications Server: IP Configuration Reference, Version 1 Release 7*, SC31-8776-08, for details of the parameters.

3.6.4 Locating PROFILE.TCPIP

The following search order is used to locate the PROFILE.TCPIP configuration file:

1. //PROFILE DD DSN=.... (Explicit Allocation)

   ```
   //PROFILE DD DSN=TCPIPE.TCPPARMS(PROFE30)
   ```

2. jobname.nodename.TCPIP (Implicit Allocation)
3. hlq.nodename.TCPIP (Implicit Allocation)
4. jobname.PROFILE.TCPIP (Implicit Allocation)
5. hlq.PROFILE.TCPIP (Implicit Allocation)

The PROFILE must exist. Otherwise, the TCP/IP address space will terminate abnormally with the message:

```
EZZ0332I DD:PROFILE NOT FOUND. CONTINUING PROFILE SEARCH
EZZ0325I INITIAL PROFILE COULD NOT BE FOUND
```

We recommend using the //PROFILE DD statement in the TCP/IP address space JCL procedure to explicitly allocate the PROFILE data set.

3.6.5 TCP/IP configuration data set names

This topic is described in *z/OS V1R7.0 CS: IP Configuration Guide*, SC31-8775. We strongly recommend that you read the information about data set names in this book, before you decide on your data set naming conventions.

The purpose here is to give an introduction to the data set naming and allocation techniques used by z/OS Communication Server IP.

With z/OS Communication Server IP you have a choice for some of the configuration data sets, whether they should be allocated implicitly or explicitly. Additionally, you need to ensure that not only MVS functions find the appropriate data sets, but also that the z/OS UNIX functions do as well.

- Implicit

  The name of the configuration data set is resolved at runtime based on a set of rules (the search order) implemented in the various components of TCP/IP. When a data set name has been resolved, the TCP/IP component uses the dynamic allocation services of MVS or of UNIX System Services to allocate that configuration data set. See *z/OS V1R7.0 CS: IP Configuration Guide,* SC31-8775, for details.

  These are some of the data sets (or files) that can only be implicitly allocated in an z/OS Communication Server IP:

  ```
  hlq.ETC.PROTO
  hlq.ETC.RPC
  hlq.HOSTS.ADDRINFO
  hlq.HOSTS.SITEINFO
  hlq.SRVRFTP.TCPCHBIN
  hlq.SRVRFTP.TCPHGBIN
  hlq.SRVRFTP.TCPKJBIN
  hlq.SRVRFTP.TCPSCBIN
  hlq.SRVRFTP.TCPXLBIN
  hlq.STANDARD.TCPCHBIN
  hlq.STANDARD.TCPHGBIN
  hlq.STANDARD.TCPKJBIN
  hlq.STANDARD.TCPSCBIN
  hlq.STANDARD.TCPXLBIN
  ```

  In the above data set names, hlq is determined using the following search sequence:

  – User ID or jobname
  – DATASETPREFIX value (or its default of TCPIP), defined in TCPIP.DATA

  Dynamically allocated data sets can include a mid-level qualifier (MLQ), for example, a node name, or a function name:

  – For data sets containing a PROFILE configuration file:

    ```
    xxxx.nodename.zzzz
    ```

  – For data sets containing a translate table used by a particular TCP/IP server:

    ```
    xxxx.function_name.zzzz (for the FTP server the function_name is SRVRFTP)
    ```

  Data set SYS1.TCPPARMS(TCPDATA) can be dynamically allocated if it contains the TCPIP.DATA configuration file.

- Explicit

For some of the configuration files, you can tell TCP/IP which files to use by coding DD statements in JCL procedures, or by setting UNIX environment variables. Again, the various data sets used by TCP/IP functions and their resolution method are described in *z/OS V1R7.0 CS: IP Configuration Guide,* SC31-8775.

TCPIP.DATA

The TCPIP.DATA configuration data set is the anchor configuration data set for the TCP/IP stack and all TCP/IP servers and clients running on that stack. In z/OS Communication Server IP, you can define the TCPIP DATA parameters in an z/OS UNIX file system file or in an MVS data set. The TCPIP.DATA configuration data set is read during initialization of *all* TCP/IP server and client functions. They must all access this data set in order to find the basic configuration information, such as the name of the TCP/IP address space (keyword TCPIPJOBNAME), the TCP/IP host name (keyword HOSTNAME), and the data set prefix to use when searching for other configuration data sets (keyword DATASETPREFIX).

Notes:

- The syntax for the parameters in the TCPIP DATA file can be found in *z/OS V1R7.0 CS: IP Configuration Reference,* SC31-8776.
- A sample TCPIP.DATA configuration file is provided in *hlq*.SEZAINST(TCPDATA).

The TCPIP.DATA file is also known as one of the Resolver configuration files. In fact, the name is now more commonly used to refer to this important file in the UNIX System Services environment because the sockets library contains a component called the Resolver. In the UNIX environment you use the /etc/resolv.conf file for the same purpose as you use TCPIP.DATA in an MVS environment.

Testing TCPIP.DATA

HOMETEST is a TSO command that can be used to test your active and current TCPIP.DATA specifications. The HOMETEST command is meant to be issued only from TSO; therefore, it uses the native MVS search order when locating configuration data sets and it uses Resolver for doing name to IP address resolutions.

Example 3-5 TCPIP.DATA with HOMETEST

```
EZAO619I Running IBM MVS TCP/IP CS V1R7 TCP/IP Configuration Tester

EZAO602I TCP Host Name is: WTSC30E

EZAO605I Using Host Tables to Resolve WTSC30E
EZAO611I The following IP addresses correspond to TCP Host Name: WTSC30E
EZAO612I 10.10.10.210

EZAO614I The following IP addresses are the HOME IP addresses defined in
PROFILE.TCPIP:
EZAO615I 10.10.10.210
EZAO615I 10.10.2.212
EZAO615I 10.10.3.213
EZAO615I 10.10.2.214
EZAO615I 10.10.3.215
EZAO615I 10.10.4.214
EZAO615I 10.10.4.215
EZAO615I 10.10.5.216
EZAO615I 10.10.20.130
EZAO615I 10.10.20.130
EZAO615I 127.0.0.1

EZAO618I All IP addresses for WTSC30E are in the HOME list!
```

```
EZAO622I Hometest was successful - all Tests Passed!
```

Configuring the SITE table (HOSTS.LOCAL)

You can set up the local hosts file to support local host name resolution. If you use only the local hosts file for this purpose, your sockets applications will only be able to resolve names and IP addresses that appear in your local hosts file.

If you need to resolve host names outside your local area, you can configure the Resolver to use a domain name server (see the NSINTERADDR statement in the TCPIP.DATA configuration file). If you use a domain name server, you do not need to set up any host definitions in your Resolver configuration, but you may still do so. If you have configured your Resolver to use a name server, it will always try to do so, unless your applications were written with a RESOLVE_VIA_LOOKUP symbol in the source code.

In our environment we decided to use the Resolver for all our network resolution requests and therefore chose to implement IPNODES rather than the HOST.LOCAL file.

Refer to Chapter 2, “The Resolver” on page 17, and the *z/OS Communications Server: IP Configuration Guide, Version 1 Release 7*, SC31-8775-07; and *z/OS Communications Server: IP Configuration Reference, Version 1 Release 7*, SC31-8776-08, for further explanation and details.

3.7 Steps for configuring stack

The following steps apply to configuring the TCP/IP stack data sets to support base functions:

1. Decide on a stack name and an associated stack DATASETPREFIX. For example, we chose TCPIPE as the stack name and TCPIPA as the DATASETPREFIX.
2. Decide on network connections for the stack. We configured two OSA-EXPRESS features, each card having two ports. We configured four connections across the two features. For redundancy we assigned two pairs of interfaces, each pair using one port per feature and each pair attached to the same VLAN. This facilitates ARP Takeover, which is recommended in a flat network (no routing daemon such as OMPROUTE).

Example 3-6 Link assignments

```
DEVICE OSA2080  MPCIPA
LINK   OSA2080LNK  IPAQENET      OSA2080 VLANID 10
DEVICE OSA20A0  MPCIPA
LINK   OSA20A0LNK  IPAQENET      OSA20A0 VLANID 11
DEVICE OSA20C0  MPCIPA
LINK   OSA20C0LNK  IPAQENET      OSA20C0 VLANID 10
DEVICE OSA20E0  MPCIPA
LINK   OSA20E0LNK  IPAQENET      OSA20E0 VLANID 11

HOME
10.10.10.210   STAVIPA1LNK
10.10.2.212    OSA2080LNK
10.10.3.213    OSA20A0LNK
10.10.2.214    OSA20C0LNK
10.10.3.215    OSA20E0LNK
```

Figure 3-9 on page 78 illustrates the OSA2080LNK and OSA20C0 pair connecting to the same VLAN via two different OSA-Express cards. The same logic applies to the OSA20A0

and OSA20E0 pair. OSA2080LNK and OSA20A0 share one 2-port feature and OSA20C0 and OSA20E0 share the other 2-port feature.

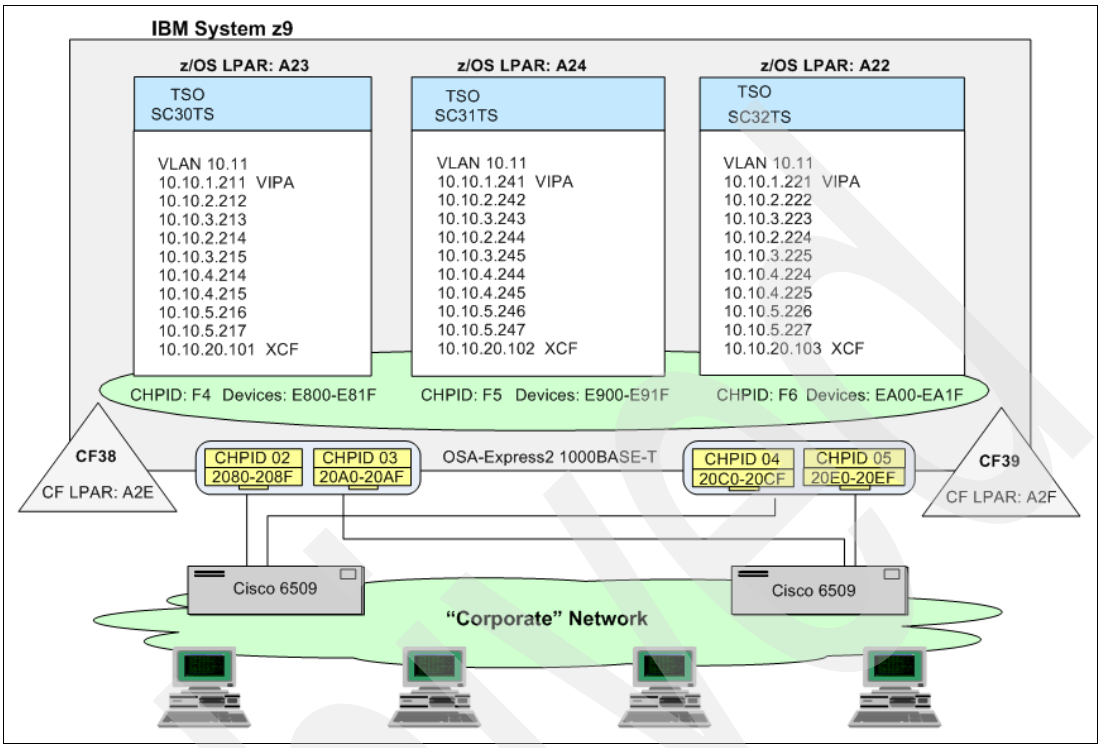


Figure 3-9 Network diagram

3. Assign static routing paths (see Example 3-7) for traffic to the network using the OSA-Express ports and IP address based on the network diagram.

Example 3-7 Defined static routes

```
BEGINRoutes
; Direct Routes - Routes that are directly connected to my interfaces
;     Destination     Subnet Mask   First Hop Link Name     Packet Size ;
 ROUTE 10.10.3.0       255.255.255.0 =         OSA20A0LNK    mtu defaultsize
 ROUTE 10.10.2.0       255.255.255.0 =         OSA20C0LNK    mtu defaultsize
 ROUTE 10.10.3.0       255.255.255.0 =         OSA20E0LNK    mtu defaultsize
; Default Route - All packets to an unknown destination are routed
;through this route.
;     Destination     First Hop    Link Name   Packet Size
 ROUTE DEFAULT         10.10.3.1    OSA20A0LNK  mtu defaultsize
 ROUTE DEFAULT         10.10.2.1    OSA20C0LNK  mtu defaultsize
 ROUTE DEFAULT         10.10.3.1    OSA20E0LNK  mtu defaultsize
ENDRoutes
```

We used multiple ROUTE DEFAULT statements for OSA-Express port redundancy.

4. Allocate the TCPPARMS library to be used for explicitly allocated configuration data sets for the stack, or create a new member in your existing TCPPARMS library. For example, we allocated TCPIPE.TCPPARMS(DATAE30).

5. Alter your SYS1.PARMLIB(BPXPRMxx). We modified our member as shown in Example 3-8 on page 79.

Example 3-8 BPXPRMxx member

```
NETWORK DOMAINNAME(AF_INET)  1
        DOMAINNUMBER(2)
        MAXSOCKETS(10000)
        TYPE(INET)     2
        INADDRANYPORT(10000)    3
        INADDRANYCOUNT(2000) 3
NETWORK DOMAINNAME(AF_INET6) 4
DOMAINNUMBER(19)
MAXSOCKETS(10000)
TYPE(INET)
```

1 defines the socket family for IPv4.

2 defines a single stack environment.

3 defines the range of IP addresses for this stack.

4 defines the socket family for IPv6 (supporting dual-mode).

6. Create a PROFILE member in TCP.TCPPARMS. To help standardize the data sets across multiple z/OS LPARs make use of INCLUDES and system SYMBOLS. See Appendix B, “Additional parameters and functions” on page 225, for examples of our profile. In our environment we had three z/OS LPARs (SC30, SC31,and SC32), each with a single stack.
7. Create a TCPDATA member in TCPIPE.TCPPARMS(DATAE30).
8. Create a new system address space JCL procedure (see Example 3-9).

Example 3-9 Address space JCL procedure (SC30)

```
//TCPIPE    PROC PARMS='CTRACE(CTIEZB00),IDS=00',
//             PROFILE=PROFE&SYSCLONE.,TCPDATA=DATAE&SYSCLONE 1
//TCPIPE    EXEC PGM=EZBTCPIP,REGION=0M,TIME=1440,
//             PARM='&PARMS'
//SYSPRINT DD SYSOUT=*,DCB=(RECFM=VB,LRECL=132,BLKSIZE=136)
//ALGPRINT DD SYSOUT=*,DCB=(RECFM=VB,LRECL=132,BLKSIZE=136)
//CFGPRINT DD SYSOUT=*,DCB=(RECFM=VB,LRECL=132,BLKSIZE=136)
//SYSOUT   DD SYSOUT=*,DCB=(RECFM=VB,LRECL=132,BLKSIZE=136)
//CEEDUMP  DD SYSOUT=*,DCB=(RECFM=VB,LRECL=132,BLKSIZE=136)
//SYSERROR DD SYSOUT=*
//PROFILE  DD DISP=SHR,DSN=TCPIPE.TCPPARMS(&PROFILE.)
//SYSTCPD  DD DSN=TCPIPE.TCPPARMS(&TCPDATA.),DISP=SHR
```

1 illustrates the use of SYSTEM SYMBOLS.

9. Example 3-9 also shows the SYSTCPD DD statement pointing to TCPIPE.TCPPARMS(DATAE30).
10. Define your TRLEs in VTAM. Remember to include it in the VTAM startup list in ATCCONxx.
11. Create server-specific configuration data sets, such as FTP.DATA (see Example 3-10).

Example 3-10 Server-specific configuration data set

```
StartDir        MVS
;
; File and disk parameters.
;
Primary         5          ; Primary allocation is 5 tracks
Secondary       2          ; Secondary allocation is 2 tracks
Directory       15         ; PDS allocated with 15 directory blocks
Lrecl           128        ; Logical record length is 128 bytes
```

```
BlockSize       6144       ; Block size is 6144 bytes
AutoRecall      true       ; Migrated HSM files recalled automatically
AutoMount       true       ; Nonmounted volumes mounted automatically
DirectoryMode   false      ; Use all qualifiers (Datasetmode)
SpaceType       TRACK      ; Data sets allocated in tracks
Recfm           FB         ; Fixed blocked record format
Filetype        SEQ        ; File Type = SEQ (default)
RDW             false      ; Do not retain RDWs as data
```

Note: If server-specific configuration data sets can be explicitly allocated using DD statements, we recommend that you create the configuration data set as a member in the stack-specific TCPPARMS library. If the data set has to be implicitly allocated, remember to create it with the stack-specific data set prefix.

12. Create required RACF definitions to assign started task user IDs to new address spaces. This will be done by the RACF administrator.
13. If you are using a domain name server, ensure that it is updated with your new host name and address.
14. If you are not using a domain name server, edit your TCPIPE.TCPPARMS(HOSTS) and add your new host name and address.

Depending on your system's management strategy, you may optionally create different USS table and VTAM definitions to distinguish among the different stacks.

3.8 Starting the z/OS Communications Server IP

If you IPL your z/OS system with PARMLIB definitions similar to our environment you should get messages similar to those shown in Figure 3-10 on page 81 and Figure 3-11 on page 82. These are some of the messages that may be used to verify the accuracy of the current environment customization datasets used in z/OS UNIX and TCP/IP initialization. Note how messages issued by z/OS UNIX begin with the prefix *BPX*.

```
/* the current UNIX System Services configuration file for this start up */
IEE252I MEMBER BPXPRM7A FOUND IN SYS1.PARMLIB
IEE252I MEMBER IGDSMS00 FOUND IN SYS1.PARMLIB
IEE536I SMS    VALUE 00 NOW IN EFFECT
IGD020I SMS IS NOW ACTIVE
BPXF013I FILE SYSTEM WTSCPLX5.SC30.SYSTEM.ROOT
WAS SUCCESSFULLY MOUNTED

/* successful initialization of the LE environment critical to applications using those libraries */
CEE3739I LANGUAGE ENVIRONMENT INITIALIZATION COMPLETE

/* the supported TCP/IP environments for this startup are IPv4 and IPv6 */
BPXF203I DOMAIN AF_UNIX WAS SUCCESSFULLY ACTIVATED.
BPXF203I DOMAIN AF_INET WAS SUCCESSFULLY ACTIVATED.
BPXF203I DOMAIN AF_INET6 WAS SUCCESSFULLY ACTIVATED.

/* the availability of the Resolver for applications who need network resolution performed by the Resolver */
BPXF224I THE RESOLVER_PROC, RESOLVER, IS BEING STARTED
EZZ9291I RESOLVER INITIALIZATION COMPLETE

/* successful initialization of the UNIX environment */
BPXI004I OMVS INITIALIZATION COMPLETE
EZZ4202I Z/OS UNIX - TCP/IP CONNECTION ESTABLISHED FOR TCPIP

/* this TCP/IP environment (stack) has established communication with the other TCP/IP stacks in the sysplex */
EZD1176I TCPIP HAS SUCCESSFULLY JOINED THE TCP/IP SYSPLEX GROUP

/* device (OSA2080) is inserted into the network, bound to the upper protocol layer and ready to process network traffic */
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE OSA2080

/* the Full functions of this TCP/IP stack are now available */
EZB6473I TCP/IP STACK FUNCTIONS INITIALIZATION COMPLETE.
EZAIN11I ALL TCPIP SERVICES FOR PROC TCPIP ARE AVAILABLE.
```

Figure 3-10 Initialization of UNIX System Services and TCP/IP

The first important message indicates whether the right UNIX customization data set is used. In our environment it is BPXPRM7A. This contains the *root* system upon which all other file systems are mounted and is critical for the current establishment of the correct UNIX Systems Services environment.

The following set of messages shows the initialization of SMS and is a critical component since the zFSs are SMS managed. Note that the files systems are mounted subsequently starting with the root.

The next message indicates that the LE environment is available to be exploited by TCP/IP LOTUS, WebSphere®, and parts of the z/OS base as well as Languages such as C/C++ COBOL and others.

The following set of messages indicates the successful establishment of the physical file system and availability for socket services for both IPv4 and IPv6.

The Resolver messages indicate that the resolver process is available to support network resolution, which may be critical to some applications. Note that the initialization of the Resolver is completed before TCP/IP.

The following two messages indicate the successful initialization of the UNIX Systems Services environment and TCP/IP services according to the BPXPRMxx definitions.

Our environment is defined within a sysplex; therefore, message EZD1176I indicates the connectivity to other active TCP/IP stacks within the sysplex.

Initialization of devices must be completed before they achieve READY status (displayed using the NETSTAT DEVLNKS) and connected to the network.

The EZB6473I and EZAIN11I messages are the final initialization messages to complete the successful initialization of the TCP/IP stack.

The following three messages are related to the TN3270E server and are documented because this instance of the TN3270E server is included in the TCP/IP stack. Later we will see the TN3270E server defined within its own z/OS address space.

```
/* initialization of the TN3270 server is successful and is available for sessions */
EZZ0400I TELNET/VTAM (SECOND PASS) BEGINNING FOR FILE: DD:PROFILE
EZZ6003I TELNET LISTENING ON PORT    23
EZZ0403I TELNET/VTAM (SECOND PASS) COMPLETE FOR FILE: DD:PROFILE

/* the automated start of applications defined in AUTOLOG of the TCP/IP profile dataset */
S FTPMVS
S FTPOE
S PORTMAP
S REXECD

/* the FTP application is started and assigned the name defined in the AUTOLOG definition in the Profile dataset */
*/ The User is TCPIP so that he may have the same rights and privileges as TCPIP. All TCP/IP related application */
*/ typically have the same User
IEF695I START FTPOE    WITH JOBNAME FTPOE     IS ASSIGNED TO USER
TCPIP  , GROUP TCPGRP

/* successful initialization of the FTP application */
EZY2702I Server-FTP: Initialization completed at 14:03:10 on
09/08/05.

/* Main task (process) ends and forks the FTPOE1 daemon to handle FTP requests. The suffix digit may be 1-9 */
$HASP395 FTPOE    ENDED
BPXF024I (TCPIP) Sep 8 14:03:10 ftpdY65551": EZYFT41I Server-FTP: 274
process id 65551, server job name FTPOE1.
```

Figure 3-11 Initialization of applications (like FTP) AUTOLOGGED in TCP/IP profile

The messages in Figure 3-11 are those generated by applications that are automatically started with the TCP/IP initialization process. The TN3270 server is started because it is defined within the TCP/IP address space. The FTP server is defined in the AUTOLOG section of the profile data set for this TCP/IP stack. Notice the main task of FTP (FTPOE) ends and the new spawned task (FTPOE1) becomes the current active daemon.

UNIX System Services verification

A few commands can be used to perform a simple verification of the z/OS UNIX environment after an maintenance IPL, for example:

D SMS Verifies the system is running a functional SMS environment (See Figure 3-12 on page 83.)

```
D SMS
IGD002I 16:06:23 DISPLAY SMS 457
SCDS = SYS1.SMS.SCDS
ACDS = SYS1.SMS.ACDS
COMMDS = SYS1.SMS.COMMDS
DINTERVAL = 150
REVERIFY = NO
ACSDEFAULTS = NO
   SYSTEM    CONFIGURATION LEVEL    INTERVAL SECONDS
   SC30      2005/10/03 16:06:23         15
   SC31      2005/10/03 16:06:09         15
   SC32      2005/10/03 16:06:15         15
   WTSCPLX5  ---------- --------         N/A
```

Figure 3-12 SMS display

You see in the output from the SMS display that we are supporting three different LPARs and the captured information pertains to SC30, SC31, and SC32.

Example 3-11 Displaying the OMVS system that is running

```
D OMVS,ASID=ALL
BPXO040I 16.12.11 DISPLAY OMVS 465
OMVS     000E ACTIVE          OMVS=(7A)   1
USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
OMVSKERN BPXOINIT 002F          1          0 MRI--- 14.08.28     4.803 2
  LATCHWAITPID=         0 CMD=BPXPINPR
  SERVER=Init Process                    AF=    0 MF=00000 TYPE=FILE
TCPIP    NFSCLNT  0020   16842754          1 1A---- 14.08.34    39.213
  LATCHWAITPID=         0 CMD=BPXVCMT
TCPIP    NFSCLNT  0020   16842755          1 1R---- 14.08.34    39.213
  LATCHWAITPID=         0 CMD=BPXVCLNY
TCPIP    TCPIP    0034      65540          1 MR---B 14.08.28   124.859
  LATCHWAITPID=         0 CMD=EZBTCPIP
TCPIP    TCPIP    0034   50397189          1 MR---B 14.08.34   124.859  3
  LATCHWAITPID=         0 CMD=EZBTTMST
TCPIP    TCPIP    0034      65545          1 1F---B 14.08.31   124.859
  LATCHWAITPID=         0 CMD=EZASASUB
OMVSKERN INETD1   0030      65550          1 1FI--- 14.08.38      .014
  LATCHWAITPID=         0 CMD=/usr/sbin/inetd /etc/inetd.conf
TCPIP    REXECD   003A      65552          1 1FI--- 14.08.41      .008
  LATCHWAITPID=         0 CMD=RSHD
TCPIP    PORTMAP  0039   33619985          1 1FI--- 14.08.41      .007
  LATCHWAITPID=         0 CMD=PORTMAP
TCPIP    FTPMVS1  0031      65554          1 1FI--- 14.08.41      .015
  LATCHWAITPID=         0 CMD=FTPD
TCPIP    FTPOE1   0036      65555          1 1FI--- 14.08.41      .012
  LATCHWAITPID=         0 CMD=FTPD

TCPIP    TCPIPA   0045   16842776          1 MR---B 09.49.06     8.187 3
  LATCHWAITPID=         0 CMD=EZBTCPIP
TCPIP    TCPIPA   0045   16842777          1 1R---B 09.49.08     8.187
  LATCHWAITPID=         0 CMD=EZBTTSSL
TCPIP    TCPIPA   0045   16842779          1 1F---B 09.49.08     8.187
  LATCHWAITPID=         0 CMD=EZASASUB
```

In Example 3-11 you can see that the OMVS member that is running is related to 1 BPXPRM7A and the initialization process running is as superuser 2 OMVSKERN, and the

PID is 1 (the first process to start). Also note that there is another TCP/IP started task running 3. What is also significant here is that OMVS=DEFAULT is not displayed in the output. In our previous review of the z/OS UNIX environment we mentioned that the z/OS UNIX System Services must be customized in *full-function mode*. The display tells you that, at the very least, your system is not running in default mode (*minimal mode*).

We can also notice the different TPC/IP stacks and tasks associated with them. There is TCPIPA and TCPIP (the default stack), both executing EZBTCPIP.

In Example 3-11 on page 83 also notice multiple tasks are associated with the same RACF user ID, TCPIP. This has the advantage of easier maintenance and system definitions; however, there is the disadvantage of no distinguishing features among messages for individual tasks. Many users of TCP/IP and UNIX System Services would lean towards ease of problem determination and would assign individual RACF user IDs to each OMVS user.

You will find a thorough discussion on the use and implementation of RACF in *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 4 - Security*, SG24-7172.

```
D OMVS,F
 BPXO045I 16.17.08 DISPLAY OMVS 494
 OMVS     000E ACTIVE         OMVS=(7A)
 TYPENAME   DEVICE ----------STATUS----------- MODE  MOUNTED    LATCHES
 AUTOMNT       13 ACTIVE                  RDWR  09/27/2005  L=15
   NAME=*AMD/u                            14.08.27    Q=0
   PATH=/u
   OWNER=SC32     AUTOMOVE=Y CLIENT=N
 TFS        55636 ACTIVE                  RDWR  09/27/2005  L=37
   NAME=/DEV                              14.08.27    Q=0
   PATH=/SC30/dev
   MOUNT PARM=-s 10
   OWNER=SC30     AUTOMOVE=Y CLIENT=N
 TFS        55635 ACTIVE                  RDWR  09/27/2005  L=36
   NAME=/SC30/TMP                           14.08.27    Q=0
   PATH=/SC30/tmp
INPUT ===>                                SCROLL ===> CS
  PATH=/SC30/tmp
  MOUNT PARM=-s 500
  OWNER=SC30     AUTOMOVE=N CLIENT=N
TFS        55467 ACTIVE                  RDWR  09/27/2005  L=32
  NAME=/SC31/TMP                           14.08.27    Q=0
  PATH=/SC31/tmp
  MOUNT PARM=-s 500
  OWNER=SC31     AUTOMOVE=N CLIENT=Y
HFS        80929 ACTIVE                  RDWR  10/02/2005  L=48
  NAME=CS04.HFS                            17.32.04    Q=0
  PATH=/u/cs04
  OWNER=SC30     AUTOMOVE=Y CLIENT=N
HFS        55634 ACTIVE                  RDWR  09/27/2005  L=35
  NAME=OMVS.SC30.VAR                       14.08.27    Q=0
  PATH=/SC30/var
  OWNER=SC30     AUTOMOVE=U CLIENT=N
HFS        55633 ACTIVE                  RDWR  09/27/2005  L=34
  NAME=OMVS.SC30.ETC                       14.08.27    Q=0
  PATH=/SC30/etc
  OWNER=SC30     AUTOMOVE=U CLIENT=N
HFS        55632 ACTIVE                  RDWR  09/27/2005  L=33
  NAME=WTSCPLX5.SC30.SYSTEM.ROOT           14.08.27    Q=0
  PATH=/SC30
  OWNER=SC30     AUTOMOVE=U CLIENT=N
HFS        55466 ACTIVE                  RDWR  09/27/2005  L=31
  NAME=OMVS.SC31.VAR                       14.08.27    Q=0
  PATH=/SC31/var
  OWNER=SC31     AUTOMOVE=U CLIENT=Y
```

Figure 3-13 Display of mounted file systems

Figure 3-13 shows the display of available file systems after the initialization of the z/OS UNIX Systems Services environment. The display should list all the files defined in the mount statement in the BPXPRMxx member, which in our scenario is BPXPRM7A.

Figure 3-14 on page 86 shows some of the files defined in the active BPXPRM7A member for comparative purposes only. We can compare the names defined in the active BPXPRM7A member with the names that are actually active using the D OMVS,F command.

```
  FILESYSTYPE TYPE(TFS) ENTRYPOINT(BPXTFS)

  VERSION('&SYSR1.')
  SYSPLEX(YES)

ROOT  FILESYSTEM('WTSCPLX5.SYSPLEX.ROOT')
      TYPE(HFS)
      AUTOMOVE
      MODE(RDWR)

MOUNT FILESYSTEM('WTSCPLX5.&SYSNAME..SYSTEM.ROOT')
      MOUNTPOINT('/&SYSNAME.')
      UNMOUNT
      TYPE(HFS)  MODE(RDWR)

MOUNT FILESYSTEM('OMVS.&SYSNAME..ETC')
      MOUNTPOINT('/&SYSNAME./etc')
      UNMOUNT
      TYPE(HFS)  MODE(RDWR)

MOUNT FILESYSTEM('OMVS.&SYSNAME..VAR')
      MOUNTPOINT('/&SYSNAME./var')
      UNMOUNT
      TYPE(HFS)  MODE(RDWR)

MOUNT FILESYSTEM('/&SYSNAME./TMP')
      TYPE(TFS) MODE(RDWR)
      MOUNTPOINT('/&SYSNAME./tmp')
      PARM('-s 500')
      UNMOUNT
```

Figure 3-14 Active BPXPRM7A member

The OMVS processes may also be displayed within the z/OS UNIX environment and similar comparisons can be made. The way to look at UNIX processes is to use the shell environment and to execute the UNIX command **ps -ef**. This displays all processes and their environments in forest or family tree format.

The *z/OS V1R7.0 UNIX System Services Planning*, GA22-7800-08, and the *z/OS V1R7.0 UNIX System Services User's Guide*, SA22-7802-07, provide details of UNIX commands in the z/OS UNIX environment.

Example 3-12 UNIX System Services processes display from the shell

```
1 @ SC30:/u/cs01>ps -ef
     UID        PID       PPID  C    STIME TTY       TIME CMD
 BPXROOT          1          0  -   Oct 11 ?         0:06 BPXPINPR
 BPXROOT   16842754          1  -   Oct 11 ?         0:30 BPXVCMT
 BPXROOT   16842755          1  -   Oct 11 ?         0:30 BPXVCLNY
 BPXROOT   83951620          1  -   Oct 13 ?         1:39 EZASASUB
 BPXROOT   67174405          1  -   Oct 13 ?         1:39 EZBTCPIP
 BPXROOT      65543          1  - 22:35:02 ?         0:01 OMPROUTE
 BPXROOT   33619976          1  -   Oct 13 ?         1:39 EZBTTSSL
 BPXROOT   33619977          1  -   Oct 13 ?         1:39 EZBTMCTL
 BPXROOT      65546          1  -   Oct 11 ?         1:51 DSIATTMT
 BPXROOT      65547          1  -   Oct 11 ?         1:51 DSIATTMT
 BPXROOT      65548          1  -   Oct 11 ?         0:30 GFSCMAIN
 BPXROOT      65550          1  -   Oct 11 ?         0:00 /usr/sbin/inetd /etc/i
```

```
netd.conf
 BPXROOT   33619983          1  -   Oct 13 ?         1:39 EZBTTMST
 BPXROOT      65552          1  -   Oct 11 ?         0:20 DFHSIP
 BPXROOT   33619985          1  - 16:11:41 ?         0:10 EZBTCPIP
 BPXROOT   33619987          1  - 16:11:44 ?         0:10 EZBTTSSL
```

Notice that in Example 3-12 on page 86, the UNIX System Services after this initialization is running with user ID BPXROOT. The reason for this is that RACF cannot map a UNIX System Services UID to an MVS user ID correctly if there are multiple MVS user IDs defined with the same UID. So RACF uses the last referenced MVS user ID.

Some of the typical UNIX commands are:

- **mkdir/u/cso1** creates the directory for the user mount point. The permission bits would be set as specified in the etc/profile or $home/.profile.
- **ls -all** lists the files with their permission bits. From time to time you may need to change the permission bits in the file.
- The **chmod** command is used to change the permission bits associated with files.
- The TSO/E interface may be utilized to work with zOS UNIX files. You may browse files using the ISHELL PDSE interface or you may execute the **obrowse** command from the OMVS shell environment. You may also edit files using the ISHELL tools or you can use the **oedit** command from the OMVS shell.

Note: Both **obrowse** and **oedit** are TSO commands. If you used telnet or rlogin to get to the UNIX System Services shell, you have to use the **cat** command and the vi editor.

The ISHELL provides an ISPF look and feel, the OMVS shell a more UNIX or DOS look and feel, and of course for real UNIX users there is the vi editor.

Starting z/OS TCP/IP after IPL

Example 3-13 CS for z/OS IP startup

```
S TCPIPE
$HASP100 TCPIPE   ON STCINRDR
IEF695I START TCPIPE   WITH JOBNAME TCPIPE   IS ASSIGNED TO USER
TCPIP   , GROUP TCPGRP
$HASP373 TCPIPE   STARTED
IEE252I MEMBER CTIEZB00 FOUND IN SYS1.IBM.PARMLIB     1
IEE252I MEMBER CTIIDS00 FOUND IN SYS1.IBM.PARMLIB
EZZ7450I FFST SUBSYSTEM IS NOT INSTALLED
EZZ0300I OPENED INCLUDE FILE 'TCPIPE.TCPPARMS(HOME30)'
EZZ0300I OPENED INCLUDE FILE 'TCPIPE.TCPPARMS(STATIC30)'
EZZ0300I OPENED INCLUDE FILE 'TCPIPE.TCPPARMS(TELN30)'
EZZ0300I OPENED PROFILE FILE DD:PROFILE
EZZ0309I PROFILE PROCESSING BEGINNING FOR DD:PROFILE
EZZ0309I PROFILE PROCESSING BEGINNING FOR TCPIPE.TCPPARMS(HOME30)
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'TCPIPE.TCPPARMS(HOME30)'
EZZ0304I RESUMING PROCESSING OF FILE DD:PROFILE
EZZ0309I PROFILE PROCESSING BEGINNING FOR TCPIPE.TCPPARMS(STATIC30)
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'TCPIPE.TCPPARMS(STATIC30)'
EZZ0304I RESUMING PROCESSING OF FILE DD:PROFILE
EZZ0309I PROFILE PROCESSING BEGINNING FOR TCPIPE.TCPPARMS(TELN30)
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'TCPIPE.TCPPARMS(TELN30)'
EZZ0304I RESUMING PROCESSING OF FILE DD:PROFILE
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE DD:PROFILE   2
EZZ0641I IP FORWARDING NOFWDMULTIPATH SUPPORT IS ENABLED
```

```
EZZ0350I SYSPLEX ROUTING SUPPORT IS ENABLED   3
EZZ0351I SOURCEVIPA SUPPORT IS ENABLED         4
EZZ0624I DYNAMIC XCF DEFINITIONS ARE ENABLED   6
EZZ0338I TCP PORTS 1 THRU 1023 ARE RESERVED
EZZ0338I UDP PORTS 1 THRU 1023 ARE RESERVED
EZZ0613I TCPIPSTATISTICS IS ENABLED        7
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE IUTIQDF5
BPXF206I ROUTING INFORMATION FOR TRANSPORT DRIVER TCPIPE HAS BEEN INITIALIZED OR
UPDATED.
EZZ4202I Z/OS UNIX - TCP/IP CONNECTION ESTABLISHED FOR TCPIPE     8
IEF196I IEF237I 20C6 ALLOCATED TO TP20C6
IEF196I IEF237I EA06 ALLOCATED TO TPEA06
EZD1176I TCPIPE HAS SUCCESSFULLY JOINED THE TCP/IP SYSPLEX GROUP
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE IUTIQDF4
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE OSA20E0
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE OSA2080
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE IUTIQDF6
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE OSA20A0
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE OSA20C0
EZB6473I TCP/IP STACK FUNCTIONS INITIALIZATION COMPLETE.       9
EZAIN11I ALL TCPIP SERVICES FOR PROC TCPIPE ARE AVAILABLE.   9
EZZ4324I CONNECTION TO 10.10.20.130 ACTIVE FOR DEVICE IUTSAMEH   10
EZZ4324I CONNECTION TO 10.10.20.130 ACTIVE FOR DEVICE IUTSAMEH   10
EZZ4324I CONNECTION TO 10.10.20.130 ACTIVE FOR DEVICE IUTSAMEH   10
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE IUTIQDIO                  11
EZZ0400I TELNET/VTAM (SECOND PASS) BEGINNING FOR FILE: DD:PROFILE
EZZ0309I PROFILE PROCESSING BEGINNING FOR TCPIPE.TCPPARMS(HOME30)
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'TCPIPE.TCPPARMS(HOME30)'
EZZ0304I RESUMING PROCESSING OF FILE DD:PROFILE
EZZ0309I PROFILE PROCESSING BEGINNING FOR TCPIPE.TCPPARMS(STATIC30)
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'TCPIPE.TCPPARMS(STATIC30)'
EZZ0304I RESUMING PROCESSING OF FILE DD:PROFILE
EZZ0309I PROFILE PROCESSING BEGINNING FOR TCPIPE.TCPPARMS(TELN30)
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'TCPIPE.TCPPARMS(TELN30)'
EZZ0304I RESUMING PROCESSING OF FILE DD:PROFILE
EZZ6003I TELNET LISTENING ON PORT    23
EZZ0403I TELNET/VTAM (SECOND PASS) COMPLETE FOR FILE: DD:PROFILE      12
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE IUTSAMEH
EZZ4324I CONNECTION TO 10.20.10.100 ACTIVE FOR DEVICE IUTSAMEH          13
EZZ4324I CONNECTION TO 10.20.40.100 ACTIVE FOR DEVICE IUTSAMEH          13
EZZ4324I CONNECTION TO 10.30.20.100 ACTIVE FOR DEVICE IUTSAMEH          13
S FTPDE30
$HASP100 FTPDE30  ON STCINRDR
IEF695I START FTPDE30  WITH JOBNAME FTPDE30  IS ASSIGNED TO USER
TCPIP   , GROUP TCPGRP
$HASP373 FTPDE30  STARTED
$HASP395 FTPDE30  ENDED
+EZY2702I Server-FTP: Initialization completed at 15:04:00 on
10/18/05.
```

1 shows how the member that defines CTRACE processing has been found: CTIEZB00. This is discussed in Appendix A, “Component trace (CTRACE)” on page 209.

2 shows how the PROFILE.TCPIP for the stack has been found and processed.

3 Sysplexrouting is enabled so communication between z/OS TCP/IP is possible.

4 indicates that we will use the VIPA address for our outbound datagram source IP address.

6 Dynamic XCFs are enabled (DYNAMICXCF parameter).

7 TCPIPSTATICTICS will be generated,

8 shows how the stack has been bound to UNIX System Services. It indicates that the Common INET pre-router has successfully obtained a copy of the IP layer routing table from the stack.

9 The stack (TCP/IP) is successfully initialized and READY FOR WORK.

10 The MPCPTP connection to these IP addresses for IUTSAMEHs are active and REDAY FOR WORK. This facilitates inter-stack communications (between TCP/IP and VTAM for EE) as well as between z/OS TCP/IP stacks using XCF devices. This is triggered by the DYNAMICXCF parameter. IUTSAMEH connections do not need any I/O devices. For an IUTSAMEH connection, the device name is the reserved name IUTSAMEH. VTAM automatically activated the IUTSAMEH TRLE.

11 indicates that HiperSockets connectivity and routing are supported by this stack. HiperSockets are enabled with the DYNAMICXCF parameter and may be used for communication between stacks rather than using the XCF connections. For further discussions on this refer to Chapter 4, “Connectivity” on page 101.

12 The internal Telnet server (TN3270E) is successfully initialized and READY for connection requests.

Important: Since TCP/IP shares its Data Link Controls (DLCs) with VTAM, you must restart TCP/IP if you restart VTAM.

Verifying TCP/IP configuration

After the configuration files are updated we verified the configuration and we restarted the TCP/IP address space, ensuring that we saw the following message:

```
EZB6473I TCP/IP STACK FUNCTIONS INITIALIZATION COMPLETE
```

If the message is not displayed, the messages issued by the TCP/IP address space should describe why TCP/IP did not start.

Displaying the TCP/IP configuration

To display the enabled features and operating characteristics of a TCP/IP stack, enter any of the following commands:

- TSO/E command NETSTAT CONFIG
- MVS command D TCPIP,procname,NETSTAT,CONFIG
- UNIX shell command onetstat -f

Example 3-14 shows the output from the NETSTAT CONFIG display.

Example 3-14 NETSTAT CONFIG display

```
D TCPIP,TCPIPE,NETSTAT,CONFIG
EZD0101I NETSTAT CS V1R7 TCPIPE 400
TCP CONFIGURATION TABLE:         1
DEFAULTRCVBUFSIZE:  00016384  DEFAULTSNDBUFSIZE: 00016384
DEFLTMAXRCVBUFSIZE: 00262144
MAXRETRANSMITTIME:  120.000   MINRETRANSMITTIME: 0.500
ROUNDTRIPGAIN:      0.125     VARIANCEGAIN:      0.250
VARIANCEMULTIPLIER: 2.000     MAXSEGLIFETIME:    30.000
DEFAULTKEEPALIVE:   00000120  DELAYACK:          YES
RESTRICTLOWPORT:    YES       SENDGARBAGE:       NO
TCPTIMESTAMP:       YES       FINWAIT2TIME:      600
```

```
TTLS:               NO
UDP CONFIGURATION TABLE:    2
DEFAULTRCVBUFSIZE: 00065535  DEFAULTSNDBUFSIZE: 00065535
CHECKSUM:          YES
RESTRICTLOWPORT:   YES       UDPQUEUELIMIT:     YES
IP CONFIGURATION TABLE:         3
FORWARDING: YES    TIMETOLIVE: 00064  RSMTIMEOUT:  00060
FIREWALL:   NO
IPSECURITY: NO
ARPTIMEOUT: 01200  MAXRSMSIZE: 65535  FORMAT:      LONG
IGREDIRECT: NO     SYSPLXROUT: YES    DOUBLENOP:   NO
STOPCLAWER: NO     SOURCEVIPA: YES
MULTIPATH:  NO     PATHMTUDSC: NO     DEVRTRYDUR:  0000000090
DYNAMICXCF: YES
  IPADDR: 10.10.20.130     SUBNET: 255.255.255.0    METRIC: 08
  SECCLASS: 255
IQDIOROUTE: NO
TCPSTACKSRCVIPA: NO
IPV6 CONFIGURATION TABLE:
FORWARDING:    YES  HOPLIMIT:   00255  IGREDIRECT:  NO
SOURCEVIPA:    NO   MULTIPATH:  NO     ICMPERRLIM:  00003
IGRTRHOPLIMIT: NO
DYNAMICXCF: NO
TCPSTACKSRCVIPA: NO
SMF PARAMETERS:       4
TYPE 118:
  TCPINIT:      00   TCPTERM:    00   FTPCLIENT:    03
  TN3270CLIENT: 04   TCPIPSTATS: 00
TYPE 119:
  TCPINIT:      NO   TCPTERM:    NO   FTPCLIENT:    YES
  TCPIPSTATS:   NO   IFSTATS:    NO   PORTSTATS:    NO
  STACK:        NO   UDPTERM:    NO   TN3270CLIENT: YES
GLOBAL CONFIGURATION INFORMATION:     5
TCPIPSTATS: YES  ECSALIMIT: 0000000K  POOLLIMIT: 0000000K
MLSCHKTERM: NO
SYSPLEX MONITOR:
  TIMERSECS: 0060  RECOVERY: NO   DELAYJOIN: NO   AUTOREJOIN: NO
NETWORK MONITOR CONFIGURATION INFORMATION: 6
PKTTRCSRV: NO   TCPCNNSRV: NO   SMFSRV: NO
END OF THE REPORT
```

Parameters such as SOURCEVIPA can be either ENABLED or DISABLED. A value of 01 in the NETSTAT CONFIG display means it is ENABLED.

1 shows what settings are in effect in the TCPCONFIG parameters.

2 shows what is set for the UDPCONFIG parameters.

3 shows the settings in effect in the IPCONFIG parameters.

4 shows what is in effect for SMFCONFIG.

5 shows the settings in effect for GLOBALCONFIG.

6 shows the setting in effect for Network Monitoring Information.

Display status of devices

Display the devices using the MVS display command, TSO NETSTAT, or UNIX onetstat -d (see Example 3-15 on page 91).

Example 3-15 Results of device display

```
MVS TCP/IP NETSTAT CS V1R7       TCPIP Name: TCPIPA          00:48:39
DevName: LOOPBACK          DevType: LOOPBACK
  DevStatus: Ready
  LnkName: LOOPBACK          LnkType: LOOPBACK   LnkStatus: Ready 1
NetNum: n/a  QueSize: n/a
    ActMtu: 65535
  BSD Routing Parameters:
    MTU Size: n/a               Metric: 00
    DestAddr: 0.0.0.0           SubnetMask: 0.0.0.0
  Multicast Specific:
    Multicast Capability: No
  Link Statistics:
    BytesIn                          = 602814
    Inbound Packets                  = 9190
    Inbound Packets In Error         = 0
    Inbound Packets Discarded        = 0
    Inbound Packets With No Protocol = 0
    BytesOut                         = 602814
    Outbound Packets                 = 9190
    Outbound Packets In Error        = 0
    Outbound Packets Discarded       = 0
IntfName: LOOPBACK6        IntfType: LOOPBACK6 IntfStatus: Ready 2
NetNum: n/a  QueSize: n/a
    ActMtu: 65535
  Multicast Specific:
    Multicast Capability: No
  Interface Statistics:
    BytesIn                          = 0
    Inbound Packets                  = 0
    Inbound Packets In Error         = 0
    Inbound Packets Discarded        = 0
    Inbound Packets With No Protocol = 0
    BytesOut                         = 0
    Outbound Packets                 = 0
    Outbound Packets In Error        = 0
    Outbound Packets Discarded       = 0
DevName: OSA2080           DevType: MPCIPA
  DevStatus: Ready 3
LnkName: OSA2080LNK        LnkType: IPAQENET   LnkStatus: Ready 3
NetNum: n/a  QueSize: n/a  Speed: 0000000100
    IpBroadcastCapability: No
    CfgRouter: Non                   ActRouter: Non
  ArpOffload: Yes  4         ArpOffloadInfo: Yes  4
    ActMtu: 1492
  VLANid: 10  5              VLANpriority: Disabled
    DynVLANRegCfg: No                DynVLANRegCap: Yes
    ReadStorage: GLOBAL (4096K)      InbPerf: Balanced
  ChecksumOffload: Yes  4    SegmentationOffload: Yes  4
    SecClass: 255
  BSD Routing Parameters:
    MTU Size: 1500              Metric: 10
    DestAddr: 0.0.0.0           SubnetMask: 255.255.255.0
  Multicast Specific:
    Multicast Capability: Yes
    Group             RefCnt
    -----             ------
    224.0.0.5         0000000001
    224.0.0.1         0000000001
  Link Statistics:
```

```
BytesIn                          = 7312941
Inbound Packets                  = 57814
Inbound Packets In Error         = 24429
Inbound Packets Discarded        = 0
Inbound Packets With No Protocol = 0
BytesOut                         = 1050260
Outbound Packets                 = 9237
Outbound Packets In Error        = 6
Outbound Packets Discarded       = 0
```

1 indicates that IPv4 is supported and the loopback device is READY.

2 indicates that IPv6 is supported and the loopback device is READY.

3 indicates the overall status of the OSA device OSA2080: READY. If this status is not READ verify that the VTAM Major node is active. This can be done with the VTAM command.

4 The OFFLOAD feature is enabled.

5 The VLAN ID defined on the LINK statement in the PROFILE data set.

Example 3-16 Results of the TRLE display

```
D NET,TRL,TRLE=OSA2080P
IST097I DISPLAY ACCEPTED
IST075I NAME = OSA2080P, TYPE = TRLE 384
IST1954I TRL MAJOR NODE = OSA2080
IST486I STATUS= ACTIV, DESIRED STATE= ACTIV   A
IST087I TYPE = LEASED            , CONTROL = MPC , HPDT = YES
IST1715I MPCLEVEL = QDIO       MPCUSAGE = SHARE
IST1716I PORTNAME = OSA2080    LINKNUM =   0   OSA CODE LEVEL = 0804
IST1577I HEADER SIZE = 4096 DATA SIZE = 0 STORAGE = ***NA***
IST1221I WRITE DEV = 2081 STATUS = ACTIVE     STATE = ONLINE     B
IST1577I HEADER SIZE = 4092 DATA SIZE = 0 STORAGE = ***NA***
IST1221I READ  DEV = 2080 STATUS = ACTIVE     STATE = ONLINE      B
IST1221I DATA  DEV = 2082 STATUS = ACTIVE     STATE = N/A           C
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1717I ULPID = TCPIPE
```

A indicates that the Major node is ACTIVE and ONLINE.

B The READ and WRITE channels are ACTIVE and ONLINE.

C The data channel is also ACTIVE.

Display storage usage

The z/OS Communications Server uses the Communications Storage Manager (CSM) to manage storage pools. The recommendation is to increase storage allocations by a minimum of 20 MB for TCP/IP in the CSA definition in IEASYSxx and the FIXED and ECSA definitions in IVTPRMxx.

Check your storage utilization to ensure that you made the correct allocations. Storage usage may also be controlled using the GLOBALCONFIG ECSALIMIT and GLOBALCONFIG POOLLIMIT parameters. ECSALIMIT allows you to specify the maximum amount of extended common service area (ECSA) that TCP/IP can use, and POOLLIMIT allows you to specify the maximum amount of authorized private storage that TCP/IP can use within the TCP/IP address space. You can also use the MVS command **`D TCPIP,tcpproc,STOR`** to display TCP/IP storage usage (see Example 3-17 on page 93).

Example 3-17 Results of storage display

```
D TCPIP,TCPIPE,STOR
EZZ8454I TCPIPE   STORAGE    CURRENT   MAXIMUM     LIMIT 757
EZZ8455I TCPIPE   ECSA         6914K     8122K   NOLIMIT
EZZ8455I TCPIPE   POOL         4232K     4348K   NOLIMIT
EZZ8459I DISPLAY TCPIP STOR COMPLETED SUCCESSFULLY
```

Verifying TCPIP.DATA statement values in z/OS

To display which TCPIP.DATA statement values are being used and where they are being obtained from, use trace resolver output. You can obtain trace resolver output at your TSO screen by issuing the following TSO commands:

```
alloc f(systcpt) dsn(*)
READY
netstat up
READY
free f(systcpt)
READY
```

Tip: When directing trace resolver output to a TSO terminal, define the screen size to be only 80 columns wide. Otherwise, trace output is difficult to read.

Verifying TCPIP.DATA statement values in USS

To display which TCPIP.DATA statement values are being used and where they are being obtained from, use trace resolver output. You can obtain trace resolver output by issuing the following z/OS UNIX shell commands:

```
#
export RESOLVER_TRACE=stdout
#
onetstat –u
#
set -A RESOLVER_TRACE
```

Verifying PROFILE.TCPIP

Many configuration values specified within the PROFILE.TCPIP file can be verified with the TSO NETSTAT or z/OS UNIX **onetstat** commands. To verify the physical network and hardware definitions, use the MVS D TCPIP,,N,DEV, NETSTAT DEVLINKS or **onetstat -d** commands. To see operating characteristics, use z/OS displays, NETSTAT CONFIG, or onetstat -f.

Verifying interfaces with PING and TRACERTE

PING and TRACERTE can be used in the TSO environment to verify adapters or interfaces attached to the z/OS host. In the z/OS UNIX environment, **oping** and **otracert** can be used with identical results. For information about the syntax and output of the commands, refer to *z/OS Communications Server: IP System Administrator's Commands, Version 1 Release 7*, SC31-8781-05. Given that your PROFILE.TCPIP file contains the interfaces of your installation and that the TCPIP.DATA file contains the correct TCPIPJOBNAME, the TCP/IP address space is configured and you can go on to configuring routes, servers, and so on.

Verifying local name resolution with TESTSITE

Use the TESTSITE command to verify that the hlq.HOSTS.ADDRINFO and hlq.HOSTS.SITEINFO data sets can correctly resolve the name of a host, gateway, or net. For more information about the TESTSITE command, refer to *z/OS Communications Server: IP System Administrator's Commands, Version 1 Release 7*, SC31-8781-05.

Verifying PROFILE.TCPIP and TCPIP.DATA using HOMETEST

Use the HOMETEST command to verify the HOSTNAME, DOMAINORIGIN, SEARCH, and NSINTERADDR TCPIP.DATA statements. HOMETEST will use the resolver to obtain the IP addresses assigned to the HOSTNAME and compare them to the HOME list specified in PROFILE.TCPIP. A warning message will be issued if any HOSTNAME IP addresses are missing from the HOME list.

Activate TRACE RESOLVER if you would like detailed information about how the HOSTNAME is resolved to IP addresses. The information will also include what TCPIP.DATA data set names were used. This can be done by issuing the following TSO command before running HOMETEST. The detailed information will be displayed on your TSO screen.

```
allocate dd(systcpt) da(*)
Issue the following TSO command after HOMETEST to turn off TRACE RESOLVER output.
free dd(systcpt)
```

If you do not have TRACE RESOLVER turned on before running HOMETEST, the following is displayed (Figure 3-15).

```
hometest Running IBM MVS TCP/IP CS V1R6 TCP/IP Configuration Tester
FTP.DATA file not found. Using hardcoded default values.
TCP Host Name is: SC30
Using Name Server to Resolve SC30
The following IP addresses correspond to TCP Host Name: SC30
10.10.10.210
The following IP addresses are the HOME IP addresses defined in PROFILE.TCPIP:
10.10.10.210
127.0.0.1
All IP addresses for SC30 are in the HOME list! Hometest was successful - all Tests Passed!
```

Figure 3-15 HOMETEST

3.9 Re-configuring the system with z/OS commands

The z/OS Communications Server provides a way to change the running TCP/IP configuration dynamically: The VARY OBEYFILE command. This command replaces the **OBEYFILE** TSO command. The VARY command is an z/OS Console command. It allows you to add, delete, or completely redefine all devices dynamically as well as change TN3270 parameters, routing, and almost any TCP/IP parameter in the profile. These changes are in effect until the TCP/IP started task is started again or another VARY OBEYFILE command overrides them. Authorization is through the user's RACF profile containing the MVS.VARY.TCPIP.OBEYFILE definition. There is no OBEY statement in the PROFILE.TCPIP, which in earlier MVS TCP/IP implementations provided authorization.

For further details on the VARY OBEYFILE command, see *z/OS V1R7.0 CS: IP System Administrator's Commands*, SC31-8781. For more information about RACF definitions, see *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 2 - Standard Applications*, SG24-7170.

3.9.1 Deleting a device and adding/changing a device

You could use the OBEYFILE command to reconfigure the devices being used by the stack. Reconfiguration could be the deletion of existing devices, addition of new devices, or redefinition of an existing device. The syntax of the statements for OBEYFILE processing is the same as that being used in PROFILE.TCPIP.

Device reconfiguration is a three-step process:

1. Stop the device with an z/OS console command (VARY STOP) or with a VARY OBEYFILE that names a data set in which the STOP command is defined.
2. Activate an OBEYFILE that deletes the links and the devices.
3. Activate an OBEYFILE that adds the new or changed links and devices and then starts them.

If you wish to delete a device, the order of steps you take is important. The DELETE statement in PROFILE.TCPIP allows you to remove LINK, DEVICE and PORT or PORTRANGE definitions. The general sequence for deleting and adding back a device is:

1. Stop the device.
2. Remove the HOME address by excluding it from the full stack's HOME list.
3. Delete the link.
4. Delete the device.
5. Add the new or changed device.
6. Add the new or changed link.
7. Add the HOME statements for the full stack.
8. Add the full gateway statements for the stack if you are using static routing.
9. Start the device.

3.9.2 Modifying a device

In this example we wanted to change the address of OSA-Express device OSA2080 at address 2080 from 10.10.2.212 to 10.10.2.201. This process would involve deleting the current definition and redefining the device.

In Example 3-18 you see all the devices in the TCPIPE stack. Under each device a single link is defined, which is associated with an IP address.

Example 3-18 Displays of devices and home before deletion

```
D TCPIP,TCPIPE,N,HOME
 EZD0101I NETSTAT CS V1R7 TCPIPE 269
269 00000090  HOME ADDRESS LIST:
269 00000090  LINKNAME:   STAVIPA1LNK
269 00000090    ADDRESS:  10.10.10.210
269 00000090      FLAGS:  PRIMARY
269 00000090  LINKNAME:   OSA2080LNK
269 00000090    ADDRESS:  10.10.2.212
269 00000090      FLAGS:
269 00000090  LINKNAME:   OSA20A0LNK
269 00000090    ADDRESS:  10.10.3.213
269 00000090      FLAGS:
269 00000090  LINKNAME:   OSA20C0LNK
269 00000090    ADDRESS:  10.10.2.214
269 00000090      FLAGS:
269 00000090  LINKNAME:   OSA20E0LNK
269 00000090    ADDRESS:  10.10.3.215
269 00000090      FLAGS:
269 00000090  LINKNAME:   EZASAMEMVS
269 00000090    ADDRESS:  10.10.20.130
269 00000090      FLAGS:
269 00000090  LINKNAME:   IQDIOLNK0A0A1482
269 00000090    ADDRESS:  10.10.20.130
269 00000090      FLAGS:
269 00000090  LINKNAME:   LOOPBACK
269 00000090    ADDRESS:  127.0.0.1
```

```
269 00000090      FLAGS:
269 00000090  INTFNAME:   LOOPBACK6
269 00000090    ADDRESS:  ::1
269 00000090      TYPE:   LOOPBACK
269 00000090      FLAGS:
269 00000090  12 OF 12 RECORDS DISPLAYED
269 00000090  END OF THE REPORT
```

Notice the address of OSA2080LNK (10.10.2.212). We need to change this in the running system by stopping, deleting, redefining, and adding back the OSA-Express device and link and home address.

Because the STOP command is executed as the last statement within an OBEYFILE regardless of its position within the file, you cannot do the STOP and DELETE in one step. Trying to do so will result in the error messages illustrated in Example 3-19.

Example 3-19 Timing problems with device/link deletion

```
V TCPIP,TCPIPE,O,TCPIPE.TCPPARMS(DELE30)
 EZZ0060I PROCESSING COMMAND: VARY TCPIP,TCPIPE,O,TCPIPE.TCPPARMS(DELE30)
 EZZ0300I OPENED OBEYFILE FILE 'TCPIPE.TCPPARMS(DELE30)'
 EZZ0309I PROFILE PROCESSING BEGINNING FOR 'TCPIPE.TCPPARMS(DELE30)'
 EZZ0395I DELETE LINK OSA2080LNK ON LINE 20 FAILED BECAUSE LINK IS ACTIVE
 EZZ0395I DELETE DEVICE OSA2080 ON LINE 21 FAILED BECAUSE DEVICE IS AC
 TIVE
 EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'TCPIPE.TCPPARMS(DELE30)'
 EZZ0303I OBEYFILE FILE CONTAINS ERRORS
 EZZ0331I NO HOME ADDRESS ASSIGNED TO LINK OSA2080LNK
 EZZ0059I VARY OBEY COMMAND FAILED: SEE PREVIOUS MESSAGES
 BPXF206I ROUTING INFORMATION FOR TRANSPORT DRIVER TCPIPE HAS BEEN
 INITIALIZED OR UPDATED.
 EZZ4315I DEACTIVATION COMPLETE FOR DEVICE OSA2080
```

We also want to delete the LINK before the DEVICE (step 3 and step 4) in the right sequence. When we add them later, we reverse the order again as in the PFROFILE data set. The sequence of device definitions is always DEVICE, then LINK, so the statements in the profile need to be reversed. If they are not, the following error occurs (Figure 3-20).

Example 3-20 Another timing problem with device/link deletion

```
V TCPIP,TCPIPE,O,TCPIPE.TCPPARMS(DELE30)
EZZ0060I PROCESSING COMMAND: VARY TCPIP,TCPIPE,O,TCPIPE.TCPPARMS(DELE30)
EZZ0300I OPENED OBEYFILE FILE 'TCPIPE.TCPPARMS(DELE30)'
EZZ0309I PROFILE PROCESSING BEGINNING FOR 'TCPIPE.TCPPARMS(DELE30)'
EZZ0395I DELETE DEVICE OSA2080 ON LINE 15 FAILED BECAUSE DEVICE HAS  A LINK DEFINED
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'TCPIPE.TCPPARMS(DELE30)'
EZZ0303I OBEYFILE FILE CONTAINS ERRORS
EZZ0059I VARY OBEY COMMAND FAILED: SEE PREVIOUS MESSAGES
```

We will now change the characteristic of the device and link of OSA2080LNK. In the following steps we stop the device (see Example 3-21).

Example 3-21 Command to stop the device

```
V TCPIP,TCPIPE,STOP,OSA2080
  EZZ0060I PROCESSING COMMAND: VARY TCPIP,TCPIPE,STOP,OSA2080
  EZZ0053I COMMAND VARY STOP COMPLETED SUCCESSFULLY
  EZZ4315I DEACTIVATION COMPLETE FOR DEVICE OSA2080
```

Then delete it from the stack (see Example 3-22).

Example 3-22 Command to delete the device

```
V TCPIP,TCPIPE,O,TCPIPE.TCPPARMS(DELE30)
EZZ0060I PROCESSING COMMAND: VARY TCPIP,TCPIPE,O,TCPIPE.TCPPARMS(DELE30)
EZZ0300I OPENED OBEYFILE FILE 'TCPIPE.TCPPARMS(DELE30)'
EZZ0309I PROFILE PROCESSING BEGINNING FOR 'TCPIPE.TCPPARMS(DELE30)'
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'TCPIPE.TCPPARMS(DELE30)'
EZZ0053I COMMAND VARY OBEY COMPLETED SUCCESSFULLY
```

The results of the delete are shown in the display NETSTAT for the HOME addresses (see Example 3-23). Notice the missing OSA2080LNK link as compared with Example 3-18 on page 95.

Example 3-23 Omission of OSA2080LNK

```
D TCPIP,TCPIPE,N,HOME
EZD0101I NETSTAT CS V1R7 TCPIPE 533
533 00000090  HOME ADDRESS LIST:
533 00000090  LINKNAME:   STAVIPA1LNK
533 00000090    ADDRESS:  10.10.10.210
533 00000090      FLAGS:
533 00000090  LINKNAME:   OSA20A0LNK
533 00000090    ADDRESS:  10.10.3.213
533 00000090      FLAGS:
533 00000090  LINKNAME:   OSA20C0LNK
533 00000090    ADDRESS:  10.10.2.214
533 00000090      FLAGS:
533 00000090  LINKNAME:   OSA20E0LNK
533 00000090    ADDRESS:  10.10.3.215
533 00000090      FLAGS:
533 00000090  LINKNAME:   EZASAMEMVS
533 00000090    ADDRESS:  10.10.20.130
533 00000090      FLAGS:
533 00000090  LINKNAME:   IQDIOLNK0A0A1482
533 00000090    ADDRESS:  10.10.20.130
533 00000090      FLAGS:
533 00000090  LINKNAME:   LOOPBACK
533 00000090    ADDRESS:  127.0.0.1
533 00000090      FLAGS:
533 00000090  INTFNAME:   LOOPBACK6
533 00000090    ADDRESS:  ::1
533 00000090      TYPE:   LOOPBACK
533 00000090      FLAGS:
533 00000090  11 OF 11 RECORDS DISPLAYED
533 00000090  END OF THE REPORT
```

In Example 3-24, you see the statements necessary to delete the IP address, the LINK, and the DEVICE.

Example 3-24 OBEYFILE member to delete the device OSA2080

```
HOME
   10.10.4.214   IUTIQDF4LNK
   10.10.4.215   IUTIQDF5LNK
   10.10.5.216   IUTIQDF6LNK
   10.10.10.210   STAVIPA1LNK
;;;10.10.2.212    OSA2080LNK
   10.10.3.213    OSA20A0LNK
```

```
   10.10.2.214    OSA20C0LNK
   10.10.3.215    OSA20E0LNK
;
 DELETE  LINK  OSA2080LNK
 DELETE DEVICE OSA2080
```

We next add the device and link back with the changed address definition **3**, as Example 3-25 illustrates.

Example 3-25 OBEYFILE member to add the device (ADDE30)

```
DEVICE OSA2080  MPCIPA
 LINK   OSA2080LNK  IPAQENET      OSA2080 VLANID 10
;
HOME
   10.10.4.214   IUTIQDF4LNK
   10.10.4.215   IUTIQDF5LNK
   10.10.5.216   IUTIQDF6LNK
   10.10.10.210   STAVIPA1LNK
   10.10.2.201    OSA2080LNK 3
   10.10.3.213    OSA20A0LNK
   10.10.2.214    OSA20C0LNK
   10.10.3.215    OSA20E0LNK
```

This is followed by a display to verify the addition to the stack, as shown in Example 3-26.

Example 3-26 Display with OSA2080 using a new address

```
D TCPIP,TCPIPE,N,HOME
EZD0101I NETSTAT CS V1R7 TCPIPE 164
HOME ADDRESS LIST:
LINKNAME:   IUTIQDF4LNK
  ADDRESS:  10.10.4.214
    FLAGS:  PRIMARY
LINKNAME:   IUTIQDF5LNK
  ADDRESS:  10.10.4.215
    FLAGS:
LINKNAME:   IUTIQDF6LNK
  ADDRESS:  10.10.5.216
    FLAGS:
LINKNAME:   STAVIPA1LNK
  ADDRESS:  10.10.10.210
    FLAGS:
LINKNAME:   OSA2080LNK
  ADDRESS:  10.10.2.201
    FLAGS:
LINKNAME:   OSA20A0LNK
  ADDRESS:  10.10.3.213
    FLAGS:
LINKNAME:   OSA20C0LNK
  ADDRESS:  10.10.2.214
    FLAGS:
LINKNAME:   OSA20E0LNK
  ADDRESS:  10.10.3.215
    FLAGS:
LINKNAME:   EZASAMEMVS
  ADDRESS:  10.10.20.130
    FLAGS:
LINKNAME:   IQDIOLNK0A0A1482
  ADDRESS:  10.10.20.130
```

```
    FLAGS:
LINKNAME:   LOOPBACK
  ADDRESS:  127.0.0.1
    FLAGS:
INTFNAME:   LOOPBACK6
  ADDRESS:  ::1
    TYPE:   LOOPBACK
    FLAGS:
12 OF 12 RECORDS DISPLAYED
```

Since we are using a flat network (no dynamic routing) we need to add BEGINRoutes statements to the stack. The member with the definition change is shown in Example 3-27.

Example 3-27 OBEYFILE member to add the gateway statements

```
BEGINRoutes
; Direct Routes - Routes that are directly connected to my interfaces
;     Destination     Subnet Mask   First Hop Link Name     Packet Size ;

 ROUTE 10.10.3.0       255.255.255.0 =        OSA20A0LNK    mtu defaultsize
 ROUTE 10.10.2.0       255.255.255.0 =        OSA20c0LNK    mtu defaultsize
 ROUTE 10.10.3.0       255.255.255.0 =        OSA20e0LNK    mtu defaultsize
 ROUTE 10.10.4.0       255.255.255.0 =        IUTIQDF4LNK   mtu defaultsize
 ROUTE 10.10.4.0/24                  =        IUTIQDF5LNK   mtu defaultsize
 ROUTE 10.10.5.0       255.255.255.0 =        IUTIQDF6LNK   mtu defaultsize
; Default Route - All packets to an unknown destination are routed
;through this route.
;     Destination     First Hop    Link Name   Packet Size

 ROUTE DEFAULT         10.10.3.1    OSA20A0LNK  mtu defaultsize
 ROUTE DEFAULT         10.10.2.1    OSA20C0LNK  mtu defaultsize
 ROUTE DEFAULT         10.10.3.1    OSA20E0LNK  mtu defaultsize
ENDRoutes
```

3.10 Job log versus syslog as diagnosis tool

In the past, we often used the TCP/IP job log to detect problems. Most procedures now send messages to the syslogd daemon or the MVS console log. Refer to the *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 2 - Standard Applications*, SG24-7170, for more information about the syslog daemon. Individual server documentation also provides information about diagnosis.

3.11 Message types: Where to find them

For an explanation of z/OS UNIX and TCP/IP messages or SNA sense codes, refer to the following publications:

- Messages with prefix of BPX

  You will find the explanations for these messages in *z/OS V1R7.0 MVS System Messages, Vol 3 (ASB-BPX)*, SA22-7633.

- Messages with prefix of EZA

  For z/OS Communications Server IP, you will find the explanations for these messages in *z/OS V1R7.0 CS: IP Messages Volume 1 (EZA)*, SC31-8783.

- Messages with prefix of EZB

  For z/OS Communications Server IP, you will find the explanations for these messages in *z/OS V1R7.0 CS: IP Messages Volume 2 (EZB)*, SC31-8784.

- Messages with prefix of EZY

  For z/OS Communications Server IP, you will find the explanations for these messages in *z/OS V1R7.0 CS: IP Messages Volume 3 (EZY)*, SC31-8785.

- Messages with prefix of EZZ and SNM™

  For z/OS Communications Server IP, you will find the explanations for these messages in *z/OS V1R7.0 CS: IP Messages Volume 4 (EZZ-SNM)*, SC31-8786.

- Messages with prefix of FOMC, FOMM, FOMO, FSUC, and FSUM

  You will find the explanations for these messages in *z/OS V1R7.0 UNIX System Services Messages and Codes*, SA22-7807.

- Eight-digit SNA sense codes and DLC codes

  You will find the explanations for these codes in *z/OS V1R7.0 CS: IP and SNA Codes*, SC31-8791.

- UNIX System Services return codes and reason codes

  You will find the explanations for these codes in *z/OS V1R7.0 UNIX System Services Messages and Codes*, SA22-7807.

3.12 Health Checker

IBM Health Checker for z/OS is now a z/OS base component that installations can use to gather information about their system environment and system parameters to help identify potential configuration problems before they impact availability or cause outages. Individual products, z/OS components, or ISV software can provide checks that take advantage of the IBM Health Checker for the z/OS framework. For additional information about checks and about IBM Health Checker for z/OS, see *IBM Health Checker for z/OS: User's Guide*. z/OS users can obtain the IBM Health Checker for z/OS from the z/OS Downloads page at:

http://www.ibm.com/servers/eserver/zseries/zos/downloads/

4

Connectivity

In today's networked world, the usability of a computer system is defined by its connectivity. While there are many ways for TCP/IP traffic to reach IBM's mainframes, this chapter discusses the most commonly used, and most important types of mainframe connectivity.

Detailed topics regarding these interfaces are provided, including useful implementation information, design scenarios, and set up examples.

This chapter discusses the following.

Section	Topic
4.1, “What we mean by connectivity” on page 102	Discusses the network connectivity options supported by z/OS Communications Server TCP/IP. Key characteristics of VLAN implementation are also described.
4.2, “Important and commonly used interfaces” on page 105	Discusses the commonly used interfaces supported by the IBM System z9 and zSeries servers that deliver the best throughput and performance, offer the most flexibility, and provide highest levels of availability.
4.3, “The common design scenarios for connectivity” on page 113	Presents commonly implemented connectivity design scenarios, their dependencies, advantages, considerations, and our recommendations.
4.4, “How connectivity is implemented” on page 121	Presents selected implementation scenarios, tasks, configuration examples, and problem determination suggestions.

4.1 What we mean by connectivity

Connectivity is the pipeline through which data is exchanged between clients and servers via physical and logical communication interfaces and the network. The IBM System z9 and zSeries servers provide a wide range of interface options for connecting your z/OS system to an IP network or to another IP host. Some interfaces offer point-to-point or point-to-multipoint connectivity, while others support Local Area Network (LAN) connectivity. Figure 4-1 depicts the physical interfaces (and device types) provided by System z9 and zSeries servers. The physical network interface is enabled through z/OS Communications Server (TCP/IP) definitions.

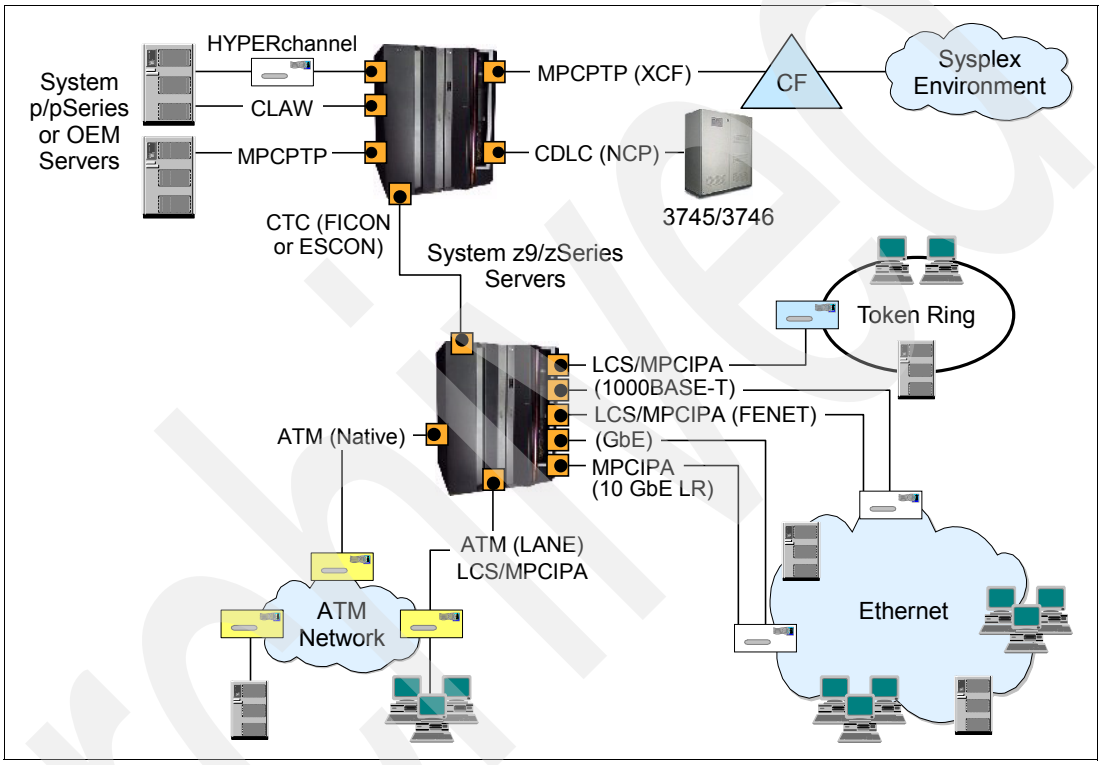


Figure 4-1 System z9 and zSeries - Physical interfaces

Connectivity within the System z9 or zSeries servers is provided by internal or logical interfaces such as SAMEHOST or HiperSockets.

4.1.1 Basic concepts

Network connectivity is handled by the physical and logical interfaces to enable transport of IP datagrams. Using the OSI model as an example, it is Layer 1 (physical layer) and Layer 2 (Data link control Layer). The z/OS Communications Server has implemented several types of interfaces to connect to different networking environments. These environments vary from point-to-point connections, like MPCPTP, CTC, and CLAW to LAN connections such as LCS and MPCIPA (as shown in Figure 4-1). The supported interfaces are as follows:

ATM Enables TCP/IP to send data to an asynchronous transfer mode (ATM) network using an OSA-Express ATM adapter over an ATM virtual circuit.

CDLC ESCON® attachments can be used to provide native IP transport, using the channel data link control (CDLC), between the 3746 IP and host systems running the z/OS Communications Server. The host

systems can be directly attached to the 3746-9x0 or connected via an ESCON director (ESCD).

CLAW Provides access from IBM System p or pSeries® directly to a TCP/IP stack or stacks over a channel. The CLAW (common link access to workstation) interface can also be used to provide connectivity to the original equipment manufacturer (OEM), such as the Cisco Channel Interface Processor (CIP).

CTC Provides access to TCP/IP hosts by way of a channel-to-channel (CTC) connection established over a FICON or ESCON channel.

HYPERchannel Provides access to TCP/IP hosts by way of HYPERchannel series A devices and series DX devices that function as series A devices.

LCS There are a variety of communications adapters that support a protocol called the LAN Channel Station (LCS). One example is the Open Systems Adapter-Express (OSA-Express).

MPCIPA Provides access to TCP/IP hosts through Multipath Channel IP Assist (MPCIPA), using OSA-Express in Queued Direct I/O (QDIO) mode and HiperSockets using the internal Queued Direct I/O (iQDIO).

MPCPTP Provides access to TCP/IP hosts through Multipath Channel Point-to-Point (MPCPTP) links. MPCPTP is used to directly connect to other hosts or z/OS LPARs, or by configuring it to utilize Coupling Facility links (if the z/OS LPAR is part of a sysplex).

SAMEHOST The SAMEHOST Data Link Control (DLC) enables communication between CS for z/OS IP and other servers running on the same MVS image.

The configuration and control of these interfaces is provided by VTAM. They are enabled in VTAM as TRLE minor nodes. For most of them, their definitions are created by VTAM dynamically and are transparent to the application.

Role of VTAM in the TCP/IP configuration

The z/OS Communications Server provides a set of High Performance Data Transfer (HPDT) services that includes MultiPath Channel (MPC), a high-speed channel interface designed for network protocol use (for example, APPN or TCP/IP).

Multiple protocols can either share or have exclusive use of a set of channel paths to an attached platform. MPC provides the user with the ability to have multiple device paths, defined as a single logical connection.

The term MPC group is used to define a single MPC connection that can contain multiple read and write paths. The number of read and write paths does not have to be equal, but there must be at least one read and write path defined within each MPC group.

MPC groups are defined using the Transport Resource List (TRL), where each defined MPC group becomes an entry (that is, a TRLE) in the TRL table.

The user defines the channel paths that are a part of the group in the TRLE. Each TRLE is identified by a resource_name. For OSA-Express, the TRLE also has a port_name to identify the association between VTAM and TCP/IP, allowing connectivity to the OSA-Express port. For details on defining a TRLE, refer to *z/OS Communications Server SNA Resource Definition Reference,* SC31-8778.

Positioning the interface options

To position the interface options, we have listed their characteristics based on interface type, attachment type, protocol type, and their importance in today's IP network (see Table 4-1). The importance of a given interface is determined by its acceptance in the industry, data throughput rate, as well as its Reliability, Availability, and Serviceability (RAS) capabilities.

Table 4-1 Supported interfaces

Interface type	Attachment type	Protocol type	Importance
ATM	ATM Native mode through OSA-Express	Asynchronous Transfer Mode (ATM) network	Low
CDLC	Network Control Program via 3745/3746	Point-to-point (PTP)	Low
CLAW	IBM System p or pSeries servers Channel attached routers	Point-to-point (PTP) Point-to-Multipoint (PTM)	Low
CTC	FICON or ESCON channel	Point-to-point (PTP)	Medium
HYPERchannel	Series A devices	Point-to-Multipoint (PTM)	Low
LCS	OSA-Express: 1000BASE-T Fast Ethernet Token Ring ATM Native and LAN Emulation	Local Area Network (LAN)	Medium
MPCIPA	HiperSockets[a] and OSA-Express: 10 Gigabit Ethernet Gigabit Ethernet 1000BASE-T Fast Ethernet Token Ring ATM LAN Emulation	Local Area Network (LAN)	High
MPCPTP (XCF)	IUTSAMEH XCF (CF link)	Point-to-point (PTP)	Low
SAMEHOST (Data Link Control)	SNALINK LU0 SNALINK LU6.2 X25NPSI	Point-to-point (PTP) Point-to-point (PTP) X.25 network	Low
IPAQENET6 [b]	OSA-Express: 10 Gigabit Ethernet Gigabit Ethernet 1000BASE-T Fast Ethernet	Local Area Network (LAN)	High
IPAQIDIO6 [b]	HiperSockets	Internal Local Area Network (LAN)	High
MPCPTP6 [b]	IUTSAMEH XCF	Point-to-point (PTP) Point-to-multipoint (PTM)	Low

a. Can also be used in conjunction with DYNAMICXCF
b. Supported network interface for IPv6 only

For further information about these protocols, refer to *z/OS V1R7.0 Communications Server: IP Configuration Reference*, SC31-8776.

4.2 Important and commonly used interfaces

This section discusses the commonly used interfaces supported by the IBM System z9 and zSeries servers. They are considered important because they deliver the best throughput and performance, as well as offer the most flexibility and highest levels of availability. These interfaces include:

- OSA-Express
- HiperSockets
- Dynamic Cross-system Coupling Facility (dynamic XCF)

High-bandwidth and high-speed networking technologies

The z/OS Communications Server's support of high-bandwidth and high-speed networking technologies is provided by OSA-Express (in QDIO mode) and HiperSockets (internal QDIO). The OSA-Express features comply with the most commonly used IEEE standards, used in LAN environments, while HiperSockets is used for transporting IP traffic between TCP/IP stacks running in a logical partitions (LPAR) at memory speed within a System z9 or zSeries server.

Queued Direct I/O (QDIO)

Queued Direct I/O is a highly efficient data transfer mechanism that satisfies the increasing volume of applications and increasing bandwidth demands. It dramatically reduces system overhead, and improves throughput by using system memory queues and a signaling protocol to directly exchange data between the OSA-Express microprocessor and network software, using data queues in main memory and utilizing Direct Memory Access (DMA).

The components that make up QDIO are Direct Memory Access (DMA), Priority Queuing, dynamic OSA Address Table building, LPAR-to-LPAR communication, and Internet Protocol (IP) Assist functions.

When an OSA-Express port is defined to run in QDIO mode with the z/OS Communications Server, it can only transport IP traffic. However, SNA can be transported over the IP connectivity using encapsulation technologies such as Enterprise Extender and TN3270.

HiperSockets implementation is based on the OSA-Express queued direct input/output (QDIO) protocol, hence HiperSockets is also called internal QDIO (iQDIO). The microcode emulates the link control layer of an OSA-Express QDIO interface. The communication is through system memory of the server via I/O queues. IP traffic is transferred at memory speeds between LPARs, eliminating the I/O subsystem overhead and external network delays.

4.2.1 OSA-Express (MPCIPA)

As mentioned, the OSA-Express can use the I/O architecture called QDIO when defined as channel type OSD. This architecture provides a highly optimized data transfer interface that eliminates the need for channel command words (CCWs) and interrupts during data transmission, resulting in accelerated TCP/IP packet transmission. This is done by providing a data queue between TCP/IP and the OSA-Express. OSA-Express utilizes a direct memory access (DMA) protocol to transfer the data to and from the TCP/IP stack. This design eliminates ESCON bus performance limitations.

The OSA-Express also provides the offloading of IP processing from the host, which is called IP assist (IPA). With IP assist, the OSA-Express offloads the following processing from the host:

- All MAC handling is done in the card. The TCP/IP stack no longer has to fully format the datagrams for LAN-specific media.
- ARP processing for identifying the physical address.
- Packet filtering, screening, and discarding of LAN packets.

Also in QDIO mode the OSA-Express feature receives configuration information from the host dynamically. This reduces configuration and setup time, eliminates duplicate data entry, and reduces the possibility of data entry errors and incompatible definitions.

Non-QDIO mode

Some OSA-Express features also support LCS (known as non-QDIO mode). Like any other channel-attached control unit and device, an OSA-Express feature can execute channel programs (CCW chains) and present I/O interrupts to the issuing applications. For non-QDIO mode, the OSA-Express features are defined as channel type OSE. The non-QDIO mode requires the use of the OSA/SF for setup and customization of the OSA-Express features. When an OSA-Express port is defined to run in non-QDIO mode with z/OS Communications Server, it can be used to transport SNA traffic, IP traffic, or both concurrently.

Figure 4-2 illustrates the much shorter I/O process of an OSA-Express port when in QDIO mode compared with non-QDIO mode (the same I/O path as the OSA-2 features). I/O interrupts and I/O path-lengths are minimized, resulting in significantly improved performance versus non-QDIO mode, reduction of System Assist Processor (SAP) utilization, improved response time, and server cycle reduction.

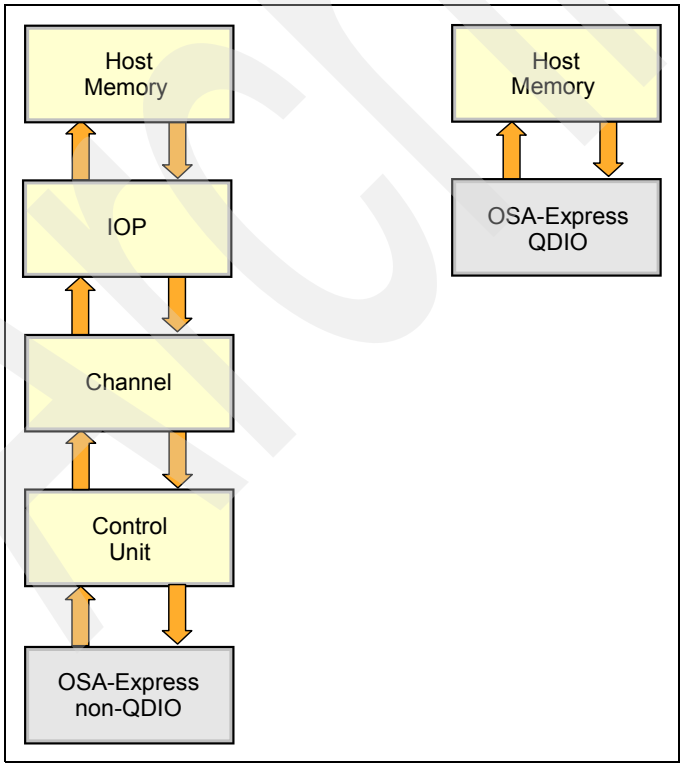


Figure 4-2 OSA-Express QDIO versus non-QDIO

Recommendation: We recommend the use of QDIO mode in conjunction with the supported OSA-Express Ethernet features wherever possible.

Table 4-2 lists the OSA-Express2 and OSA-Express Ethernet features that are available on the System z9 and zSeries servers. Included is the mode of operation in which they can run and the necessary TCP/IP and VTAM definition types.

Table 4-2 OSA-Express features

OSA-Express feature	Operation mode	TCP/IP device type	TCP/IP link type	VTAM definitions
10 GbE LR	QDIO	MPCIPA	IPAQENET	TRLE
GbE	QDIO	MPCIPA	IPAQENET	TRLE
1000BASE-T	QDIO	MPCIPA	IPAQENET	TRLE
	Non-QDIO	LCS	ETHERNet, 802.3, or ETHEROR802.3	N/A
FENET	QDIO	MPCIPA	IPAQENET	TRLE
	Non-QDIO	LCS	ETHERNet, 802.3, or ETHEROR802.3	N/A

OSA-Express VLAN support

The OSA-Express Ethernet features also support IEEE standards 802.1p/q, which describes priority tagging and VLAN identifier tagging. Deploying VLAN IDs allows a physical LAN to be partitioned or subdivided into discrete virtual LANs. This support is provided by the z/OS TCP/IP stack and OSA-Express in QDIO mode.

VLAN connection types

IEEE 802.1q VLANs operate by defining switch ports as members of virtual LANs. A z/OS TCP/IP stack can be assigned to a VLAN based on whether it is VLAN-aware or VLAN-unaware. A VLAN-aware stack understands VLAN memberships (which users belong to a particular VLAN) and VLAN formats.

Ports on a switch used to attach VLAN-unaware equipment are called *access ports*, while ports used to connect to other switches or VLAN-aware servers are known as *trunk ports*. Network packets generated by VLAN-aware equipment are marked with a *tag,* which identifies the packet to the VLAN.

Trunk mode

Trunk mode indicates that the switch should allow all VLAN ID tagged packets to pass through the switch port without altering the VLAN ID. This mode is intended for servers that are VLAN capable, and filters and processes all VLAN ID tagged packets. In trunk mode, the switch expects to see VLAN ID tagged packets inbound to the switch port.

Access mode

Access mode indicates that the switch should filter on specific VLAN IDs and only allow packets that match the configured VLAN IDs to pass through the switch port. The VLAN ID is then removed from the packet before it is sent to the server. That is, VLAN ID filtering is controlled by the switch. In access mode, the switch expects to see packets without VLAN ID tags inbound to the switch port.

Broadcast in VLANs

All ports that are members of the same VLAN, including trunk ports, operate as though they are part of the same physical network. When a multicast or broadcast frame is received from a device on a particular VLAN, the switch transmits the frame to all ports (both trunk and access) belonging to the same VLAN.

The only difference is that the frame transmitted onto the trunk port will have the VLAN tag intact, so that the VLAN-aware equipment at the other end of the link knows how to handle it.

VLAN isolation

An important point about VLANs is that they provide isolation. VLANs behave like separate physical networks, even though they may be contained within the same switch.

In order for devices in different VLANs to communicate, IP routing must occur. In the network shown in Figure 4-3, workstations in VLAN 100 and VLAN 101 cannot communicate, because there is no routing path between the two VLANs. Workstations in VLANs 100 and 102 can communicate, as long as the IP router to which they are attached is configured appropriately.

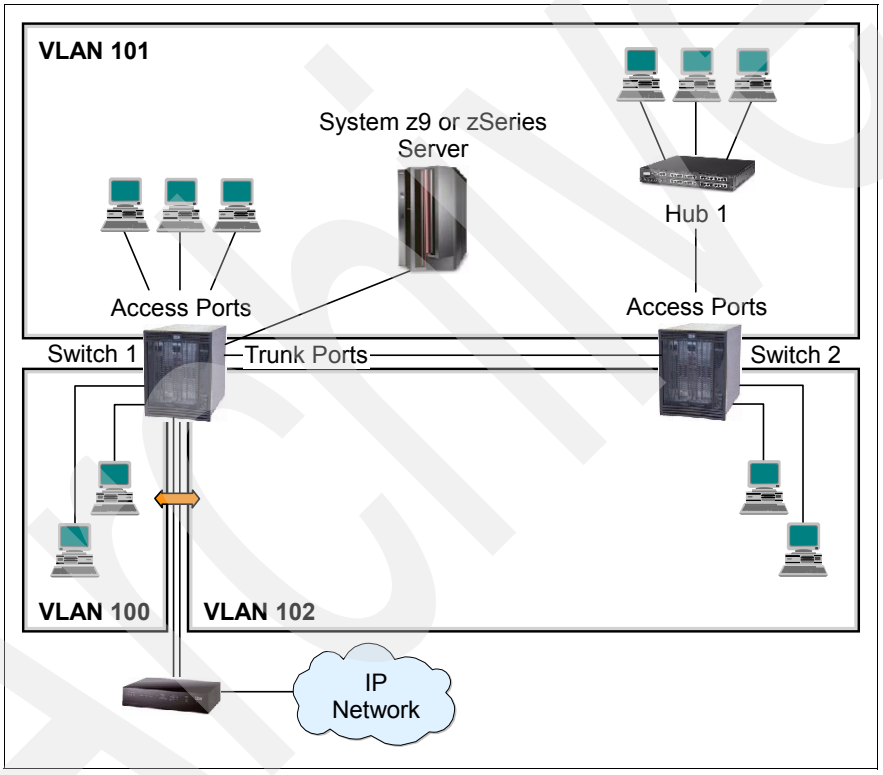


Figure 4-3 VLAN port types

In this example network, VLAN 100 only exists in Switch 1, because the trunk port to Switch 2 is not a member of VLAN 100. VLANs 101 and 102 span the two switches, since the trunk ports in each switch are members of both VLANs.

There is a single VLAN-aware router connected to Switch 1, which provides access to an external network to users in VLANs 100 and 102. The trunk port to which the router is attached is defined to VLANs 100 and 102, not to VLAN 101. Therefore, no workstations in VLAN 101 can reach the router. Hub 1 is a VLAN-unaware device, so the port in Switch 2 that it connects to is an access port.

This example illustrates two reasons why VLANs are generally used:

- Staff in different physical locations retain common access to resources.

  The server is used by staff in both buildings. By defining these workstations to the same VLAN, no additional configuration or equipment is required for either location to access the server, while at the same time ensuring that other staff do not obtain access.

- Consolidation of resource access.

  The external network has to be accessed by different staff in both buildings. Extending VLAN 102 across to the router port in Switch 1 (and configuring the router correctly) saves having to provide an additional link to the external network or the router from the other building.

VLAN support of Generic Attribute Registration Protocol - GVRP

GVRP is defined in the IEEE 802.1P standard for the control of IEEE 802.1Q VLANs. It can be used to help simplify networking administration and management of VLANs. With GVRP support, an OSA-Express2 port can register or de-register its VLAN IDs with a GVRP-capable switch and dynamically update its table as the VLANs change. Support of GVRP is exclusive to System z9 and is applicable to all of the OSA-Express2 features when in QDIO mode. It is supported by z/OS Communication Server 1.7 with PTF UQ06129 applied. Defining DYNVLANREG in the LINK statement of the OSA-Express2 port will enable GVRP.

OSA-Express router support

OSA-Express also provides primary (PRIRouter) and secondary (SECRouter) router support. This function allows a single TCP/IP stack, on a per-protocol (IPv4 and IPv6) basis, to register and act as a router stack base on a given OSA-Express port. Secondary routers can also be configured to provide for conditions in which the primary router becomes unavailable and the secondary router takes over for the primary router

VLAN and primary/secondary router support

The OSA-Express primary router support takes into consideration and interacts with the VLAN ID support (VLAN ID registration and tagging). OSA-Express supports a primary and secondary router on a per-VLAN basis (per registered VLAN ID). Therefore, if TCP/IP is configured with a specific VLAN ID and also configured as a primary or secondary router, that stack serves as a router for just that specific VLAN. This allows each OSA-Express (CHPID) to have a primary router per VLAN. Configuring multiple primary routers (one per VLAN) has many advantages and preserves traffic isolation for each VLAN.

For more information regarding OSA-Express features and capabilities, refer to *OSA-Express Implementation Guide*, SG24-5948.

4.2.2 HiperSockets (MPCIPA)

HiperSockets, also known as internal Queued Direct I/O (iQDIO), is a hardware feature that provides high-speed communicating LPAR-to-LPAR within the same sever (via memory). It also provides secure data flows between LPARs and high availability, providing there is no network attachment dependency or exposure to adapter failures.

The HiperSockets device is represented by the IQD CHPID and its associated subchannel devices. All LPARs that are configured in HCD to use the same IQD CHPID have internal connectivity and therefore have the capability to communicate using HiperSockets.

HiperSockets MPC group

VTAM will build a single HiperSockets MPC group using the subchannel devices associated with a single IQD CHPID. VTAM will use two subchannel devices for the read and write control devices, and 1 to 8 devices for data devices. Each TCP/IP stack will be assigned a single data device.

Therefore, in order to build the MPC group, there must be a minimum of three subchannel devices defined (within HCD) and associated with the same IQD CHPID. The maximum number of subchannel devices that VTAM will use is 10 (supporting 8 data devices or 8 TCP/IP stacks) per LPAR or MVS image.

When the server supports HiperSockets and the CHPIDs have been configured in HCD (IOCP), TCP/IP connectivity is provided if:

- DYNAMICXCF is configured on the IPCONFIG (IPv4) or the IPCONFIG6 (IPv6) statements.
- A user-defined HiperSockets (MPCIPA) DEVICE and LINK for IPv4 or (IPAQIDIO) INTERFACE for IPv6 is configured and started.

IQD CHPID can be viewed as a *logical LAN* within the server. The zSeries 900 and 800 servers allow up to four separate IQD CHPIDs, creating the capability of having up to four separate logical LANs within the same server. System z9 and zSeries 990 and 890 servers support up to 16 IQD CHPIDs per server.

Each IQD CHPID can be assigned to a set of LPARs (configured in HCD), making it possible to isolate these LPARs in separate logical LANs, as shown in of Figure 4-4.

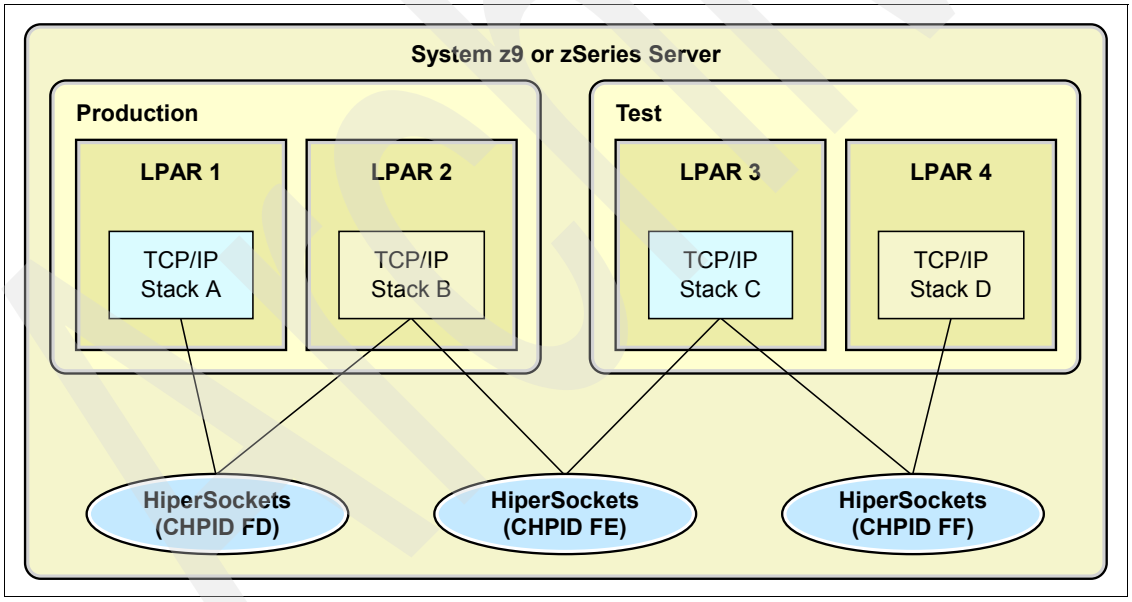


Figure 4-4 HiperSockets - Multiple logical LANs

HiperSockets DYNAMICXCF connectivity

z/OS images within the same server with DYNAMICXCF coded will use the HiperSockets DYNAMICXCF connectivity instead of the standard XCF connectivity, under these conditions:

- The TCP/IP stacks must be on the same server.
- For the DYNAMICXCF HiperSockets device (IUTIQDIO), the stacks must be using the same IQD CHPID, even with different channel subsystems (spanning).

- The stacks must be configured (HCD) to use HiperSockets.
- For IPv6 HiperSockets connectivity, both stacks must be at the z/OS V1R7 level.
- The initial HiperSockets activation must complete successfully.

When an IPv4 DYNAMICXCF HiperSockets device and link are created and successfully activated, a subnetwork route is created across the HiperSockets link. The subnetwork is created by using the DYNAMICXCF IP address and mask. This allows any LPAR within the same server to be reached, even ones that are not within the sysplex. To do that the LPAR that is outside of the sysplex environment must define at least one IP address for the HiperSockets endpoint that is within the subnetwork defined by the DYNAMICXCF IP address and mask.

When multiple stacks reside within the same LPAR that supports HiperSockets, both IUTSAMEH and HiperSockets links or interfaces will coexist. In this case, it is possible to transfer data across either link. Because IUTSAMEH links have better performance, it is always better to use them for intra-stack communication. A host route will be created by DYNAMICXCF processing across the IUTSAMEH link, but not across the HiperSockets link.

HiperSockets Accelerator

The Communications Server leverages the technological advances and high-performing nature of the I/O processing offered by HiperSockets with the IBM System z9 or zSeries servers and OSA-Express, using the QDIO architecture. This is achieved by optimizing IP packet forwarding processing that occurs across these two types of technologies. This function is referred to as HiperSockets Accelerator. It is a configurable option, and is activated by defining the IQDIORouting option on the IPCONFIG statement.

When the TCP/IP stack is configured with HiperSockets Accelerator, it allows IP packets received from HiperSockets to be forwarded to an OSA-Express port (or vice versa) without the need for those IP packets to be processed by the TCP/IP stack.

When using this function, one or more LPARs contain the *routing* stack, which manages connectivity via OSA-Express ports to the LAN, while the other LPARs connect to the routing stack using the HiperSockets, as shown in Figure 4-5 on page 112.

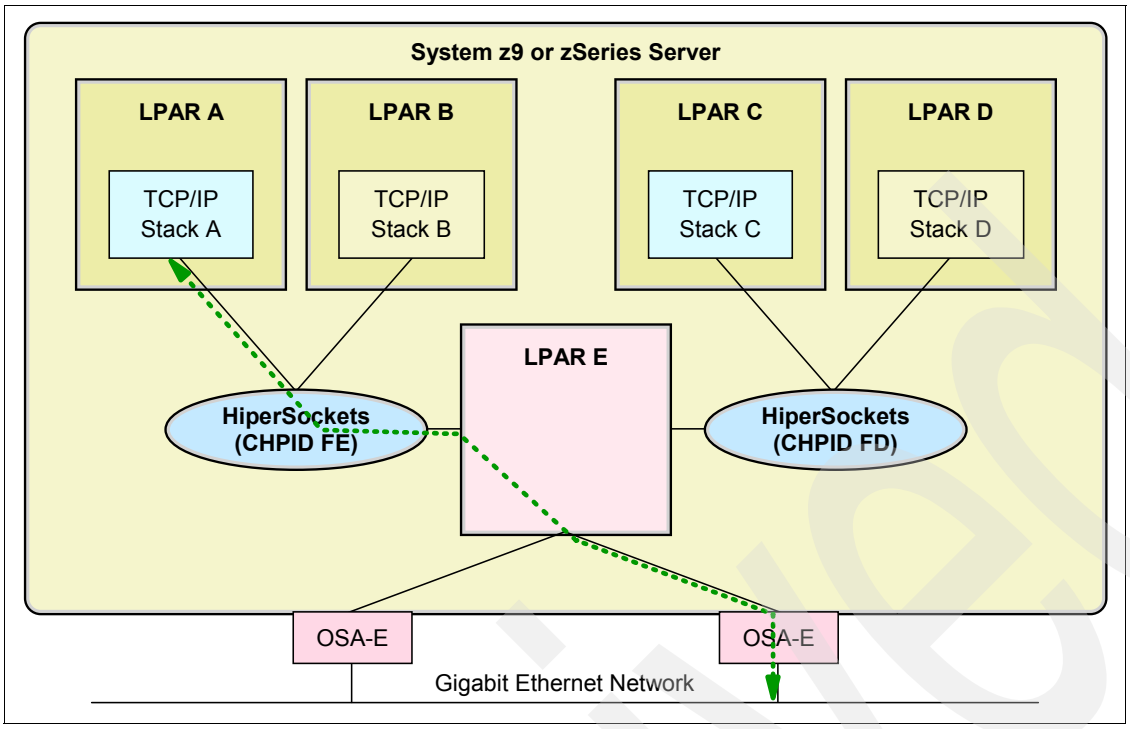


Figure 4-5 HiperSockets Accelerator

Note: This example is intended purely to demonstrate IP traffic flow. We do not recommend implementing HiperSockets Accelerator using a single LPAR.

For more information about HiperSockets, refer to *zSeries HiperSockets*, SG24-6816.

4.2.3 Dynamic XCF

From an IP topology perspective, DYNAMICXCF establishes fully meshed IP connectivity to all other z/OS TCP/IP stacks in the Sysplex. We only need one end-point specification in each stack for fully meshed connectivity to all other stacks in the Sysplex. When a new stack gets started, Dynamic XCF connectivity is automatically established.

Note: Only one dynamic XCF network is supported per Sysplex.

Dynamic XCF is required to support Sysplex Distributor and nondisruptive dynamic VIPA movement (discussed in detail in *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 3 - High Availability, Scalability, and Performance*, SG24-7171).

Dynamic XCF uses Sysplex Sockets support, allowing the stacks to communicate with each other and exchange information such as VTAM CPNAMEs, MVS SYSCLONE value, and IP addresses. The dynamic XCF definition is activated by coding the IPCONFIG DYNAMICXCF parameter in the TCP/IP profile.

Dynamic XCF creates definitions for DEVICE, LINK, HOME, and BSDROUTINGPARMS statements and the START statement dynamically. When activated, the dynamic XCF devices and links appear to the stack as though they had been defined in the TCP/IP profile. They can be displayed using standard commands, and they can be stopped and started.

During TCP/IP initialization, the stack joins the XCF group, ISTXCF, through VTAM. When other stacks in the group discover the new stack, the definitions are created automatically, the links are activated, and the remote IP address for each link is added to the routing table. After the remote IP address has been added, IP traffic can flow across one of the following interfaces:

- IUTSAMEH (within the same LPAR)
- HiperSockets (within the same server)
- XCF signaling (different server, either using the coupling facility link or a CTC connection)

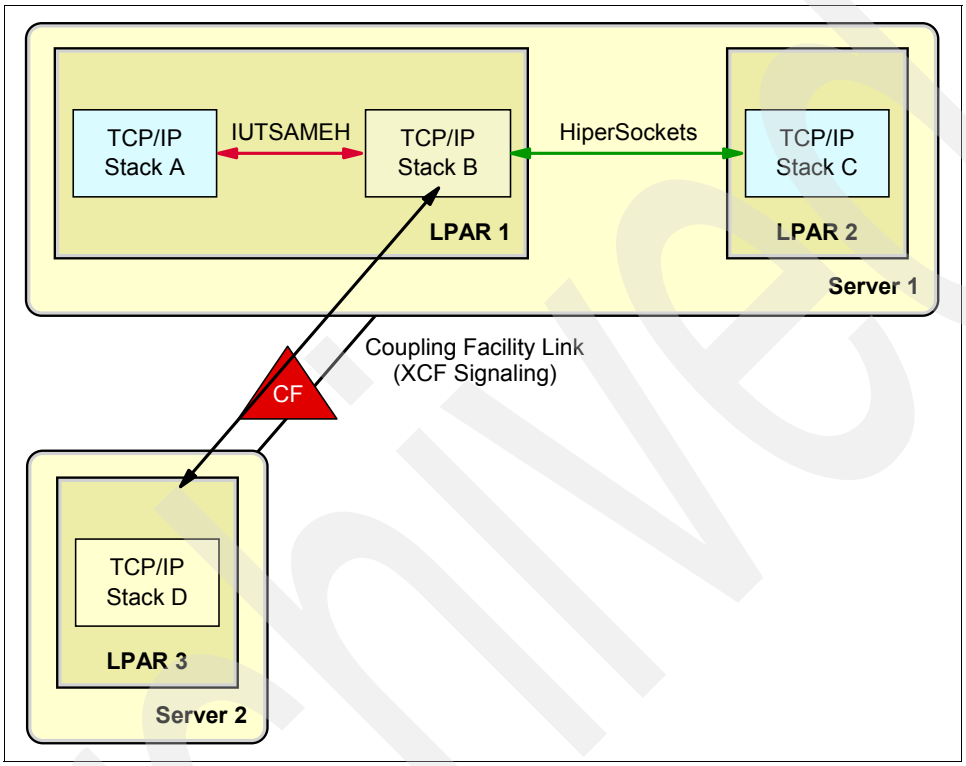


Figure 4-6 Dynamic XCF support

For additional information about dynamic XCF, Sysplex Distributor, and nondisruptive dynamic VIPA movement refer to *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 3 - High Availability, Scalability, and Performance*, SG24-7171.

4.3 The common design scenarios for connectivity

To design a connectivity scenario in a z/OS environment we must take the following into account:

- As a server environment, the network connectivity to the external corporate network should be carefully designed to provide a high-availability environment avoiding single-points of failures.
- If a z/OS LPAR is seen as a stand-alone server environment on the corporate network it should be designed as an end-point.
- If a z/OS LPAR will be used as a front-end concentrator (for example, making use of HiperSockets Accelerator), it should be designed as an intermediate network or node.

Recommendation: Although there are specialized cases where multiple stacks per LPAR can provide value, we in general recommend implementing only one TCP/IP stack per LPAR. The reasons for this recommendation are as follows:

- A TCP/IP stack is capable of exploiting all available resources defined to the LPAR in which it is running. Therefore, starting multiple stacks will not yield any increase in throughput.
- When running multiple TCP/IP stacks, additional system resources, such as memory, CPU cycles, and storage, are required.
- Multiple TCP/IP stacks add a significant level of complexity to TCP/IP system administration tasks.
- It is not necessary to start multiple stacks to support multiple instances of an application on a given port number, such as a test HTTP server on port 80 and a production HTTP server also on port 80. This type of support can instead be implemented using BIND-specific support where the two HTTP server instances are each associated to port 80 with their own IP address, via the BIND option on the PORT reservation statement.

One example where multiple stacks can have value is when an LPAR needs to be connected to multiple isolated security zones in such a way that there is no network level connectivity between the security zones. In this case, a TCP/IP stack per security zone can be used to provide that level of isolation, without any network connectivity between the stacks.

Based on these considerations, we are going to show what we consider to be the best-practice scenarios to build up a z/OS Communications Server TCP/IP configuration scenario, using OSA-Express (QDIO), HiperSockets (iQDIO), and dynamic XCF.

This section focuses on the interface implementation only, which means establishing Layer 2 and a subset of Layer 3 (IP addressing) connectivity. For connectivity beyond the immediate LAN environment, also refer to Chapter 5, “Routing” on page 139, for IP routing details.

To build our test scenario, we used the system environment shown in Figure 4-7 on page 115. Note that we are defining our LPARs as end-points.

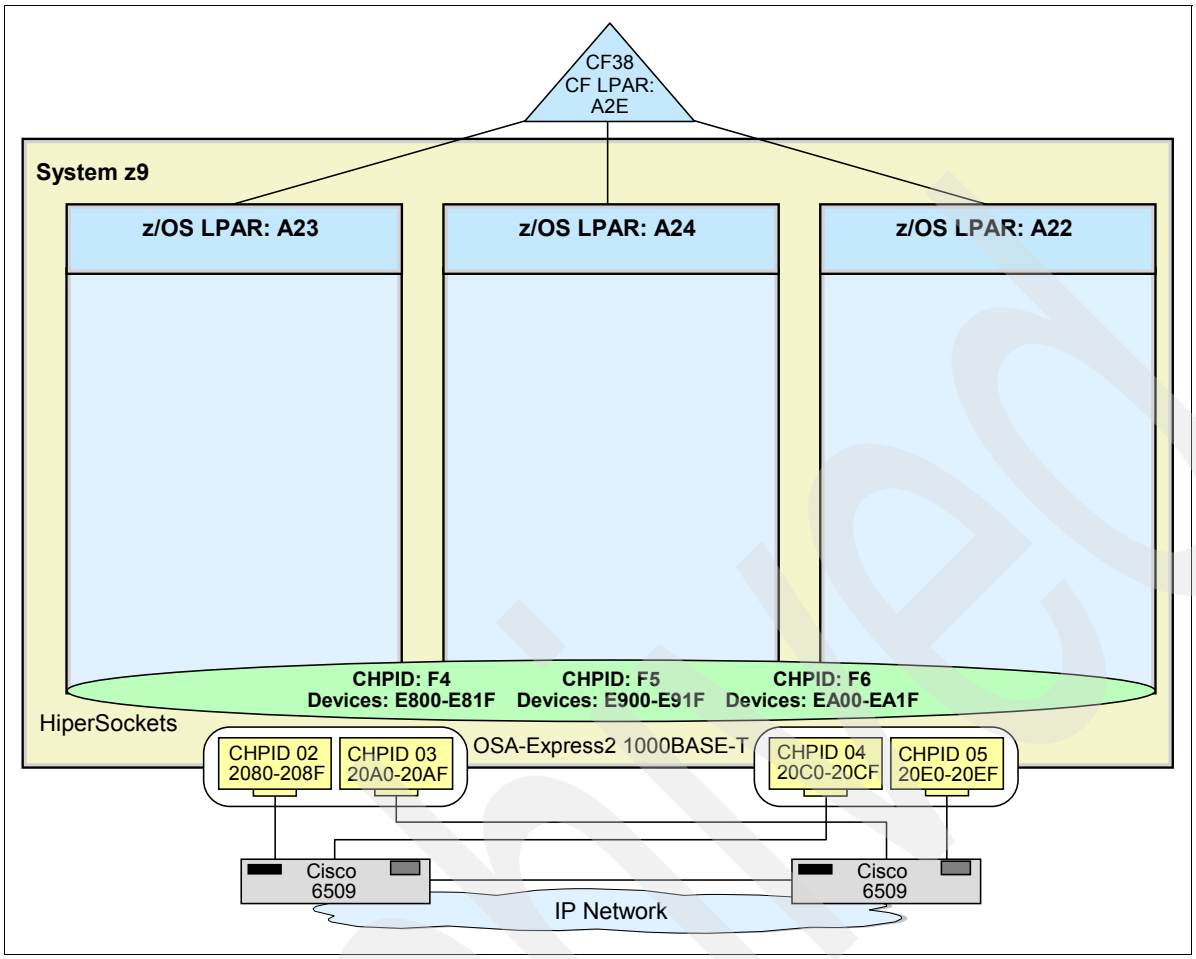


Figure 4-7 z/OS connectivity scenario

With this environment, we will build connectivity scenarios that will include two OSA-Express 1000BASE-T features with two ports each. The ports will connect to the LAN environment (two Cisco switches). We will also implement a HiperSockets internal LAN to interconnect all LPARs within the same System z9. And, finally, we will use dynamic XCF connectivity, provided by the sysplex environment, for IP traffic. The scenarios we will discuss are as follows:

- OSA-Express connectivity
- HiperSockets connectivity
- Dynamic XCF connectivity

The other interfaces described at the beginning of this chapter are still supported and can be used to connect to IP networks or IP hosts. However, in the following sections, we only show how to implement what we considered to be the most commonly used interfaces.

4.3.1 OSA-Express connectivity

Configuring an OSA-Express (QDIO mode) in a single stack scenario is the simplest way to implement your z/OS TCP/IP stack into a LAN environment. This scenario, however, still needs to be planned to avoid any single-points of failure. Hence, we must have at least two OSA-Express features connecting to two different switches in the network.

Since we are dealing with multiple LPARs in our server, for redundancy purposes we have shared the OSA-Express ports (CHPID type OSD) across all LPARs.

In our test scenario, we have two OSA-Express 1000BASE-T features, each with two ports. This allows us to have four CHPIDs, 02, 03, 04, and 05, shared by our three LPARs; SC30, SC31, and SC32 (see Figure 4-7 on page 115).

To make better use of our OSA-Express ports and to control data traffic patterns, we defined one port on each OSA-Express feature with a separate VLAN ID, creating two subnetworks to be used by all LPARs. In a high availability configuration, these OSA-Express ports will be the path to all of our IP addresses for the LAN environment.

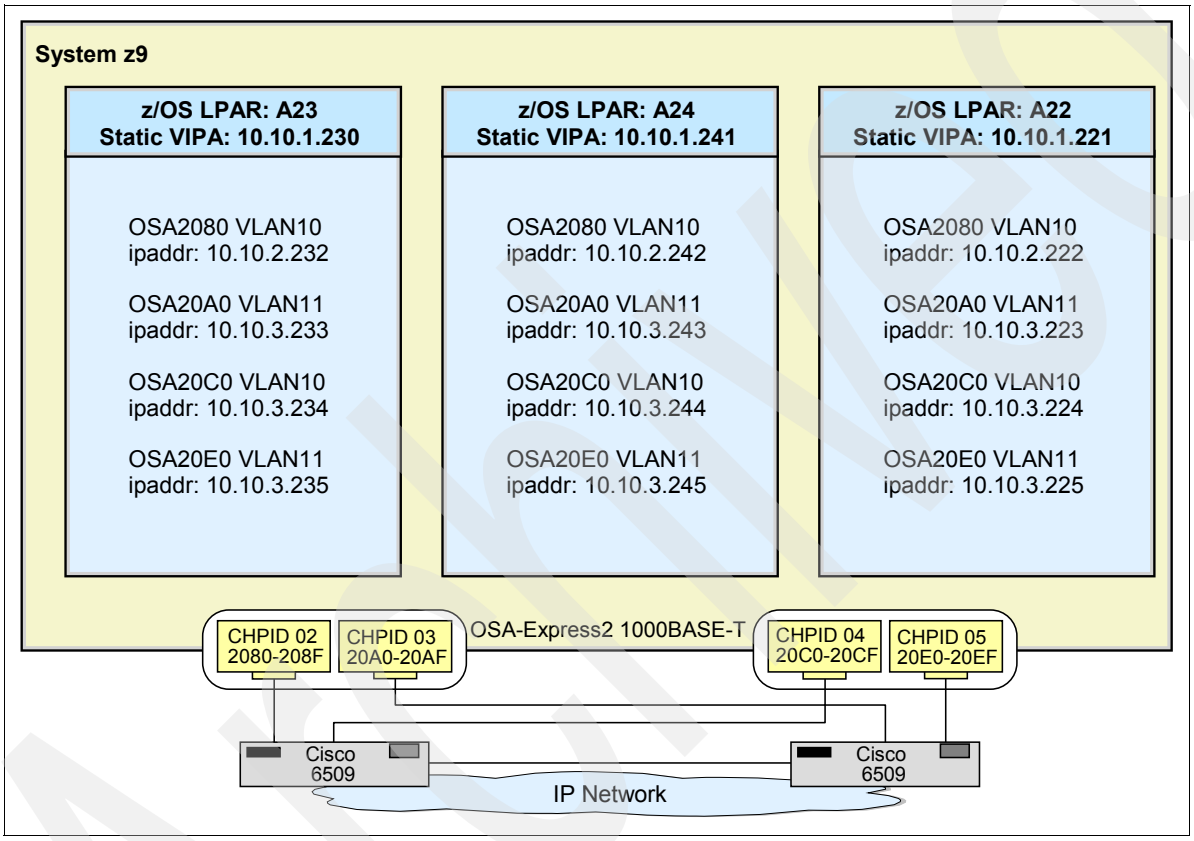


Figure 4-8 OSA-Express (QDIO) implementation

Dependencies

To implement this scenario, we have the following dependencies:

- The OSA-Express port must be defined as CHPID type OSD to the server using HCD or IOCP to enable QDIO. This CHPID must be defined as shared to all LPARs that will use the OSA-Express port (see Example 4-1 on page 122).
- To define an OSA-Express port in QDIO mode, use the MPCIPA DEVICE statement, specifying the PORTNAME value from the TRLE definition as the device_name. The TRLE must be defined as MPCLEVEL=QDIO.
- The Virtual LAN identifiers (VLAN IDs) defined to each OSA-Express port must be recognized by each switch.
- The switch ports where the OSA-Express ports are connected must be configured in trunk mode.

Advantages

The advantages of implementing an OSA-Express port in QDIO mode over non-QDI O (for example, LCS) are:

- IP-Assist to handle MAC addressing, ARP processing, some filtering
- Dynamically maintains the OSA Address Table in a shared environment
- Supports high-speed LPAR-to-LPAR communication
- Direct Memory Access (DMA) protocol reduces IO interrupts
- Supports outbound priority queuing
- Network definitions built dynamically
- Only QDIO is capable of delivering high-bandwidth and high-speed networking
- Supports VLAN IDs to isolate services without creating independent TCP/IP stacks

Considerations

When planning connectivity for a LAN environment, there may not be a requirement to isolate data traffic or services for certain servers or clients as we have shown in this scenario. Hence, VLAN IDs can be omitted.

If there is a requirement for VLANs, however, we recommend adding the VLAN IDs to your IP addressing scheme to aid in the mapping of IP addresses to VLANs based on data traffic patterns or access to resources.

Also, to simplify administration and management of VLANs consider using Generic Attribute VLAN Registration Protocol wherever possible. For details, refer to “VLAN support of Generic Attribute Registration Protocol - GVRP” on page 109.

4.3.2 HiperSockets connectivity

HiperSockets provides very fast TCP/IP communications between different Logical Partitions (LPARs) through the system memory of the System z9 or zSeries server. The LPARs that are so connected form an *internal LAN*, passing data between the LPARs at memory speeds, and thereby totally eliminating the I/O subsystem overhead and external network delays.

To create this scenario, we define the HiperSockets, which is represented by the IQD CHPID and its associated devices. All LPARs that are configured to use the shared IQD CHPID have internal connectivity and therefore have the capability to communicate using HiperSockets.

In our test environment we will use two IQD CHPIDs, F4 and F5. Each one of them will create a separate logical LAN with its own subnetwork. Figure 4-9 on page 118 depicts these interfaces to our test scenario.

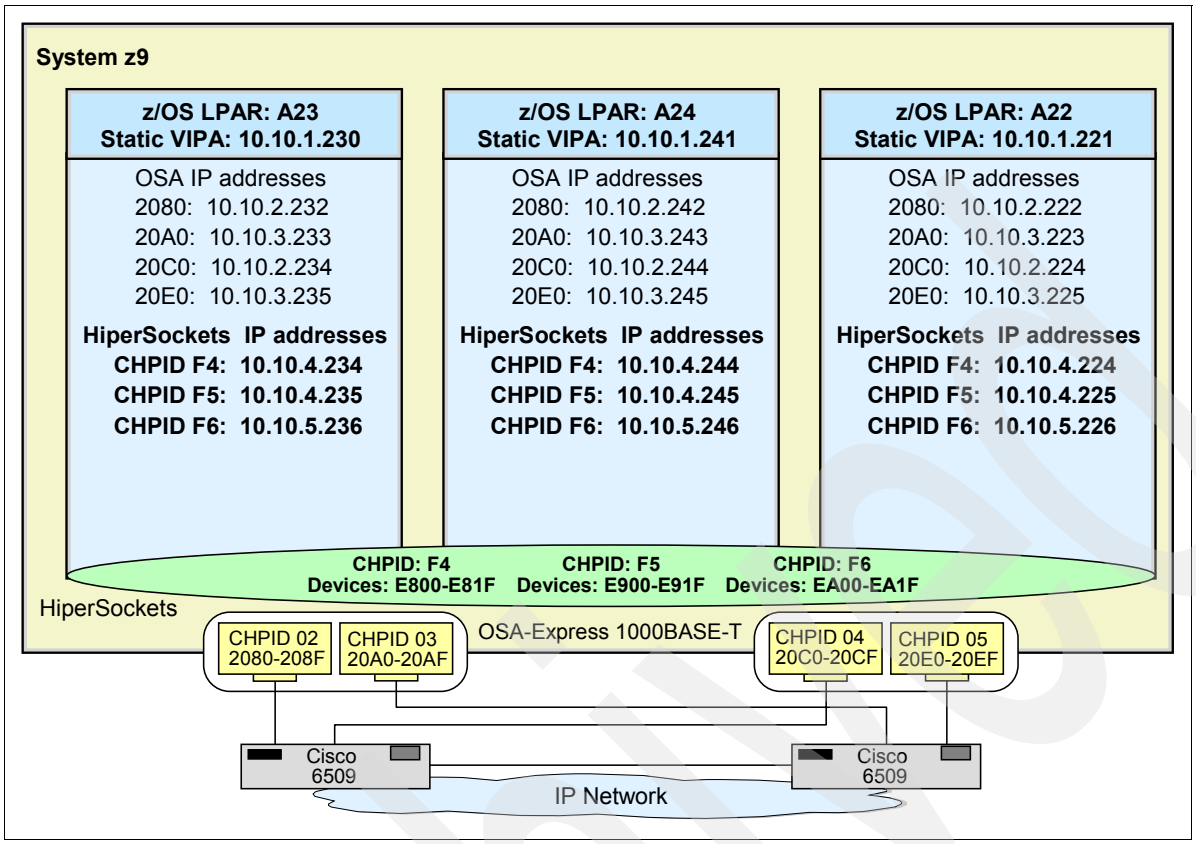


Figure 4-9 HiperSockets implementation scenario

Dependencies

The dependencies are:

- The HiperSockets must be defined as CHPID type IQD to the server using HCD or IOCP. This CHPID must be defined as shared to all LPARs that will be part of the HiperSockets internal LAN (see Example 4-1 on page 122).
- When explicitly defined, a correspondent TRLE must be created in VTAM using a port name IUTIQD*xx*, where *xx* is the CHPID number.
- When more than one IQD CHPID is configured to a specific LPAR, VTAM start option IQDCHPID must be used to specify which specific IQD CHPID this LPAR should use.

Note: In both cases, the TRLE is dynamically built by VTAM. The IQDCHPID VTAM start option controls the VTAM selection of which IQD CHPID (and related devices) to include in the HiperSockets MPC group (IUTIQDIO) when it is dynamically built for DYNAMICXCF connectivity.

For additional details regarding how to configure a user-defined HiperSockets device or interface, refer to *z/OS V1R7.0 Communications Server: IP Configuration Reference*, SC31-8776.

Advantages

HiperSockets can be used to communicate among consolidated servers within a single System z9 or zSeries server. All the hardware boxes running these separate servers can be

eliminated, along with the cost, complexity, and maintenance of the networking components that interconnect them.

Consolidated servers that have to access corporate data residing on the System z9 or zSeries server can do so at memory speeds, bypassing all the network overhead and delays.

HiperSockets can be customized to accommodate varying traffic sizes. (In contrast, LANs, like Ethernet and Token Ring, have a maximum frame size predefined by their architecture.) With HiperSockets, a maximum frame size can be defined according to the traffic characteristics transported for each of the four possible HiperSockets.

Since there is no server-to-server traffic outside the System z9 or zSeries server, a much higher level of network availability, security, simplicity, performance, and cost effectiveness is achieved as compared with servers communicating across a LAN. For example:

- Since HiperSockets has no external components. It provides a very secure connection. For security purposes, servers can be connected to different HiperSockets. All security features, like firewall filtering, are available for HiperSockets interfaces as they are with other TCP/IP network interfaces.
- HiperSockets looks like any other TCP/IP interface; therefore, it is transparent to applications and supported operating systems.
- HiperSockets can also improve TCP/IP communications within a sysplex environment when the DYNAMICXCF is used (for example, in cases where Sysplex Distributor uses HiperSockets within the same CEC to transfer IP packets to the target systems).

Considerations

The z/OS Communications Server does not support VLANs in conjunction with HiperSockets. Therefore, if you have a requirement to restrict data traffic flow among the certain LPARs in the server, we recommend that you connect those LPARs to a separate HiperSockets.

4.3.3 Dynamic XCF connectivity

The last connectivity scenario we are going to add to our test environment is to connect all images within the same Sysplex environment through a dynamic XCF connection, created by the DYNAMICXCF definition in the TCP/IP profile.

Once defined, DYNAMICXCF will provide connectivity between stacks under the same LPAR by using the IUTSAMEH device (SAMEHOST) and between LPARs through HiperSockets using a IUTiQDIO device. To connect other z/OS images or other servers, an XCF coupling facility link is created.

In our test scenario we use DYNAMICXCF through HiperSockets with IQD CHPID F7. So by defining the DYNAMICXCF statement, we will create the XCF subnetwork through HiperSockets, as seen in Figure 4-10 on page 120.

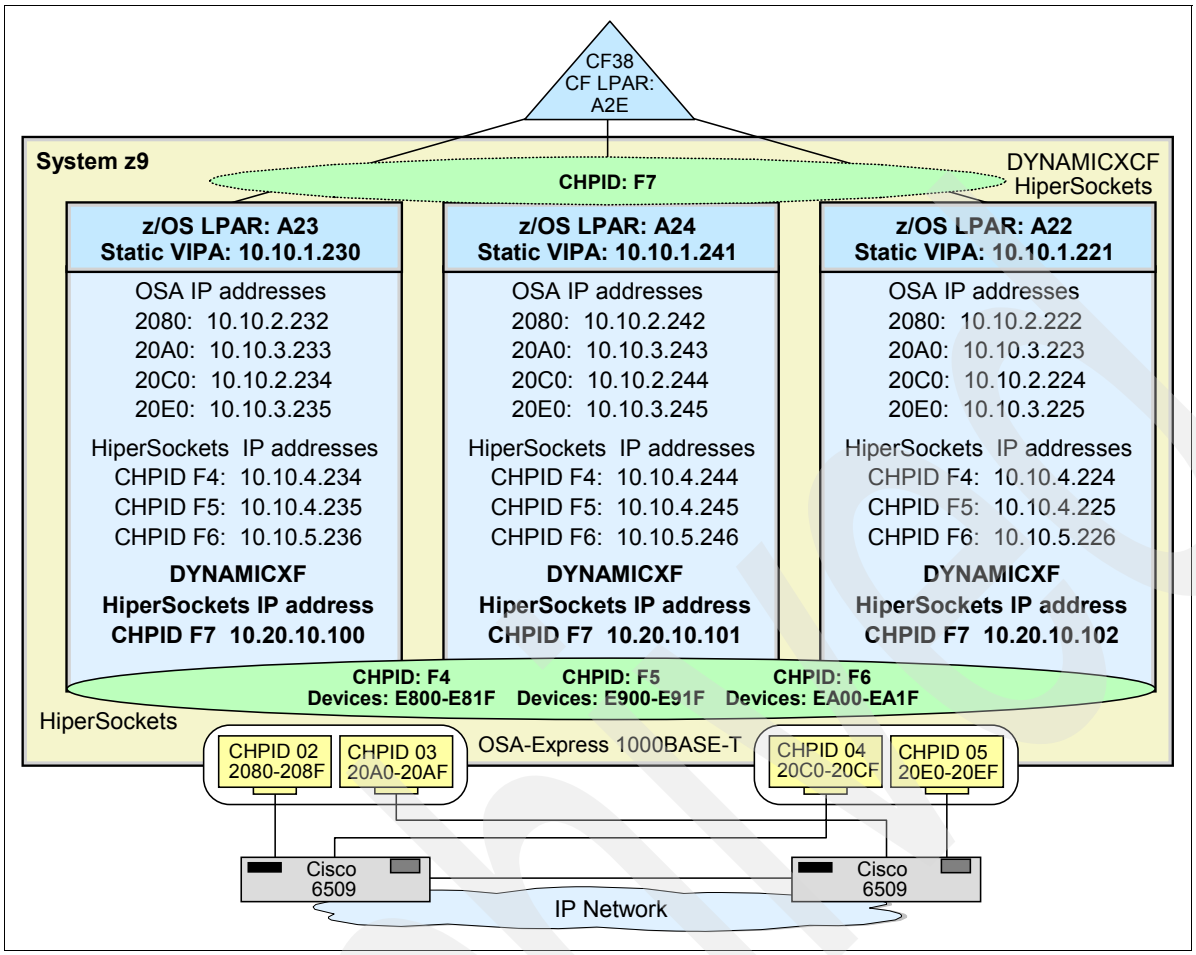


Figure 4-10 XCF implementation scenario with HiperSockets

Dependencies

The dependencies are:

- All z/OS hosts must belong to the same sysplex.
- VTAM must have XCF communications enabled by specifying XCFINIT=YES or XCFINIT=DEFINE as a startup parameter or by activating the VTAM XCF local SNA major node, ISTLSXCF. For details about configuration, refer to *z/OS V1R7.0 Communications Server SNA Network Implementation*, SC31-8777-05.
- DYNAMICXCF must be specified in the TCP/IP profile of each stack.
- The IQD CHPID being used for the DYNAMICXCF device cannot be the user-defined HiperSockets device (IQD CHPID). To avoid this, a VTAM start options, IQDCHPID, can be used to identify which IQD CHPID will be used by DYNAMICXCF.

Advantages

You have a choice of defining the XCF connectivity to other TCP/IP stacks individually or using the dynamic XCF definition facility. Dynamic XCF significantly reduces the number of definitions that you need to create whenever a new system joins the sysplex or when you need to start up a new TCP/IP stack. These changes become more numerous as the number of stacks and systems in the sysplex grows. This could lead to configuration errors. With dynamic XCF you do not need to change the definitions of the existing stacks in order to accommodate the new stack.

Considerations

The z/OS Communications Server V1R7 has improved and optimized Sysplex IP routing. In a Sysplex environment, you may prefer to use a connection other than a Coupling Facility link for cross-server connectivity because XCF is heavily used by other workloads (in particular, for distributed application data sharing). This option can be configured with the VIPAROUTE statement in the VIPADYNAMIC statement and allows for the use of OSA-Express features such as Gigabit Ethernet and 10 Gigabit Ethernet. For details refer to *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 3 - High Availability, Scalability, and Performance*, SG24-7171.

4.4 How connectivity is implemented

In this section we show the implementation details for the most commonly used interfaces, which include:

- "OSA-Express implementation with VLAN ID" on page 122
- "HiperSockets implementation" on page 125
- "DYNAMICXCF implementation" on page 127

When these implementation tasks are finished, we will have the configuration shown in Figure 4-11.

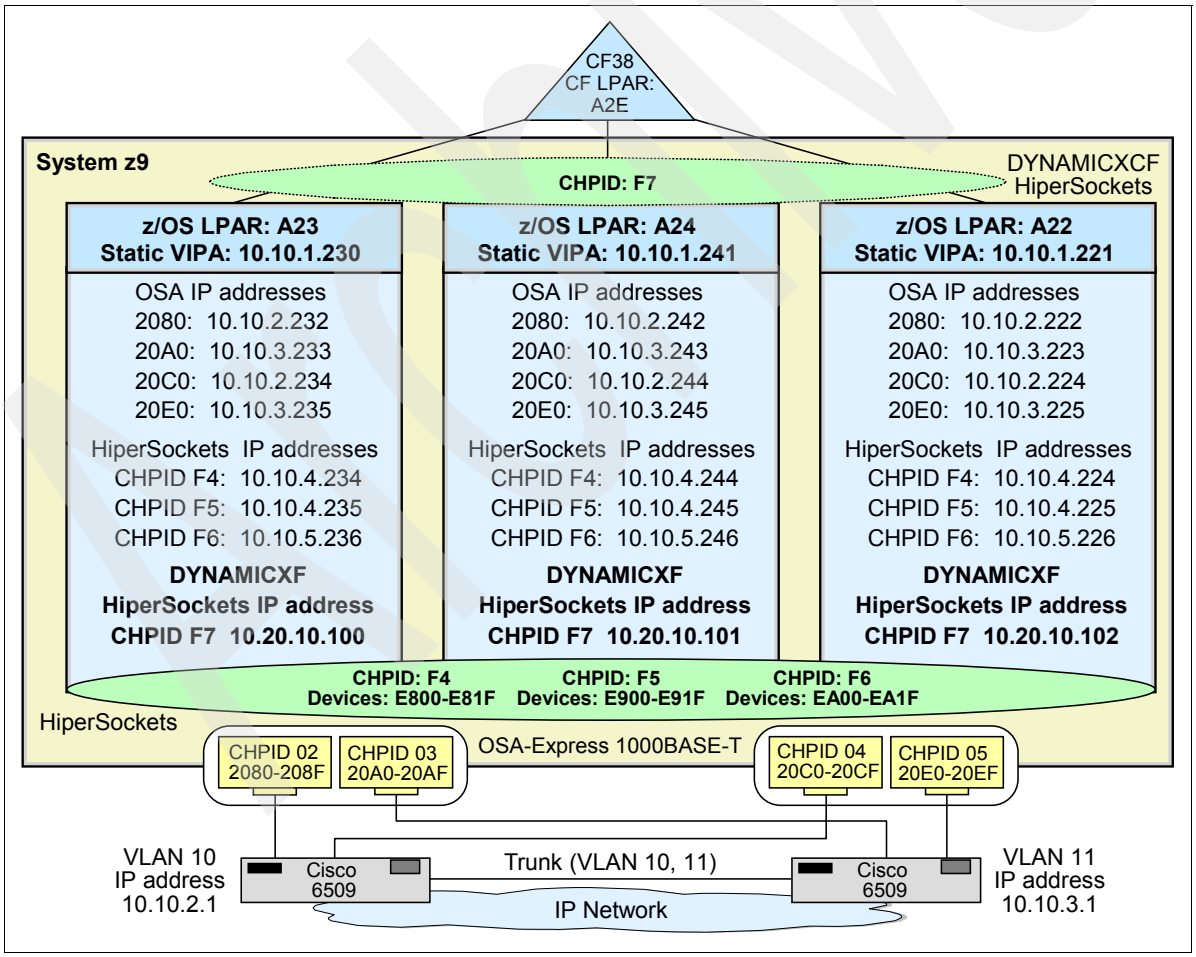


Figure 4-11 ITSO test environment

Example 4-1 depicts some of the IOCP statements we used in our test environment (OSA-Express CHPID 02 and HiperSockets CHPID F4). These statements are required by the input/output subsystem and the operating system. Since all our OSA-Express and HiperSockets connectivity will be used across all three LPARs, we defined the CHPIDs as shared.

Example 4-1 IOCP statements

```
ID    MSG1='IODF46',MSG2='SYS6.IODF46 - 2006-01-16 12:30',    *
            SYSTEM=(2094,1),                                    *
            TOK=('SCZP101',00800221991E2094123016290106016F00000000,*
            00000000,'06-01-16','12:30:16','SYS6','IODF46')
         RESOURCE PARTITION=((CSS(0),(A0A,A),(A0B,B),(A0C,C),(A0D,D),(A*
            0E,E),(A0F,F),(A01,1),(A02,2),(A03,3),(A04,4),(A05,5),(A*
            06,6),(A07,7),(A08,8),(A09,9)),(CSS(1),(A1A,A),(A1B,B),(*
            A1C,C),(A1D,D),(A1E,E),(A1F,F),(A11,1),(A12,2),(A13,3),(*
            A14,4),(A15,5),(A16,6),(A17,7),(A18,8),(A19,9)),(CSS(2),*
            (A2A,A),(A2B,B),(A2C,C),(A2D,D),(A2E,E),(A2F,F),(A21,1),*
            (A22,2),(A23,3),(A24,4),(A25,5),(A26,6),(A27,7),(A28,8),*
            (A29,9))),                                          *
            MAXDEV=((CSS(0),65280,0),(CSS(1),65280,0),(CSS(2),65280,*
         CHPID PATH=(CSS(2),02),SHARED,PARTITION=((A22,A23,A24),(=)),  *
            PCHID=390,TYPE=OSD
         CHPID PATH=(CSS(2),F4),SHARED,PARTITION=((A22,A23,A24),(=)),  *
            TYPE=IQD
         CNTLUNIT CUNUMBR=2080,PATH=((CSS(2),02)),UNIT=OSA
         IODEVICE ADDRESS=(2080,015),UNITADD=00,CUNUMBR=(2080),UNIT=OSA
         IODEVICE ADDRESS=208F,UNITADD=FE,CUNUMBR=(2080),UNIT=OSAD
         CNTLUNIT CUNUMBR=E800,PATH=((CSS(2),F4)),UNIT=IQD
         IODEVICE ADDRESS=(E800,032),CUNUMBR=(E800),UNIT=IQD
```

4.4.1 OSA-Express implementation with VLAN ID

To implement OSA-Express (QDIO) in our test environment we followed these tasks:

1. Verify the switch port configuration.
2. Define a TRLE in VTAM to represent each OSA-Express port.

In the TCP/IP profile:

1. Create DEVICE and LINK statements for each OSA-Express port.
2. Create a HOME address to each defined LINK.
3. Create a direct route statement for each LINK.

Figure 4-12 on page 123 shows the OSA-Express implementation scenario.

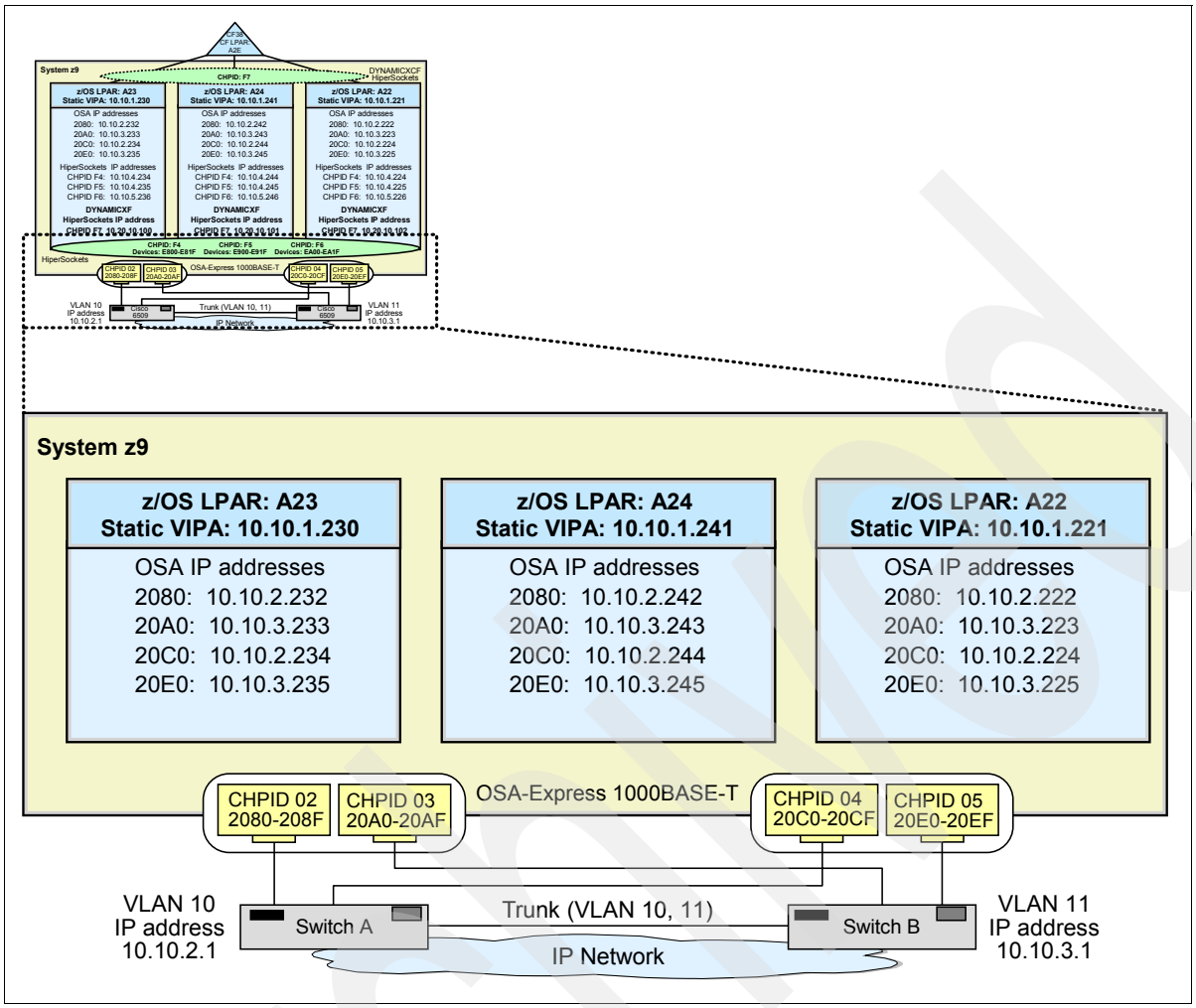


Figure 4-12 OSA-Express implementation scenario

Verify the switch port configuration

It is important to be aware of the switch configuration and definitions to which the OSA-Express ports will be connected. You will need confirm following information:

- The switch ports to which the OSA-Express ports are connected. Table 4-3 shows OSA-Express and switch port assignment with VLAN IDs.

Table 4-3 OSA-Express and switch port assignment with VLAN IDs

OSA-Express port	Connects to switch	Switch port	VLAN ID
CHPID 02 (2080)	Switch A	Interface FAST 2/1	10
CHPID 03 (20A0)	Switch B	interface GIGA 1/3	11
CHPID 04(20C0)	Switch A	Interface FAST 2/3	10
CHPID 04(20E0)	Switch B	Interface GIGA 1/5	11

- The IP subnetwork and mask; we used the following:
  - Subnetwork 10.10.2.0, mask 255.255.255.0 for VLAN 10
  - Subnetwork 10.10.3.0, mask 255.255.255.0 for VLAN 11

- The appropriate switch ports should be defined in trunk mode, as shown in Example 4-2.

Example 4-2 Switch port definition from Switch B port 1/3

```
interface GigabitEthernet1/3
 switchport
 switchport trunk encapsulation dot1q
 switchport mode trunk
 no ip address
```

Define a TRLE in VTAM to represent each OSA-Express port

Each OSA-Express port must have a TRLE definition defined (see Example 4-3). The PORTNAME (1) must match the device name of the DEVICE definition in the TCP/IP profile. The statement MPCLEVEL (2) must be specified as QDIO.

Example 4-3 TRLE definition

```
OSA2080  VBUILD TYPE=TRL
OSA2080P TRLE  LNCTL=MPC,                                      *
               READ=2080,                                      *
               WRITE=2081,                                     *
               DATAPATH=(2082-2087),                           *
             1 PORTNAME=OSA2080,                               *
             2 MPCLEVEL=QDIO
```

For all OSA-Express ports in our scenarios, we used the following PORTNAMES:

```
OSA2080
OSA20A0
OSA20C0
OSA20E0
```

Create DEVICE and LINK statements for each OSA-Express port

The next step is to create the device and link statements for each OSA-Express port, as shown in Example 4-4. An OSA-Express port must be defined as an MPCIPA device type 1.

The link definition describes the type of transport used, in our case, QDIO Ethernet, defined as IPAQENET 2. VLAN ID 3 defines the VLAN number the packets will be tagged with as they are being sent out to the switch.

Example 4-4 OSA-Express device and link definitions

```
;Osa definitions... relates to sys1.vtamlst trle definitions
;major nodes : osa2080,osa20a0,osa20c0 and osa20e0
 DEVICE OSA2080  MPCIPA 1
 LINK   OSA2080LNK  IPAQENET 2     OSA2080 VLANID 10 3
 DEVICE OSA20A0  MPCIPA
 LINK   OSA20A0LNK  IPAQENET      OSA20A0 VLANID 11
 DEVICE OSA20C0  MPCIPA
 LINK   OSA20C0LNK  IPAQENET      OSA20C0 VLANID 10
 DEVICE OSA20E0  MPCIPA
 LINK   OSA20E0LNK  IPAQENET      OSA20E0 VLANID 11
```

Create a HOME address to each defined LINK

Each link configured must have its own IP address. Our OSA-Express ports are defined with the IP addresses shown in Example 4-5 on page 125.

Example 4-5 OSA-Express HOME addresses

```
HOME
   10.10.2.232    OSA2080LNK
   10.10.3.233    OSA20A0LNK
   10.10.2.234    OSA20C0LNK
   10.10.3.235    OSA20E0LNK
```

Define the characteristics of each LINK statement using BSDROUTINGPARMS

To define the link characteristics, such as MTU (1) and Subnet Mask (2), we used the BSDROUTINGPARMS statement. If not supplied, defaults will be supplied from static routing definitions in BEGINROUTES, OMPROUTE configuration (if OMPROUTE is running), or if none supplied, the stack's interface layer based on hardware capabilities and characteristics of devices and links. We defined our links with the highest MTU size, as shown in Example 4-6.

Example 4-6 BSDRoutingparms statements

```
BSDROUTINGPARMS TRUE
; Link name      MTU    Cost metric Subnet Mask    Dest address
STAVIPA1LNK     1492 1       0       255.255.255.252   0
 OSA2080LNK     1492         0       255.255.255.0 2   0
 OSA20A0LNK     1492         0       255.255.255.0     0
 OSA20C0LNK     1492         0       255.255.255.0     0
 OSA20E0LNK     1492         0       255.255.255.0     0
IUTIQDF4LNK    57344         0       255.255.255.0     0
IUTIQDF5LNK    57344         0       255.255.255.0     0
IUTIQDF6LNK    57344         0       255.255.255.0     0
; ENDBSDROUTINGPARMS
```

4.4.2 HiperSockets implementation

The steps to implement HiperSockets are basically the same as we did to implement the OSA-Express devices. What changes is that there is no external configurations to be done, and the TRLE is created dynamically by VTAM.

When defining an MPCIPA HiperSockets, use the DEVICE statement to specify the IQD CHPID hexadecimal value. The reserved device name prefix IUTIQD*xx* must be specified. The suffix *xx* indicates the hexadecimal value of the corresponding IQD CHPID that was configured with HCD or IOCP. Figure 4-9 on page 118 shows the HiperSockets implementation scenario.

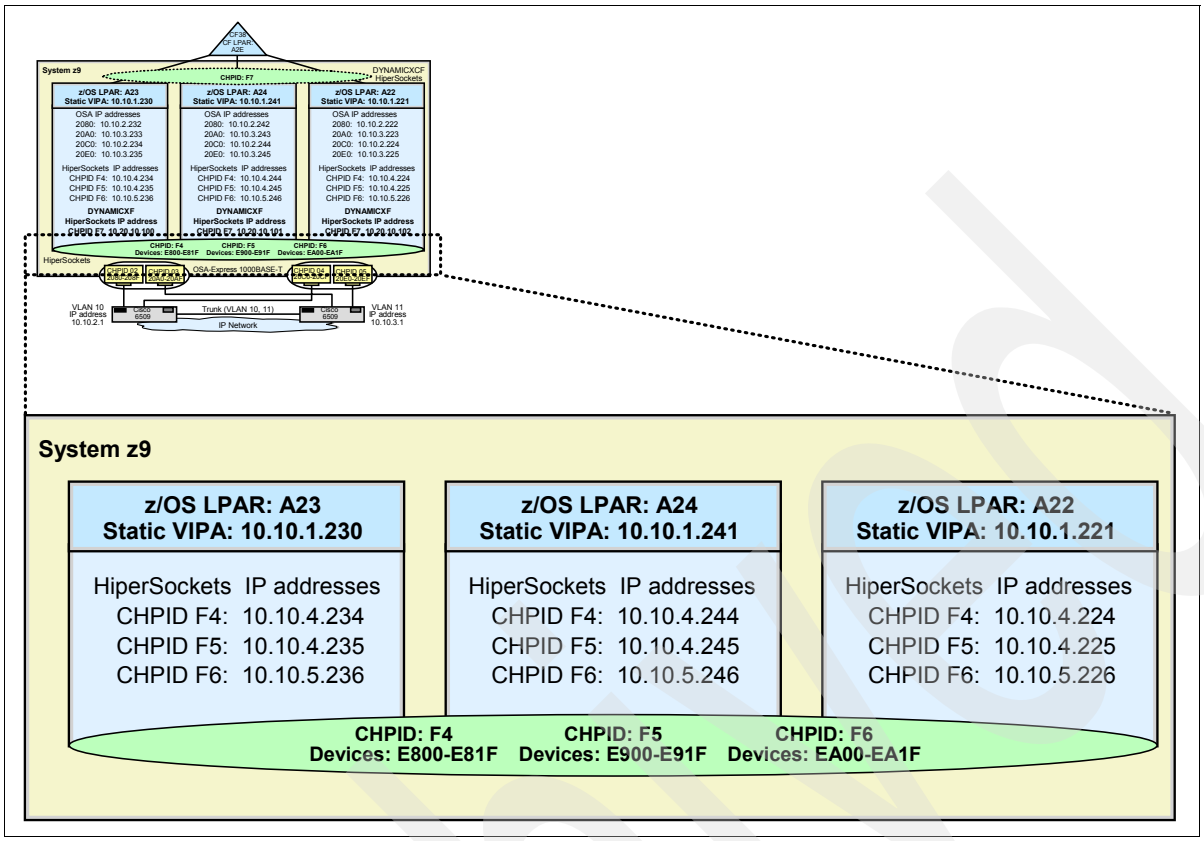


Figure 4-13 HiperSockets implementation scenario

Create a DEVICE and LINK statements for each HiperSockets CHPID

Define the device and link statements for each HiperSockets CHPID being implemented, as shown in Example 4-7. A HiperSockets CHPID must be defined as an MPCIPA type of device **1**.

The link definition describes the type of transport being used. A HiperSockets link is defined as IPAQIDIO **2**.

Example 4-7 HiperSockets device and link definitions

```
;HiperSockets definition. The TRLE is dynamically created on VTAMs
DEVICE IUTIQDF4 MPCIPA    1
LINK   IUTIQDF4LNK  IPAQIDIO  2    IUTIQDF4
DEVICE IUTIQDF5 MPCIPA    1
LINK   IUTIQDF5LNK  IPAQIDIO  2   IUTIQDF5
DEVICE IUTIQDF6 MPCIPA    1
LINK   IUTIQDF6LNK  IPAQIDIO  2   IUTIQDF6
```

Important: The hexadecimal value specified here represents the CHPID and cannot be the same value of that used for the dynamic XCF HiperSockets interface.

Create a HOME address to each defined LINK

Each link configured must have its own IP address. Our HiperSockets links are defined with the IP addresses, as shown in Example 4-8 on page 127.

Example 4-8 HiperSockets HOME addresses

```
HOME
   10.10.4.234   IUTIQDF4LNK
   10.10.4.235   IUTIQDF5LNK
   10.10.5.236   IUTIQDF6LNK
```

Define the characteristics of each LINK statement using BSDROUTINGPARMS

To define the link characteristics such as MTU (1) and Subnet Mask (2), we use the BSDROUTINGPARMS statement. If not supplied, defaults will be supplied from static routing definitions in the BEGINROUTES, OMPROUTE configuration (if OMPROUTE is running), or if none supplied, the stack's interface layer based on hardware capabilities and characteristics of devices and links. We defined our links with the highest MTU size, as shown in Example 4-9.

Example 4-9 BSDRoutingparms statements

```
BSDROUTINGPARMS TRUE
;  Link name      MTU    Cost metric  Subnet Mask    Dest address
STAVIPA1LNK     1492 1       0       255.255.255.252   0
 OSA2080LNK     1492         0       255.255.255.0 2   0
 OSA20A0LNK     1492         0       255.255.255.0     0
 OSA20C0LNK     1492         0       255.255.255.0     0
 OSA20E0LNK     1492         0       255.255.255.0     0
IUTIQDF4LNK    57344         0       255.255.255.0     0
IUTIQDF5LNK    57344         0       255.255.255.0     0
IUTIQDF6LNK    57344         0       255.255.255.0     0
; ENDBSDROUTINGPARMS
```

4.4.3 DYNAMICXCF implementation

To implement XCF connections in our test environment, we can use three different types of devices:

- DynamicXCF HiperSockets device (IUTIQDIO) for connections between z/OS LPARs within the same server
- DynamicXCF SAMEHOST device (IUTSAMEH) for stacks within the same LPAR
- VTAM dynamically created ISTLSXCF to connect z/OS LPARs in other servers within the same Sysplex.

Figure 4-14 on page 128 shows our DynamicXCF implementation, using HiperSockets CHPID F7.

When using dynamic XCF for Sysplex configuration, make sure that XCFINIT=YES or XCFINIT=DEFINE is coded in the VTAM start options, or if XCFINIT=NO was specified, issue the VARY ACTIVATE command for the ISTLSXCF major node. This ensures that XCF connections between TCP stacks on different VTAM nodes in the sysplex can be established.

The VTAM ISTLSXCF major node must be active for DYNAMICXCF work, except for the following scenarios:

- Multiple TCP/IP stacks on the same LPAR; a dynamic SAMEHOST definition is generated whether ISTLSXCF is active or not.
- HiperSockets is configured and enabled across multiple z/OS LPARs that are in the same sysplex and the same server. If this is the case, a dynamic IUTIQDIO link is created whether ISTLSXCF is active or not.

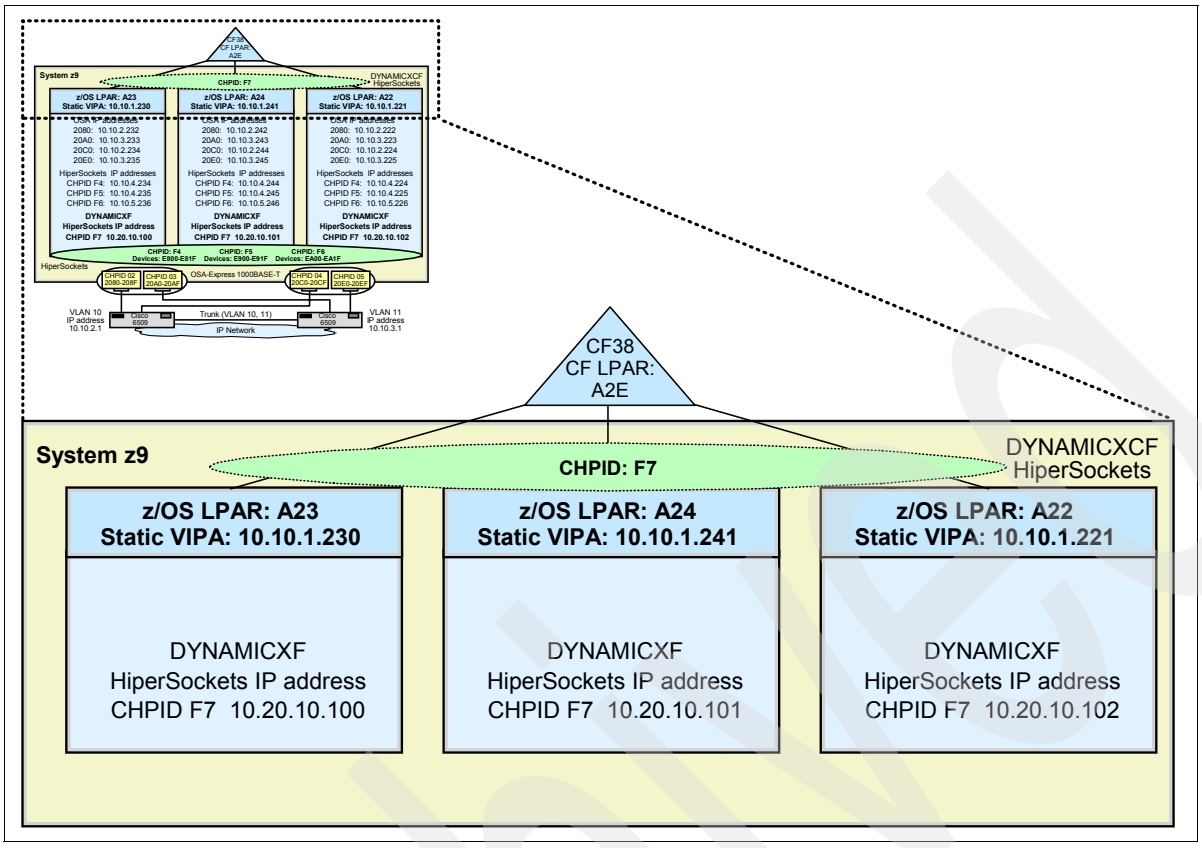


Figure 4-14 DynamicXCF with HiperSockets

To implement DYNAMICXCF in our test environment, we coded the IPCONFIG definitions in the TCP/IP profile, as shown in Example 4-10. To control the IP subnetwork used to connect all z/OS images, we define the XCF IP address, the IP mask, and the link cost in the DYNAMICXCF statement **1**.

Example 4-10 IPCONFIG DYNAMICXCF configuration

```
IPCONFIG DATAGRAMFWD SYSPLEXROUTING IPSECURITY
DYNAMICXCF 10.20.10.100 255.255.255.0 8 1
```

4.4.4 Controlling the device definitions

After we implemented all required connectivity definitions in the TCP/IP profile we can start, stop, or modify devices.

Starting a device

A device can be started by:

- Defining the START statement in the TCP/IP profile, as shown in Example 4-11.

Example 4-11 Start statements in TCP/IP profile

```
START IUTIQDF4
START IUTIQDF5
START IUTIQDF6
START OSA2080
START OSA20a0
START OSA20c0
```

```
START OSA20e0
```

- Using the z/OS console command **VARY TCPIP,tcpipproc,start,devicename**.
- Creating a file with a start statement and using the z/OS console command **Vary TCPIP,tcpipproc,OBEYFILE,datasetname**. The file defined by the file name has the START statement to activate the desired device or devices.

Using any of the starting methods will result in a series of messages, as shown in Example 4-12.

Example 4-12 Starting a TCP/IP device

```
V TCPIP,TCPIPA,START,OSA20C0
EZZ0060I PROCESSING COMMAND: VARY TCPIP,TCPIPA,START,OSA20C0
EZZ0053I COMMAND VARY START COMPLETED SUCCESSFULLY
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE OSA20C0
```

The DYNAMICXCF statement dynamically generates the DEVICE, LINK, and HOME statements. It also starts the device when the TCP/IP stack is activated, as we can see in the messages shown in Example 4-13.

Example 4-13 DYNAMICXCF messages

```
$HASP373 TCPIPA   STARTED

EZZ0350I SYSPLEX ROUTING SUPPORT IS ENABLED
EZZ0624I DYNAMIC XCF DEFINITIONS ARE ENABLED
EZD1176I TCPIPA HAS SUCCESSFULLY JOINED THE TCP/IP SYSPLEX GROUP
EZZ4324I CONNECTION TO 10.20.10.101 ACTIVE FOR DEVICE IUTSAMEH   a
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE IUTIQDIO b
```

a This message indicates that the TCPIPA stack has been connected to the other stacks through XCF using a SAMEHOST device.

b This message indicates that XCF will also use HiperSockets to connect other TCP/IP stacks within the same server, using a IUTIQDIO device.

Stopping a device

A device can be stopped by:

- Using the z/OS console command **Vary TCPIP,*tcpipproc*,STOP,*devicename***.
- Creating a file with the stop statement to the desired device or devices and using the z/OS console command **Vary TCPIP,*tcpipproc*,OBEYFILE,*datasetname***.

When you stop a device, you will see messages as shown in Example 4-14.

Example 4-14 Stop command resulting messages

```
V TCPIP,TCPIPA,STOP,OSA20C0
EZZ0060I PROCESSING COMMAND: VARY TCPIP,TCPIPA,STOP,OSA20C0
EZZ0053I COMMAND VARY STOP COMPLETED SUCCESSFULLY
EZZ4329I LINK OSA2080LNK HAS TAKEN OVER ARP RESPONSIBILITY FOR
INACTIVE LINK OSA20C0LNK
EZZ4315I DEACTIVATION COMPLETE FOR DEVICE OSA20C0
```

Note: When an OSA-Express device is stopped or loses its connection to the switch, another OSA-Express device defined to the TCP/IP stack will take over the IP address. A gratuitous ARP is broadcasted on the LAN to advertise the new MAC address related to the IP address being taken over. Message EZZ4329I in Example 4-14 indicates this action.

Modifying characteristics of a device

You can temporally modify the characteristics of a device by issuing the OBEY command:

```
Vary TCPIP,tcpipproc,OBEYFILE,datasetname
```

Authorization to use this command is through the user's RACF profile. The datasetname cannot be a z/OS UNIX file system file. The data set contains the modified TCP/IP configuration statements (see Example 4-15).

Example 4-15 OBEYFILE example

```
;Original BSDROUTINGPARMS statement for link OSA20C0

;BSDROUTINGPARMS TRUE
;  Link name      MTU    Cost metric  Subnet Mask    Dest address
;OSA20C0LNK     1492        0       255.255.255.0     0
;ENDBSDROUTINGPARMS

;Modified BSDROUTINGPARMS statement for link OSA20C0

BSDROUTINGPARMS TRUE
;  Link name      MTU    Cost metric  Subnet Mask    Dest address
OSA20C0LNK     1024        0       255.255.255.0     0
ENDBSDROUTINGPARMS
```

Important: Dynamic XCF cannot be changed by using the OBEYFILE command. If you want to change the IPCONFIG DYNAMICXCF parameters, stop TCP/IP, code a new IPCONFIG DYNAMICXCF statement in the initial profile, and restart TCP/IP.

4.4.5 Verifying the connectivity status

In this section, we verify the status of all devices defined to the TCP/IP stack or VTAM.

Verifying the device status in TCP/IP stack

To verify the status of all devices being activated in the TCP/IP stack we use the NETSTAT command with the DEVLIST option, as seen in Example 4-16.

Example 4-16 Using command D TCPIP,TCPIPA,N,DEV to verify the Device status

```
D TCPIP,TCPIPA,N,DEV
 EZD0101I NETSTAT CS V1R7 TCPIPA 810
DEVNAME: OSA2080           DEVTYPE: MPCIPA
   DEVSTATUS: READY
   LNKNAME: OSA2080LNK        LNKTYPE: IPAQENET   LNKSTATUS: READY
     NETNUM: N/A  QUESIZE: N/A  SPEED: 0000000100
     IPBROADCASTCAPABILITY: NO
     CFGROUTER: NON                    ACTROUTER: NON
     ARPOFFLOAD: YES                   ARPOFFLOADINFO: YES
     ACTMTU: 1492
     VLANID: 10                        VLANPRIORITY: DISABLED
     DYNVLANREGCFG: NO                 DYNVLANREGCAP: YES
     READSTORAGE: GLOBAL (4096K)       INBPERF: BALANCED
```

```
    CHECKSUMOFFLOAD: YES             SEGMENTATIONOFFLOAD: YES
    SECCLASS: 255
  BSD ROUTING PARAMETERS:
    MTU SIZE: 1492              METRIC: 10
    DESTADDR: 0.0.0.0           SUBNETMASK: 255.255.255.0
  MULTICAST SPECIFIC:
    MULTICAST CAPABILITY: YES
    GROUP             REFCNT
    -----             ------
    224.0.0.1         0000000001
   LINK STATISTICS:
     BYTESIN                          = 506807
     INBOUND PACKETS                  = 2803
     INBOUND PACKETS IN ERROR         = 5258
     INBOUND PACKETS DISCARDED        = 0
     INBOUND PACKETS WITH NO PROTOCOL = 0
     BYTESOUT                         = 429
     OUTBOUND PACKETS                 = 3
     OUTBOUND PACKETS IN ERROR        = 0
     OUTBOUND PACKETS DISCARDED       = 0
```

Displaying TCP/IP device resources in VTAM

The device drivers for TCP/IP are provided by VTAM. When CS for z/OS IP devices are activated, there must be an equivalent Transport Resource List Element (TRLE) defined to VTAM. The devices that are exclusively used by z/OS Communications Server IP have TRLEs that are automatically generated for them.

Since the device driver resources are provided by VTAM, you have the ability to display the resources using VTAM display commands.

For dynamically generated TRLEs, the device type and address can be decoded from the generated TRLE name. The format is *IUTtaaaa*:

- *IUT* - Fixed for all dynamically generated TRLEs.
- *t* - Shows the device type, which indicates the following:
  - C - Indicates this is a CDLC device.
  - H - Indicates this is a HYPERCHANNEL device.
  - I - Indicates this a QDIO device.
  - L - Indicates this is a LCS device.
  - S - Indicates this is a SAMEHOST device.
  - W - Indicates this is a CLAW device.
  - X - Indicates this is a CTC device.
- *aaaa* - The read device number. For SAMEHOST connections, this is a sequence number.

For XCF links, the format of the TRLE name is ISTT*xxyy*. ISTT is fixed, *xx* is the SYSCLONE value of the originating VTAM, and *yy* is the SYSCLONE value of the destination VTAM.

To display a list of all TRLEs active in VTAM, use the command D NET,TRL, as shown in Example 4-17.

Example 4-17 D NET,TRL command output

```
D NET,TRL
IST097I DISPLAY ACCEPTED
IST350I DISPLAY TYPE = TRL 137
IST924I -------------------------------------------------------------
```

```
IST1954I TRL MAJOR NODE = ISTTRL
IST1314I TRLE = ISTT3130  STATUS = ACTIV       CONTROL = XCF
IST1314I TRLE = IUTIQDF6  STATUS = ACTIV       CONTROL = MPC
IST1314I TRLE = IUTIQDF5  STATUS = ACTIV       CONTROL = MPC
IST1314I TRLE = IUTIQDF4  STATUS = ACTIV       CONTROL = MPC
IST1314I TRLE = ISTT3132  STATUS = ACTIV       CONTROL = XCF
IST1314I TRLE = IUTIQDIO  STATUS = ACTIV       CONTROL = MPC
IST1314I TRLE = IUTSAMEH  STATUS = ACTIV       CONTROL = MPC
IST1454I 7 TRLE(S) DISPLAYED
IST924I -------------------------------------------------------------
IST1954I TRL MAJOR NODE = OSA2040
IST1314I TRLE = OSA2040P  STATUS = ACTIV       CONTROL = MPC
IST1454I 1 TRLE(S) DISPLAYED
IST924I -------------------------------------------------------------
IST1954I TRL MAJOR NODE = OSA2080
IST1314I TRLE = OSA2080P  STATUS = ACTIV       CONTROL = MPC
IST1454I 1 TRLE(S) DISPLAYED
IST924I -------------------------------------------------------------
IST1954I TRL MAJOR NODE = OSA20A0
IST1314I TRLE = OSA20A0P  STATUS = ACTIV       CONTROL = MPC
IST1454I 1 TRLE(S) DISPLAYED
IST924I -------------------------------------------------------------
IST1954I TRL MAJOR NODE = OSA20C0
IST1314I TRLE = OSA20C0P  STATUS = ACTIV       CONTROL = MPC
IST1454I 1 TRLE(S) DISPLAYED
IST924I -------------------------------------------------------------
IST1954I TRL MAJOR NODE = OSA20E0
IST1314I TRLE = OSA20E0P  STATUS = ACTIV       CONTROL = MPC
IST1454I 1 TRLE(S) DISPLAYED
IST314I END
```

You can also display information of TRLEs grouped by control type, such as MPC or XCF devices, as shown in Example 4-18.

Example 4-18 D NET,TRL,CONTROL=MPC

```
D NET,TRL,CONTROL=MPC
IST097I DISPLAY ACCEPTED
IST350I DISPLAY TYPE = TRL 152
IST924I -------------------------------------------------------------
IST1954I TRL MAJOR NODE = ISTTRL
IST1314I TRLE = IUTIQDF6  STATUS = ACTIV       CONTROL = MPC
IST1314I TRLE = IUTIQDF5  STATUS = ACTIV       CONTROL = MPC
IST1314I TRLE = IUTIQDF4  STATUS = ACTIV       CONTROL = MPC
IST1314I TRLE = IUTIQDIO  STATUS = ACTIV       CONTROL = MPC
IST1314I TRLE = IUTSAMEH  STATUS = ACTIV       CONTROL = MPC
IST1454I 5 TRLE(S) DISPLAYED
IST924I -------------------------------------------------------------
IST1954I TRL MAJOR NODE = TRL31
IST1314I TRLE = MPC0410   STATUS = ACTIV       CONTROL = MPC
IST1454I 1 TRLE(S) DISPLAYED
IST924I -------------------------------------------------------------
IST1954I TRL MAJOR NODE = OSA2040
IST1314I TRLE = OSA2040P  STATUS = ACTIV       CONTROL = MPC
IST1454I 1 TRLE(S) DISPLAYED
IST924I -------------------------------------------------------------
IST1954I TRL MAJOR NODE = OSA2080
IST1314I TRLE = OSA2080P  STATUS = ACTIV       CONTROL = MPC
IST1454I 1 TRLE(S) DISPLAYED
IST924I -------------------------------------------------------------
```

```
IST1954I TRL MAJOR NODE = OSA20A0
IST1314I TRLE = OSA20A0P  STATUS = ACTIV       CONTROL = MPC
IST1454I 1 TRLE(S) DISPLAYED
IST924I -------------------------------------------------------------
IST1954I TRL MAJOR NODE = OSA20C0
IST1314I TRLE = OSA20C0P  STATUS = ACTIV       CONTROL = MPC
IST1454I 1 TRLE(S) DISPLAYED
IST924I -------------------------------------------------------------
IST1954I TRL MAJOR NODE = OSA20E0
IST1314I TRLE = OSA20E0P  STATUS = ACTIV       CONTROL = MPC
IST1454I 1 TRLE(S) DISPLAYED
IST314I END
```

We can also get specific information about a single TRLE, using the TRLE name as shown in Example 4-19, for an OSA-Express device.

Example 4-19 D NET,TRL,TRLE=OSA2080P

```
DISPLAY ACCEPTED
NAME = OSA2080P, TYPE = TRLE 164
 TRL MAJOR NODE = OSA2080
STATUS= ACTIV, DESIRED STATE= ACTIV
TYPE = LEASED             , CONTROL = MPC , HPDT = YES
 MPCLEVEL = QDIO       MPCUSAGE = SHARE
 PORTNAME = OSA2080    LINKNUM =   0   OSA CODE LEVEL = 0805
 HEADER SIZE = 4096 DATA SIZE = 0 STORAGE = ***NA***
 WRITE DEV = 2081 STATUS = ACTIVE     STATE = ONLINE
 HEADER SIZE = 4092 DATA SIZE = 0 STORAGE = ***NA***
 READ  DEV = 2080 STATUS = ACTIVE     STATE = ONLINE
 DATA  DEV = 2082 STATUS = ACTIVE     STATE = N/A
 I/O TRACE = OFF  TRACE LENGTH = *NA*
 ULPID = TCPIPA
 IQDIO ROUTING DISABLED
 READ STORAGE = 4.0M(64 SBALS)
 PRIORITY1: UNCONGESTED PRIORITY2: UNCONGESTED
 PRIORITY3: UNCONGESTED PRIORITY4: UNCONGESTED
 UNITS OF WORK FOR NCB AT ADDRESS X'10807010'
 P1 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
IST1802I P2 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
IST1802I P3 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
IST1802I P4 CURRENT = 0 AVERAGE = 1 MAXIMUM = 2
IST1221I DATA  DEV = 2083 STATUS = ACTIVE     STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
```

The same command to get information about a HiperSockets device is shown in Example 4-20.

Example 4-20 D NET,TRL,TRLE=IUTIQDF6

```
D NET,TRL,TRLE=IUTIQDF6
IST097I DISPLAY ACCEPTED
IST075I NAME = IUTIQDF6, TYPE = TRLE 168
IST1954I TRL MAJOR NODE = ISTTRL
IST486I STATUS= ACTIV, DESIRED STATE= ACTIV
IST087I TYPE = LEASED             , CONTROL = MPC , HPDT = YES
IST1715I MPCLEVEL = QDIO       MPCUSAGE = SHARE
IST1716I PORTNAME =            LINKNUM =   0   OSA CODE LEVEL = ...(
IST1577I HEADER SIZE = 4096 DATA SIZE = 16384 STORAGE = ***NA***
IST1221I WRITE DEV = EA01 STATUS = ACTIVE     STATE = ONLINE
IST1577I HEADER SIZE = 4092 DATA SIZE = 0 STORAGE = ***NA***
```

```
IST1221I READ  DEV = EA00 STATUS = ACTIVE     STATE = ONLINE
IST1221I DATA  DEV = EA02 STATUS = ACTIVE     STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1717I ULPID = TCPIPA
IST1815I IQDIO ROUTING DISABLED
IST1918I READ STORAGE = 2.0M(126 SBALS)
IST1757I PRIORITY1: UNCONGESTED PRIORITY2: UNCONGESTED
IST1757I PRIORITY3: UNCONGESTED PRIORITY4: UNCONGESTED
IST1801I UNITS OF WORK FOR NCB AT ADDRESS X'0E844010'
IST1802I P1 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
IST1802I P2 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
IST1802I P3 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
IST1802I P4 CURRENT = 0 AVERAGE = 1 MAXIMUM = 2
IST1221I DATA  DEV = EA03 STATUS = ACTIVE     STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
```

You can also display XCF TRLE specific information, as shown in Example 4-21.

Example 4-21 D NET,TRL,TRLE=ISTT3031

```
D NET,TRL,TRLE=ISTT3031
IST097I DISPLAY ACCEPTED
IST075I NAME = ISTT3031, TYPE = TRLE 435
IST1954I TRL MAJOR NODE = ISTTRL
IST486I STATUS= ACTIV, DESIRED STATE= ACTIV
IST087I TYPE = LEASED            , CONTROL = XCF , HPDT = *NA*
IST1715I MPCLEVEL = HPDT       MPCUSAGE = SHARE
IST1717I ULPID = ISTP3031
IST1503I XCF TOKEN = 02000000E00160002     STATUS = ACTIVE
IST1502I ADJACENT CP = USIBMSC.SC31M
IST314I END
```

The DYNAMICXCF configuration created a HiperSockets TRLE named IUTIQDIO. The related TRLE status can also be displayed, as shown in Example 4-22.

Example 4-22 D NET,TRL,TRLE=IUTIQDIO

```
D NET,TRL,TRLE=IUTIQDIO
IST097I DISPLAY ACCEPTED
IST075I NAME = IUTIQDIO, TYPE = TRLE 472
IST1954I TRL MAJOR NODE = ISTTRL
IST486I STATUS= ACTIV, DESIRED STATE= ACTIV
IST087I TYPE = LEASED            , CONTROL = MPC , HPDT = YES
IST1715I MPCLEVEL = QDIO       MPCUSAGE = SHARE
IST1716I PORTNAME = IUTIQDF7   LINKNUM =   0   OSA CODE LEVEL = ...(
IST1577I HEADER SIZE = 4096 DATA SIZE = 16384 STORAGE = ***NA***
IST1221I WRITE DEV = EB01 STATUS = ACTIVE     STATE = ONLINE
IST1577I HEADER SIZE = 4092 DATA SIZE = 0 STORAGE = ***NA***
IST1221I READ  DEV = EB00 STATUS = ACTIVE     STATE = ONLINE
IST1221I DATA  DEV = EB02 STATUS = ACTIVE     STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1717I ULPID = TCPIPA
IST1815I IQDIO ROUTING DISABLED
IST1918I READ STORAGE = 2.0M(126 SBALS)
IST1757I PRIORITY1: UNCONGESTED PRIORITY2: UNCONGESTED
IST1757I PRIORITY3: UNCONGESTED PRIORITY4: UNCONGESTED
IST1801I UNITS OF WORK FOR NCB AT ADDRESS X'102FB010'
IST1802I P1 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
IST1802I P2 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
IST1802I P3 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
```

```
IST1802I P4 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
IST1221I DATA  DEV = EB03 STATUS = ACTIVE     STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
```

The DYNAMICXCF configuration created a SAMEHOST TRLE named IUTSAMEH. The related TRLE status can be displayed, as shown in Example 4-23.

Example 4-23 D NET,TRL,TRLE=IUTSAMEH

```
D NET,TRL,TRLE=IUTSAMEH
IST097I DISPLAY ACCEPTED
IST075I NAME = IUTSAMEH, TYPE = TRLE 491
IST1954I TRL MAJOR NODE = ISTTRL
IST486I STATUS= ACTIV, DESIRED STATE= ACTIV
IST087I TYPE = LEASED             , CONTROL = MPC , HPDT = YES
IST1715I MPCLEVEL = HPDT       MPCUSAGE = SHARE
IST1717I ULPID = TCPIPA
IST314I END
```

4.4.6 Problem determination

Isolating network problems is an essential step to verify a connectivity problem in your environment. This section introduces commands and techniques that can use to diagnose network connectivity problems related to a specific interface.

The following diagnostic commands are available for either the z/OS UNIX environment or the TSO environment:

- PING - The PING command can be very useful to determine if a destination address can be reached in the network. Based on the results, it is possible to define whether the problem is related to the interface being tested, or whether it is a network-related problem.

Using PING we can verify the following:

- The directly attached network is defined correctly.
- The device is properly connected to the network.
- The device is able to send and receive packets on the network.
- The remote host is able to receive and send packets.

When a PING command is issued, you can receive any of the responses listed in Table 4-4.

Table 4-4 Using PING command as a debugging tool

PING command (direct network)	PING response	Possible cause and actions
ping 10.10.2.1 (intf osa2080lnk	CS V1R7: Pinging host 10.10.2.1 sendMessage(): EDC8130I Host cannot be reached.	The interface being tested has a problem. Use the **Netstat** command to verify the interface status.
ping 10.10.2.1 (intf osa2080lnk	CS V1R7: Pinging host 10.10.2.1 Ping #1 timed out	It means the ICMP packet has been sent to the network, but the destination address is either invalid or it is not able to answer. Correct the destination address or verify the destination host status. This problem should be verified in the network.

PING command (direct network)	PING response	Possible cause and actions
ping 10.10.2.1 (intf osa2080lnk	CS V1R7: Pinging host 10.10.2.1 Ping #1 response took 0.000 seconds.	This is the expected response. The interface is working.

- **Netstat** - We can use the **Netstat** command to verify our TCP/IP configuration. The information provided in the output from the **Netstat** command should be checked against the values in our configuration data sets for the TCP/IP stack. To verify our connectivity status from an interface perspective, we can use the following Netstat options:
  - Netstat HOME/-h shows all defined interfaces and their IP addresses, even those interfaces dynamically created, as shown in Example 4-24.

Example 4-24 NETSTAT HOME command results

```
D TCPIP,TCPIPA,N,HOME
EZD0101I NETSTAT CS V1R7 TCPIPA 109
HOME ADDRESS LIST:
LINKNAME:   STAVIPA1LNK
  ADDRESS:  10.10.1.230
    FLAGS:  PRIMARY
LINKNAME:   OSA2080LNK
  ADDRESS:  10.10.2.232
    FLAGS:
LINKNAME:   IUTIQDF4LNK
  ADDRESS:  10.10.4.234
    FLAGS:
LINKNAME:   EZASAMEMVS
  ADDRESS:  10.20.10.100
    FLAGS:
LINKNAME:   IQDIOLNK0A140A64
  ADDRESS:  10.20.10.100
    FLAGS:
LINKNAME:   LOOPBACK
  ADDRESS:  127.0.0.1
    FLAGS:
INTFNAME:   LOOPBACK6
  ADDRESS:  ::1
    TYPE:   LOOPBACK
    FLAGS:
12 OF 12 RECORDS DISPLAYED
```

  - Netstat DEVLINKS/-d shows the status of each interface, physical and logical, defined in the TCP/IP stack, as shown in Example 4-25 (only one interface shown as an example).

Example 4-25 NETSTAT DEVLINKS command results

```
DISPLAY TCPIP,TCPIPA,N,DEV,INTFN=OSA2080LNK
EZD0101I NETSTAT CS V1R7 TCPIPA 180
DEVNAME: OSA2080           DEVTYPE: MPCIPA
  DEVSTATUS: NOT ACTIVE
  LNKNAME: OSA2080LNK        LNKTYPE: IPAQENET   LNKSTATUS: NOT ACTIVE
    NETNUM: N/A  QUESIZE: N/A
    IPBROADCASTCAPABILITY: NO
    CFGROUTER: NON                   ACTROUTER: UNKNOWN
    ACTMTU: UNKNOWN
    SECCLASS: 255
  BSD ROUTING PARAMETERS:
```

```
   MTU SIZE: 1492              METRIC: 00
   DESTADDR: 0.0.0.0           SUBNETMASK: 255.255.255.0
 MULTICAST SPECIFIC:
    MULTICAST CAPABILITY: UNKNOWN
    GROUP             REFCNT
    -----             ------
    224.0.0.1         0000000001
  LINK STATISTICS:
    BYTESIN                          = 10892725
    INBOUND PACKETS                  = 61972
    INBOUND PACKETS IN ERROR         = 117576
    INBOUND PACKETS DISCARDED        = 0
    INBOUND PACKETS WITH NO PROTOCOL  = 0
    BYTESOUT                         = 1368
    OUTBOUND PACKETS                 = 8
    OUTBOUND PACKETS IN ERROR        = 4
    OUTBOUND PACKETS DISCARDED       = 0
1 OF 1 RECORDS DISPLAYED
END OF THE REPORT
```

– Netstat ARP/-R (for OSA-Express devices) - Use this command to query the ARP cache for a given address. Use this command when the remote host does not answer as expected, to check if an ARP entry has been created for the remote host. It also allows you to check if the relationship between the IP and MAC address is the expected one. The resulting display is shown in Example 4-26.

Example 4-26 D TCPIP,TCPIPA,N,ARP command results

```
DISPLAY TCPIP,TCPIPA,N,ARP
EZD0101I NETSTAT CS V1R7 TCPIPA 266
QUERYING ARP CACHE FOR ADDRESS 10.10.2.224
LINK: OSA2080LNK        ETHERNET: 00096B1A7490
QUERYING ARP CACHE FOR ADDRESS 10.10.2.222
LINK: OSA2080LNK        ETHERNET: 00096B1A7490
QUERYING ARP CACHE FOR ADDRESS 10.10.2.244
LINK: OSA2080LNK        ETHERNET: 00096B1A7490
QUERYING ARP CACHE FOR ADDRESS 10.10.2.242
LINK: OSA2080LNK        ETHERNET: 00096B1A7490
QUERYING ARP CACHE FOR ADDRESS 10.10.2.232
LINK: OSA2080LNK        ETHERNET: 00096B1A7490
QUERYING ARP CACHE FOR ADDRESS 10.10.2.1
LINK: OSA2080LNK        ETHERNET: 00097B61E48A
QUERYING ARP CACHE FOR ADDRESS 10.10.3.245
LINK: OSA20A0LNK        ETHERNET: 00096B1A74C2
```

The described commands can help you locate connectivity problems. If they do not, the next step to debug a direct attached network problem would be to gather documentation that would show more detailed information about traffic problems related to the interface and network.

To get this detailed information, the z/OS V1R7.0 Communications Server typically uses the component trace to capture event data and save it to an internal buffer, or writes the internal buffer to an external writer, if requested. You can later format these trace records using the Interactive Problem Control System (IPCS) subcommand CTRACE.

To debug a network connectivity problem you can use the Component trace with either of the two specific components, as follows:

- SYSTCPIP component trace with options:
  - VTAM, which shows all of the nondata-path signaling occurring between the devices and VTAM
  - VTAMDATA, which shows data-path signaling between the devices and VTAM, including a snapshot of media headers and some data

  Attention: Using this option slows performance considerably; therefore, it should be used with caution.

- SYSTCPDA component trace, used with the VARY TCPIP,PKTTRACE command. You can use the PKTTRACE statement to copy IP packets as they enter or leave TCP/IP, and then examine the contents of the copied packets.

For more information about how to set up and activate a CTRACE, refer to Appendix A, “Component trace (CTRACE)” on page 209.

5

Routing

One of the major functions of a network protocol such as TCP/IP is to efficiently interconnect a number of disparate networks. These networks may include LANs and WANs, fast and slow, reliable and unreliable, inexpensive and expensive connections. To interconnect these networks, some level of intelligence is needed at the boundaries to look at the data packets as they pass and make rational decisions as to where and how they should be forwarded. This is known as IP routing. In this chapter we look at the various types of IP routing supported in a z/OS environment.

This chapter includes the following.

Section	Topic
5.1, “Overview” on page 140	Discusses the basic concepts of IP routing
5.2, “Why IP routing is important” on page 154	Discusses key characteristics of IP routing and why it may be important in your environment
5.4, “The common design scenarios for IP routing” on page 155	Presents commonly implemented IP routing design scenarios, their dependencies, advantages, considerations, and our recommendations
5.5, “How IP routing is implemented” on page 160	Presents selected implementation scenarios, tasks, configuration examples, and problem determination suggestions

5.1 Overview

When we talk about networks, one of the key issues is how to transport data across the network. Based on the OSI reference model, the act of moving data traffic across a network from a source to a destination can be accomplished either by bridging or routing this data between the endpoints. Routing is often compared with bridging, which might seem to accomplish precisely the same thing. However, the primary difference between the two is that bridging occurs at Layer 2 (the data link control layer) of the OSI reference model, while routing occurs at Layer 3 (the network layer). This distinction provides routing and bridging with different information to use in the process of moving information from source to destination, so the two functions accomplish their tasks in different ways.

5.1.1 Basic concepts

To help understand the concepts described in this section, Table 5-1 lists some of the common IP routing-related terms. Most of the functions or protocols listed are supported by the z/OS Communications Server.

Table 5-1 IP routing terms

Term	Definition
Routing	The process used in an IP network to deliver a datagram to the correct destination.
Static routing	Routing that is manually configured and does not change automatically in response to network topology changes.
Replaceable *static routes*	Static routes that can be replaced by OMPROUTE.
Dynamic routing	Routing that is dynamically managed by a routing daemon and automatically changes in response to network topology changes.
Routing daemon	A server process that manages the IP routing table.
Autonomous system (AS)	A group of routers exchanging routing information through a common routing protocol. A single AS may represent a large number of IP networks.
Router	A device or host that interprets protocols at the Internet Protocol (IP) layer and forwards datagrams on a path towards their correct destination.
Gateway	A router that is placed between networks or subnetworks. The term is used to represent routers between autonomous systems.
Interior gateway protocols (IGP)	Dynamic route update protocols that are used between routers and hosts inside your own network and between your network and a service provider.
Exterior gateway protocols (EGP)	Dynamic route update protocols that are used between routers that are placed between two or more Autonomous Systems.

In order to route packets on the network, each network interface must have a unique IP address assigned. Whenever a packet is sent, the destination and source IP addresses are included in the packet's header information. The network layer (Layer 3) of the TCP/IP stack examines the destination IP address to determine how the packet should be forwarded. The

packet is either sent to its destination on the same network (direct routing) or based on a routing table entry, to another network via a router (indirect routing).

Types of IP routing

As mentioned, there are two types of IP routing:

Direct routing If the destination host is attached to the same physical network as the source host, IP datagrams can be directly exchanged. This is done by encapsulating the IP datagram in the physical network frame. This is called direct delivery and is referred to as direct routing.

Indirect routing Occurs when the destination host is not connected to a network directly attached to the source host. The only way to reach the destination is through one or more IP routers. The address of the first router (the first hop) is called an indirect route in the IP routing algorithm. The IP address of the first router is the only information needed by the source host to send a packet to the destination host.

If the source and destination hosts are on the same physical network, but defined in different subnetworks, indirect routing is used to communicate between the endpoints. A router is needed to forward packets between subnetworks.

IP routing table

Every IP host is capable of routing IP datagrams and maintaining an IP routing table. There are three types of entries in an IP routing table:

- Direct routes describing the networks to which the host is attached.
- Indirect routes describing networks reachable through one or more IP routers.
- The default route, which contains the route to be used when the destination IP (sub)network is not found in any of the direct or indirect routes.

Each entry contains a destination IP (sub)network address and the IP address of the next router (next hop) to which the packet should be sent.

As an example, let us look at Figure 5-1, which represents a simple network.

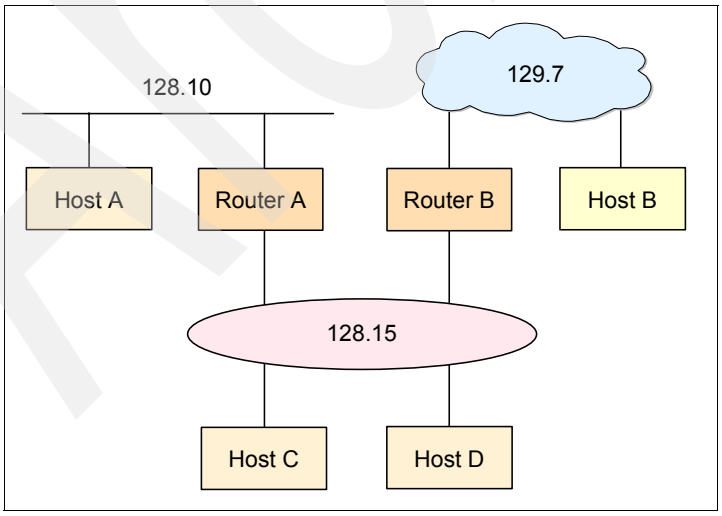


Figure 5-1 Sample network with multiple subnetworks

This example has hosts and routers located in multiple networks, and to achieve connectivity between these hosts, the routers are connected to multiple networks, creating a path between them. In this scenario, if host A wants to connect to host D, both resources must create and maintain a routing table to define which path has to be used to reach its destination. Host D in this example might contain the following (symbolic) entries as listed in Table 5-2.

Table 5-2 IP routing table for Host D

Destination	Router
129.7.0.0	Router B
128.15.0.0	Directly connected
128.10.0.0	Router A
Default	Router A
127.0.0.1	Loopback

The routing table contains routes to different routers in this network. When host D has an IP datagram to forward, it determines which IP address to forward it to using the IP routing algorithm and the routing table.

Since Host D is directly attached to network 128.15.0.0, it maintains a direct route for this network. To reach networks 129.7.0.0 and 128.10.0.0, however, it must have an indirect route through router B and router A, respectively, since these networks are not directly attached to it.

If an IP datagram is to be sent to a destination IP (sub)network, for which there is no explicit entry in the IP routing table of the sending IP host, such an IP datagram will be sent to the default router, router A.

Each host or router in the network needs to know only the next hop's IP address and not the full network topology, in order to reach any given IP network address.

IP routing algorithm

IP uses a unique algorithm to route an IP datagram, called the IP routing algorithm. In a network without subnetworks, each host in the path from source host to destination host will:

1. Inspect the destination address of the packet.
2. Divide the destination address into network and host addresses.
3. Determine whether the destination host is on the same network:
   - If so, send the IP datagram directly to the destination.
   - If not, send the IP datagram to the next router, as defined by the routing tables.

In a subnetted network, each host in the path from source host to destination host will:

1. Inspect the destination address of the packet.
2. Divide the destination address into subnetwork and host addresses.
3. Determine whether the subnetwork is directly attached to it:
   - If so, forward the packet directly to the destination.
   - If not, forward the packet to the next router as defined in the routing tables.

To review this concept, let us look at Figure 5-2 on page 143.

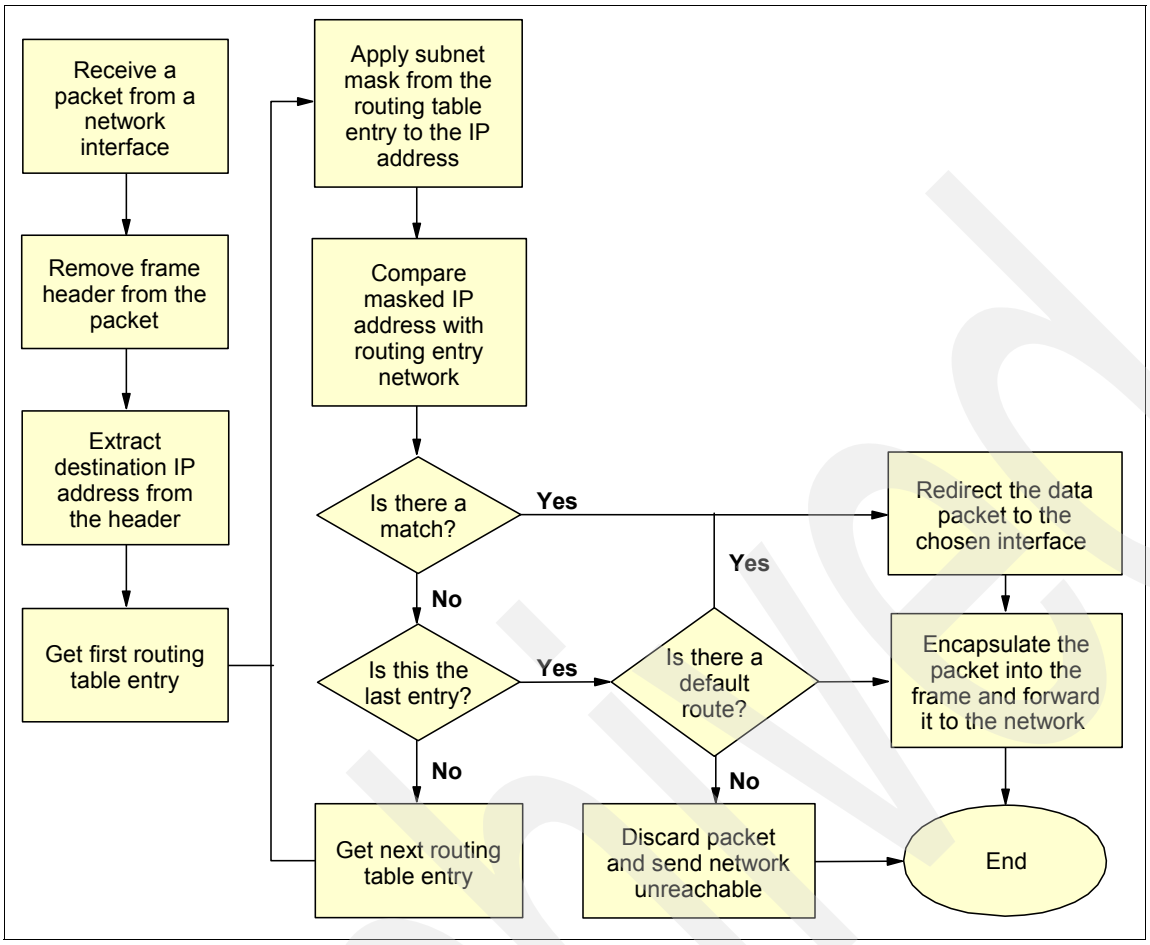


Figure 5-2 Routing algorithm mechanisms

Static and dynamic routing

There are two ways to set up the necessary routing table in a system, using static routing or dynamic routing.

Static routing requires you to manually configure the routing tables yourself. This is part of the configuration steps when you customize TCP/IP. It implies that you know the address of every network you want to communicate with and how to get there. That is, you must know the address of the first router on the way. The task of statically defining all the necessary routes may be simple for a small network and has the advantage of avoiding the network traffic overhead of a dynamic route update protocol. It also allows you to enforce rigid control on the allocation of addresses and resource access. However, it will require manual reconfiguration if you move or add a resource. In many configurations, static routing is sufficient to provide efficient routing. If your configuration is such that you do not have any backup route to remote networks, there is no need for dynamic routing.

Dynamic routing removes the need for static definition of the routing table. The router addresses and information detailing available paths to a destination will be built dynamically. These dynamically built routing tables are automatically exchanged between all the routers in your network. This sharing of the routing information enables the routers to calculate the best path through the network to any destination.

Static definitions are recommended if you have a very simple network or only one physical interface to the network. In this case, the routing table is very simple and there is no reason to

choose dynamic routing. However, if your routing tables become more and more complex due to network growth and if the z/OS system must act as a gateway, then it is far easier to let the system do the work for you by using dynamic routing. Another reason for using dynamic routing is to take advantage of redundant network interfaces and to enable fault-tolerant network attachments.

Interior gateway protocols (IGP)

An interior gateway protocol is a dynamic route update protocol used between dynamic routers running on TCP/IP hosts within a single autonomous system. The protocol is used by the routers to exchange information about which IP routes the IP hosts they have knowledge of. By exchanging IP routing information with each other, the routers are able to maintain a complete picture of all available routes inside an autonomous system.

The more commonly used interior gateway protocols are:

- Routing Information Protocol (RIP) uses a distance vector algorithm to calculate the best path to a destination based on the number of hops in the path. RIP has several limitations. Some of the limitations that exist in RIP version 1 are resolved by RIP version 2.
- RIP version 2 expands RIP version 1. Among the improvements are support for multicasting and variable subnetting. Variable subnetting allows the division of networks into variable size subnets.
- IPv6 RIP uses the same distance vector algorithm used by RIP to calculate the best path to a destination. It is intended to allow routers to exchange information for computing routes through an IPv6-based network.
- Open Shortest Path First (OSPF) uses a link state or shortest path first algorithm. OSPF's most significant advantage compared to RIP is the reduced time needed to converge after a network change. In general, OSPF is more complicated to configure than RIP and might not be suitable for small networks.
- IPv6 OSPF also uses a link state or shortest path first algorithm to calculate the best path to a destination. IPv6 OSPF has the same advantages and more complicated configuration compared to IPv6 RIP, like OSPF compared to RIP.

Table 5-3 lists the main characteristics of the routing protocols supported by the z/OS Communications Server.

Table 5-3 Interior Gateway Protocol characteristics

	RIP v1	RIP v2	IPv6 RIP	OSPF	IPv6 OSPF
Algorithm	Distance vector	Distance vector	Distance vector	Shortest path first	Shortest path first
Network load [a]	High	High	High	Low	Low
CPU processing requirements [a]	Low	Low	Low	High	High
IP network design restrictions	Many	Some	Some	Virtually none	Virtually none
Convergence time	Up to 180 seconds	Up to 180 seconds	Up to 180 seconds	Low	Low
Multicast support [b]	No	Yes	Yes	Yes	Yes
Multiple equal-cost routes	No [c]	No [c]	No [c]	Yes	Yes

a. Depends on network size and stability.

b. Multicast saves CPU cycles on hosts that do not require certain periodic updates, such as OSPF link state advertisements or RIP-2 routing table updates. Multicast frames are filtered out, either in the device driver or directly on the interface card if this host has not joined the specific multicast group.
c. RIP in OMPROUTE allows multiple equal-cost routes only for directly connected destination over redundant interfaces.

Routing daemons

Daemon is a UNIX term for a background server process. Daemons are used for dynamic routing. For z/OS Communications Server IP, there is a multiprotocol routing daemon called OMPROUTE. OMPROUTE supports the RIP version 1, RIP version 2, and OSPF routing protocols. You can send RIP version 1 or RIP version 2, but not both at the same time on a single interface. However, you can configure a RIP interface to receive both versions.

Important: The OROUTED routing daemon is removed from the z/OS Communications Server in V1R7. If OROUTED was used in a prior release and the RIP protocol is still the preferred dynamic routing method in your host configuration, use OMPROUTE to provide RIP support.

5.1.2 Open Shortest Path First (OSPF)

This section provides a brief overview of Open Shortest Path First (OSPF) routing protocol.

The OSPF protocol is based on link-state or shortest path first technology. In other words, OSPF routing tables contain details of the connections between routers, their status (active or inactive), their cost (desirability for routing), and so on. Updates are broadcast whenever a link changes status, and consists merely of a description of the changed status. OSPF can divide its network into topology subsections, known as areas, within which broadcasts are confined. It has been designed for the TCP/IP Internet environment. In CS for z/OS IP, OSPF is configured using the UNIX daemon OMPROUTE.

OSPF features include the following:

- OSPF supports variable length subnetting.
- OSPF can be configured such that all its protocol exchanges are authenticated.
- Only trusted routers can participate in an AS that has been configured with authentication.
- Least-cost routing allows you to configure path costs based on any combination of network parameters. Bandwidth, delay, and metric cost are some examples.
- No limitations to the routing metric. While RIP restricts the routing metric to 16 hops, OSPF has virtually no restrictions.
- Allows multipath routing. OSPF supports multiple paths of equal cost that connect the same points. These paths are then used for network load distribution resulting in more use of the network bandwidth.
- OSPF's area routing capability provides an additional level of routing protection and a reduction in routing protocol traffic.

OSPF terminology

This section describes some of the more common IP routing-related terms and concepts used in OSPF.

- Router ID

  A 32-bit number allocated to each router in the OSPF network protocol. This number is unique in the autonomous system.

- Areas

  OSPF networks may be divided into areas. An area consists of networks and routers that are logically grouped together. All routers within an area maintain the same topology database. All OSPF networks consist of at least one area, the backbone area.

- Backbone area

  All OSPF networks must have a backbone area. The area identifier of the backbone area is always 0.0.0.0. The backbone area is special in that it distributes routing information to all areas connected to it.

- Area border routers

  These are routers that connect two or more areas. The area border routers maintain a topology database of each area to which it is attached. All area border routers must have at least one interface in the backbone area.

- AS boundary routers

  These are the routers that connect the OSPF internetwork and exchange reachability information with other routers in other AS. They may use the exterior gateway protocols. The AS boundary routers are used to import static routes, RIP routes into the OSPF network (and vice-versa).

- Virtual link

  The virtual link is a logical link that connects an area that does not have a physical route to a backbone area. The link is treated as a point-to-point link.

- Neighboring routers

  Routers that have interfaces to the same network are called neighboring routers. To become neighbors, routers must belong to the same OSPF area, use the same security scheme, and have the same Hello and Dead intervals.

- Adjacency

  Neighboring routers are considered adjacent after they have exchanged link state information and synchronized their topology database.

- Link State Advertisement

  Link State Advertisement (LSA) is the unit of data describing the topology of the network and its adjacent routers. LSAs are flooded to other routers once the Hello protocol has established connection.

- Link state database

  Also called the topology database, the link state database contains the link state advertisements that describe the OSPF area. Each router within the OSPF area maintains an identical copy of the link state database.

- Flooding

  Flooding is the OSPF function that distributes link state advertisements and synchronizes the link state database between routers once the network topology has changed.

- OSPF Hello protocol

  The OSPF Hello protocol is used to detect and establish contact with neighboring routers. It dynamically maintains the relationship by periodically sending a Hello packet to all adjacent routers.

- Stub area

  A stub area is one that does not learn advertisements about destinations outside the AS. Instead, its area border routers advertise default routes to represent all those destinations.

Stub areas can have more than one route but no virtual links can be defined through the stub area.

- Designated router

  A designated router is a router on a shared multi-access medium such as a LAN or ATM network that performs most of the routing functions for that network. It must be adjacent to all other routers on the medium.

- Backup Designated Router

  The Backup DR is also adjacent to all other routers on the medium, listens to the DR conversations, and takes over if the DR fails. Once the DR fails, the Backup DR becomes the DR and a new Backup DR is elected according to the router priority value. The router priority value is between 0 and 127. If you do not want a router to be elected a DR, you should configure it with a router priority of zero.

- Transit Area

  An area through which the virtual link ends. Remember that virtual links behave like point-to-point links.

Link state routing

Link-state routing is a concept used in the routing of packet-switched networks. The routers tell every router in the network about its closest neighbors. The entire routing table is not distributed from any router, only the part of the table containing its neighbors. Basically, here is how link state routing is implemented by OSPF:

- Routers identify other routing devices on directly connected networks and exchange identification information with them.
- Routers advertise the details of directly connected network links and the cost of those links by exchanging link state advertisements (LSAs) with other routers in the network.
- Each router creates a link state database based on the link state advertisements: It describes the network topology for the OSPF area.
- All routers in an area maintain an identical link state database.
- A routing table is constructed from the link state database.

Link state advertisements are normally sent under the following circumstances:

- When a router discovers a new neighbor has been added to the area network
- When a connection to a neighbor is unavailable
- When the cost of a link changes
- When basic LSA refreshes are transmitted every 30 minutes

Each area has its own topology and has a gateway that connects it to the rest of the network. It dynamically detects and establishes contacts with its neighboring routers by periodically sending Hello packets.

Link State Advertisements (LSAs)

As mentioned previously, OSPF routers exchange one or more link state advertisements with adjacent routers. LSAs describe the state and cost of an individual router's interfaces that are within a specific area, the status of an individual network component, or the link state database. There are five types of LSA:

1. Router LSAs (Type-1) describe the state and cost of the routers' interfaces within the area. They are generated by every OSPF router and are flooded throughout the area.
2. Network LSAs (Type-2) describe all routers attached to the network. They are generated by the designated router and are flooded through the area.

3. Summary LSAs (Type-3) describe routes to destinations in other areas in the OSPF network. They are generated by an area border router.
4. Summary LSAs (Type-4) are also generated by an area border router and describe routes to an AS boundary router.
5. AS External LSAs (Type-5) describe routes to destinations outside the OSPF network. They are generated by an AS boundary router.

Link state database

The link state database is a collection of OSPF Link State Advertisements. OSPF, being a dynamic IP routing protocol, does not need to have routes defined to it. It dynamically discovers all the routes and the attached routers through its Hello part of the protocol. The OSPF Hello part of the protocol transmits Hello packets to all its router neighbors to establish connection. Once the neighbors have been discovered, the connection is made.

Before the link state databases are exchanged, however, the OSPF routers transmit only their LSA headers. After receiving the LSA headers, they are examined for any corruptions. If everything is fine, the request for the most recent LSAs is made. This process is bidirectional between routers. Once the Hello protocol has concluded that all the connections have been established, the link state databases are synchronized. This exchange is performed starting with the most recently updated LSAs. The link state databases are synchronized until all router LSAs in the network (within an area) have the same information. The link state protocol maintains a loop-free routing because of the synchronization of the link state databases.

Physical network types

OSPF supports a combination of different physical networks. In this section, we give a brief description of each physical network and how OSPF supports them.

- Point-to-point

  A network that connects two routers together. A PPP serial line that connects two routers is an example of a point-to-point network.

- Point-to-multipoint

  Networks that support more than two attached routers with no broadcast capabilities. These networks are treated as a collection of point-to-point links. OSPF does not use designated routers on point-to-multipoint networks. The Hello protocol is used to detect the status of the neighbors.

- Broadcast multiaccess

  Networks that support more than two attached routers and are capable of addressing a single message to all the attached routers. OSPF's Hello Protocol discovers the adjacent routers by periodically sending/receiving Hello packets. This is a typical example of how OSPF makes use of a broadcast network to its advantage. OSPF utilizes multicast in a broadcast network if implemented. Token-ring and Ethernet are some of the examples of a broadcast network.

- Nonbroadcast multiaccess (NBMA)

  Networks that support more than two attached routers but have no broadcast capabilities. Since NBMA does not support multicasting, the OSPF Hello packets must be specifically addressed to each router. Since OSPF cannot discover its neighbors through broadcasting, more configuration is required: all routers attached to the NBMA network must be configured. These routers must be configured whether or not they are eligible to become designated routers. ATM and X.25 public data networks are examples of non-broadcast networks.

5.1.3 Routing Information Protocol (RIP)

This section provides an overview of the RIP protocol. RIP is designed to manage relatively small networks.

RIP uses a hop count (distance vector) to determine the best possible route to a network or host. The hop count is also known as the routing metric, or the cost of the route. A router is defined as being zero hops away from its directly connected networks, one hop away from networks that can be reached through one gateway, and so on. The fewer hops, the better. The route that has the fewest hops will be the preferred path to a destination. A hop count of 16 means infinity, or that the destination cannot be reached. Thus, very large networks with more than 15 hops between potential partners cannot make use of RIP. The information is kept in a distance vector table, which is periodically advertised to each neighboring router. The router also receives updates from neighboring gateways and uses these to update its routing tables. If an update is not received for three minutes, a gateway is assumed to be down, and all routes through that gateway are set to a metric of 16 (infinity).

Basic distance vector algorithm

The following procedure is carried out by every entity that participates in the RIP routing protocol. This must include all of the gateways in the system. Hosts that are not gateways may participate as well.

- Keep a table with an entry for every possible destination in the system. The entry contains the distance D to the destination, and the first gateway G on the route to the network.
- Periodically, send a routing update to every neighbor. The update is a set of messages that contains all the information from the routing table. It contains an entry for each destination, with the distance shown to that destination.
- When a routing update arrives from the neighbor G', add the metric associated with the network that is shared with G'. Call the resulting distance D'. Compare the resulting distance with the current routing table entries. If the new distance D' for N is smaller than the existing value D, adopt the new route. That is, change the table entry for N to have metric D' and gateway G'. If G' is the gateway from which the existing route came, G' = G, then use the new metric, even if it is larger than the old one.

RIP version 1

RIP is a protocol that manages IP routing table entries dynamically. The gateways using RIP exchange their routing information in order to allow the neighbors to learn of topology changes. The RIP server updates the local routing tables dynamically, resulting in current and accurate routing tables. The protocol is based on the exchange of protocol data units (PDUs) between RIP servers (OMPROUTE). There are various types of PDUs, but the two most important PDUs are:

REQUEST PDU Sent from an OMPROUTE server as a request to other OMPROUTE servers to transmit their routing tables immediately.

RESPONSE PDU Sent from an OMPROUTE server to other OMPROUTE servers either as a response to a REQUEST PDU or as a result of expiration of the broadcast timer (every 30 seconds).

RIP V1 limitations

Since RIP is designed for a specific network environment, it has some limitations. These should be considered before implementing RIP in your network.

- RIP V1 declares a route invalid if it passes through 16 or more gateways. Therefore, RIP V1 places a limitation of 15 hops on the size of a large network.

- RIP V1 uses fixed metrics to compare alternative routes versus actual parameters, such as measured delay, reliability, and load. This means that the number of hops is the only parameter that differentiates a preferred route from non-preferred routes.
- The routing tables can take a relatively long time to converge or stabilize.
- RIP V1 does not support variable subnet masks or variable subnetting because it does not pass the subnet mask in its routing advertisements. Variable subnet masking refers to the capability of assigning different subnet masks to interfaces that belong to the same Class A, B, or C network.
- RIP V1 does not support discontiguous subnets. Discontiguous subnets are built when interfaces belonging to the same Class A, B, or C network but to different subnets that are not adjacent to each other. Rather, they are separated from each other by interfaces that belong to a different network. With RIP version 1, discontiguous subnets represent unreachable networks. If you find it necessary to build discontiguous subnets, you must use one of the following techniques:
  - An OSPF implementation
  - RIP version 2 protocol
  - Static routing

RIP version 2

Rather than being another protocol, RIP V2 is an extension to the functions provided by RIP V1. To use these new functions RIP V2 routers exchange the same RIP V1 messages. The version field in the message will specify version number 2 for RIP messages that use authentication or carry information in any of the newly defined fields. RIP V2 protocol extensions provide features such as:

- Route tags to provide EGP-RIP and BGP-RIP implementation. Route tags are used to separate *internal* RIP routes (routes for networks within the RIP routing domain) from *external* RIP routes, which may have been imported from an EGP (external gateway protocol) or another IGP. OMPROUTE does not generate route tags, but preserves them in received routes and readvertises them when necessary.
- Variable subnetting support. Variable length subnet masks are included in routing information so that dynamically added routes to destinations outside subnetworks or networks can be reached.
- Immediate next hop for shorter paths. Next hop IP addresses, whenever applicable, are included in the routing information. Their purpose is to eliminate packets being routed through extra hops in the network. OMPROUTE will not generate immediate next hops, but will preserve them if they are included in RIP packets.
- Multicasting to reduce load on hosts. An IP multicast address 224.0.0.9, reserved for RIP version 2 packets, is used to reduce unnecessary load on hosts that are not listening to RIP version 2 messages. RIP version 2 multicasting is dependent on interfaces that are multicast-capable.
- Authentication for routing update security. Authentication keys can be included in outgoing RIP version 2 packets for authentication by adjacent routers as a routing update security protection. Likewise, incoming RIP version 2 packets are checked against local authentication keys. The authentication keys are configurable on a router-wide or per-interface basis.
- Configuration switches for RIP V1 and RIP V2 packets. Configuration switches are provided to selectively control which versions of RIP packets are to be sent and received over network interfaces. You can configure them router-wide or per-interface.
- Supernetting support. The supernetting feature is part of the Classless InterDomain Routing (CIDR) function. Supernetting provides a way to combine multiple network routes

into fewer supernet routes. Therefore, the number of network routes in the routing tables becomes smaller for advertisements. Supernet routes are received and sent in RIP V2 messages.

RIP V2 packets are backward compatible with existing RIP V1 implementations. A RIP V1 system will process RIP V2 packets but without the RIP V2 extensions and broadcast them as RIP V1 packets to other routers. Note that routing problems may occur when variable subnet masks are used in mixed RIP V1 and RIP V2 systems. RIP V2 is based on a distance vector algorithm, just as RIP V1 is.

5.1.4 IPv6 dynamic routing

Dynamic routing in a IPv6 network can be implemented in a z/OS Communications Server in two different ways:

- IPv6 dynamic routing using router discovery
- IPv6 dynamic routing using OMPROUTE

IPv6 dynamic routing using router discovery

Enabling IPv6 router discovery in the z/OS Communications Server requires no additional z/OS Communications Server configuration. All that is needed is at least one IPv6 interface that is defined and started, and at least one adjacent router through that interface that is configured for IPv6 router discovery. If these things exist, then the z/OS Communications Server begins receiving router advertisements from the adjacent routers. Depending on the configuration in the adjacent routers, the following types of routes may be learned from the received router advertisements:

- Default route for which the originator of the router advertisement is the next hop
- Direct routes (no next hop) to prefixes that reside on the link shared by the z/OS Communications Server and the originator of the router advertisement

IPv6 dynamic routing using OMPROUTE

For IPv6, OMPROUTE implements the IPv6 RIP protocol described in RFC 2080 (RIPng for IPv6) and the IPv6 OSPF protocol described in RFC 2740 (OSPF for IPv6). It provides an alternative to the static TCP/IP gateway definitions. The z/OS host running with OMPROUTE becomes an active OSPF or RIP router in a TCP/IP network. Either or both of these routing protocols can be used to dynamically maintain the host IPv6 routing table. For example, OMPROUTE can detect when a route is created, is temporarily unavailable, or if a more efficient route exists. If both IPv6 OSPF and IPv6 RIP protocols are used simultaneously, IPv6 OSPF routes will be preferred over IPv6 RIP routes to the same destination.

RIPng or RIP next generation

RIP Next Generation (RIPng) is a distance vector routing protocol for IPv6 that is defined in RFC 2080. RIPng for IPv6 is an adaptation of the RIP V2 protocol to advertise IPv6 network prefixes. RIPng for IPv6 uses UDP port 521 to periodically advertise its routes, respond to requests for routes, and advertise route changes.

RIPng for IPv6, like other distance vector protocols, has a maximum distance of 15, in which 15 is the accumulated cost (hop count). Locations that are a distance of 16 or further are considered unreachable. RIPng for IPv6 is a simple routing protocol with a periodic route-advertising mechanism designed for use in small to medium-sized IPv6 networks. RIPng for IPv6 does not scale well to a large or very large IPv6 network.

Differences between RIPng and RIP-2

There are two important distinctions between RIP-2 and RIPng:

- Support for authentication

  The RIP-2 standard includes support for authenticating a node transmitting routing information. RIPng does not include any native authentication support. Rather, RIPng uses the security features inherent in IPv6. In addition to authentication, these security features provide the ability to encrypt each RIPng packet. This can control the set of devices that receive the routing information.One consequence of using IPv6 security features is that the AFI field within the RIPng packet is eliminated. There is no longer a need to distinguish between authentication entries and routing entries within an advertisement.

- Support for IPv6 addressing formats

  The fields contained in RIPng packets were updated to support the longer IPv6 address format.

OSPF for IPv6

OSPF for IPv6 is a link state routing protocol defined in RFC 2740 and designed for routing table maintenance within a single autonomous system. OSPF for IPv6 is an adaptation of the OSPF routing protocol version 2 for IPv4 defined in RFC 2328.

IPv6 OSPF is classified as an Interior Gateway Protocol (IGP). This means that it distributes routing information between routers belonging to a single autonomous system (AS), a group of routers all using a common routing protocol. The IPv6 OSPF protocol is based on link-state or shortest path first (SPF) technology.

At a glance, the OSPF implementation is basically the same as it is for IPv4, except for some primary differences.

Primary differences between IPv6 OSPF and IPv4 OSPFv2

IP addressing and topology semantics have been separated where possible (many LSAs do not carry IP addresses at all, only abstract topology information). Removing IP addressing from the topology description makes OSPFv3 more protocol independent.

New LSA types are added (to carry addressing and link-local information). Since IP addressing has been removed from some of the basic LSA types, new LSA types are provided to communicate IP addresses, which routers then correlate to topology information in other LSA types.

The Concept of Flooding Scope is added (scopes are link, area, autonomous system). It indicates how far an advertisement may be flooded. For example, link scope means an LSA can only be flooded on the originating link.

Support for Unknown LSA types is added (makes the protocol more extensible). Unknown LSA types can be ignored, or they can be stored and forwarded by the router, depending on the settings of bits in the LSA type field. This vastly improves interoperability between routers running different versions of the protocol; for example, a designated router could conceivably have a lower level of support than another router on the same link; since the designated router floods on behalf if the other routers on the link it could store and forward unknown LSA types received from its peers.

Multiple OSPF instances are supported on a link. An “instance id” field is added to OSPF headers, and OSPF processes only process packets whose instance ID matches their own. This opens up the possibility of one link belonging to completely different autonomous systems.

Subnet loses its importance, replaced by *Link* (since multiple IPv6 prefixes per link are allowed and expected, routing by subnet/prefix makes less sense). In OSPFv4, most routing is done by subnet. In OSPFv3 it is done by link. This is because in IPv6 a *subnet* (prefix) does not always uniquely identify a link, and a link can have more than one prefix assigned.

5.1.5 Choosing the routing protocol

The choice of a routing protocol is a major decision for the network administrator. It has a major impact to overall network performance. The selection depends on network complexity, size, and administrative policies. The protocol chosen for one type of network may not be appropriate for other types of networks. Each unique environment must be evaluated against a number of fundamental design requirements:

- Scalability to large environments: The potential growth of the network dictates the importance of this requirement. If support is needed for large, highly redundant networks, link state or hybrid algorithms should be considered. Distance vector algorithms do not scale into these environments.
- Stability during outages: Distance vector algorithms may introduce network instability during outage periods. The counting to infinity problems may cause routing loops or other non-optimal routing paths. Link state or hybrid algorithms reduce the potential for these problems.
- Speed of convergence: Triggered updates provide the ability to immediately initiate convergence when a failure is detected. All three types of protocols support this feature. One contributing factor to convergence is the time required to detect a failure. In OSPF and EIGRP networks, a series of hello packets must be missed before convergence begins. In RIP environments, subsequent route advertisements must be missed before convergence in initiated. These detection times increase the time required to restore communication.
- Metrics: Metrics provide the ability to groom appropriate routing paths through the network. Link state algorithms consider bandwidth when calculating routes. EIGRP improves this to include network delay in the route calculation.
- Vendor interoperability: The types of devices deployed in a network indicate the importance of this requirement. If the network contains equipment from a number of vendors, standard routing protocols should be used. The IETF has dictated the operating policies for the distance vector and link state algorithms described in this document. Implementing these algorithms avoids any interoperability problems encountered with nonstandard protocols.
- Ease of implementation: Distance vector protocols are the simplest routing protocol to configure and maintain. Because of this, these protocols have the largest implementation base. Limited training is required to perform problem resolution in these environments.

In small, non-changing environments, static routes are also simple to implement. These definitions change only when sites are added or removed from the network.

The administrator must assess the importance of each of these requirements when determining the appropriate routing protocol for an environment.

5.1.6 For additional information

For more details on this subject, refer to:

- *TCP/IP Tutorial and Technical Overview*, GG24-3376
- *The Basics of IP Network Design*, SG24-2580
- *z/OS CS: IP Configuration Guide*, SC31-8775

- *z/OS CS: IP Configuration Reference*, SC31-8776

5.2 Why IP routing is important

IP routing is one of the most important processes that takes place at the network layer. It is the function that really defines Layer 3 of the OSI Reference Model. Routing is what enables local networks to be linked to larger networks, virtually allowing a host to communicate with any host located anywhere. It provides the desired network scalability, while it still can be managed within its own AS boundaries.

5.3 Using OMPROUTE in a z/OS environment

OMPROUTE is the strategic routing daemon for CS for z/OS. For IPv4, OMPROUTE implements the OSPF protocol described in RFC 1583 (OSPF version 2), the OSPF subagent protocol described in RFC 1850 (OSPF version 2 Management Information Base), and the RIP protocols described in RFC 1058 (Routing Information Protocol) and in RFC 1723 (RIP version 2 - Carrying Additional Information).

For IPv6, OMPROUTE implements the IPv6 RIP protocol described in RFC 2080 (RIPng for IPv6) and the IPv6 OSPF protocol described in RFC 2740 (OSPF for IPv6).

It provides an alternative to the static TCP/IP gateway definitions. The z/OS host running with OMPROUTE becomes an active OSPF or RIP router in a TCP/IP network. Either or both of these routing protocols can be used to dynamically maintain the host IPv6 routing table.

5.3.1 Overview

OMPROUTE manages a TCP/IP stack's routing table. It is not involved in any decisions made by the stack when routing a packet to its destination.

During initialization OMPROUTE deletes all dynamic routes from the TCP/IP stack's routing table. OMPROUTE then repopulates the stack routing table using information learned via the routing protocols.

OMPROUTE does not make use of the BSDROUTINGPARMS statement. Instead, its parameters are defined in the OMPROUTE configuration file. The OMPROUTE configuration file is used to define both OSPF and RIP environments.

The OSPF and RIP protocols are communicated over interfaces defined with the OSPF_INTERFACE and RIP_INTERFACE configuration statements. Interfaces that are not involved in the communication of the RIP or OSPF protocol are configured to OMPROUTE via the INTERFACE configuration statement (unless it is a non-point-to-point interface and all default values specified on the INTERFACE statement are acceptable).

A one-to-one relationship exists between an OMPROUTE and a TCP/IP stack. OSPF/RIP support for multiple TCP/IP stacks requires multiple instances of OMPROUTE.

In an OMPROUTE environment, OSPF takes precedence over RIP and other routing protocols. If both OSPF and RIP protocols are used, OSPF routes will be preferred over RIP routes to the same destination.

OMPROUTE allows the generation of multiple, equal-cost routes to a destination, thus providing load-balancing support.

OMPROUTE supports Virtual IP Addressing (VIPA) to handle network interface failures by switching to alternate paths. VIPA routes are included in the OSPF and RIP advertisements to adjacent routers. Adjacent routers learn about VIPA routes from advertisements and can use them to reach destinations at the z/OS.

Special care is needed when coding static routes in an OMPROUTE environment. There are two types of static routes: Nonreplaceable (the default) and replaceable. OMPROUTE will replace a replaceable static route if it detects a dynamic route to the same destination. OMPROUTE cannot replace a nonreplaceable static route, even it if has detected a better, dynamic route to the same destination.

Note: OMPROUTE is the only routing deamon included in the z/OS Communications Server 1.7.

Configuring OMPROUTE

Some of the configuration of OMPROUTE is the same regardless of whether you are implementing RIP, OSPF, or both in your network. To implement and configure OMPROUTE in the z/OS Communications Server, the following steps must be followed:

1. Create the OMPROUTE catalogued procedure (if starting OMPROUTE as a z/OS procedure).

   Tip: OMPROUTE can be started as a z/OS procedure, from the z/OS shell, or from AUTOLOG.

2. Define the OMPROUTE environment variables.
3. Update the TCPIP.DATA or RESOLVER_CONFIG file for use by the OMPROUTE application server address space.
4. Update PROFILE.TCPIP.
5. Create the OMPROUTE configuration file.

OMPROUTE uses the z/OS operator console, syslogd, CTRACE, and STDOUT for its logging and tracing. The z/OS operator console and syslogd are used for major events such as initialization, termination, and error conditions. CTRACE is used for tracing the receipt and transmission of OSPF/RIP packets, as well as communications between OMPROUTE and the TCP/IP stack. STDOUT is used for detailed tracing and debugging.

Important: OMPROUTE must be started by a RACF-authorized user ID.

For more details on this subject, refer to *z/OS CS: IP Configuration Guide*, SC31-8775, or *z/OS CS: IP Configuration Reference*, SC31-8776.

5.4 The common design scenarios for IP routing

In this section we discuss two basics design scenarios for implementing the most commonly used routing mechanisms. These scenarios have the purpose of showing the best practice designs based on a z/OS Communication Server perspective, where the z/OS Communications Server host is being used as an application or server host and the routing methods used are primarily to provide access to network resources and vice versa. With this in mind, care must be taken to ensure that the z/OS Communications Server host is not overly burdened with routing work. Unlike routers or other network boxes whose sole purpose

is routing, an application host z/OS Communications Server will be doing many things other than routing, and it is not desirable for a large percentage of machine resources (memory and CPU) to be used for routing tasks, as can happen in very complex or unstable networks.

We discuss the following design scenarios:

- Design scenario 1: Static routing
- Design scenario 2: OSPF routing with OMPROUTE

For a dynamic routing solution, we based our decision for using OSPF on lesser network load, more IP network design flexibility, and lower convergence time compared to RIP v1 and RIP v2 (see Table 5-3 on page 144). Although OSPF requires higher CPU processing, there is a technique for reducing that requirement, which we will discuss in the OSPF design scenario.

5.4.1 Design scenario 1: Static routing

When planning to design a static routing environment on a z/OS Communication Server, some issues have to be addressed. Keep in mind that static routing is normally used in simple network topologies. The main reason is because static routes are configured manually for each router by the system administrator. Available network interfaces and routes through the network must be determined before the routes are configured. Except for potential ICMP router redirections, routers do not communicate with each other about the topology of the network.

Our recommended scenario to create a static routing environment is to use as few static routing definitions as possible, keeping in mind that our z/OS system is basically an application server environment. A good practice would be to define only the default gateways to the exterior networks, and let the routers do the exterior routing.

In the routers, it is recommended that we only create the route definitions to the VIPA subnetworks, either static or dynamically defined. The interior subnetworks, such as XCF and HiperSockets, do not need to be reached by the corporate network, so they do not need to be defined.

To show how to set up a static routing design we work with the scenario shown in Figure 5-3 on page 157.

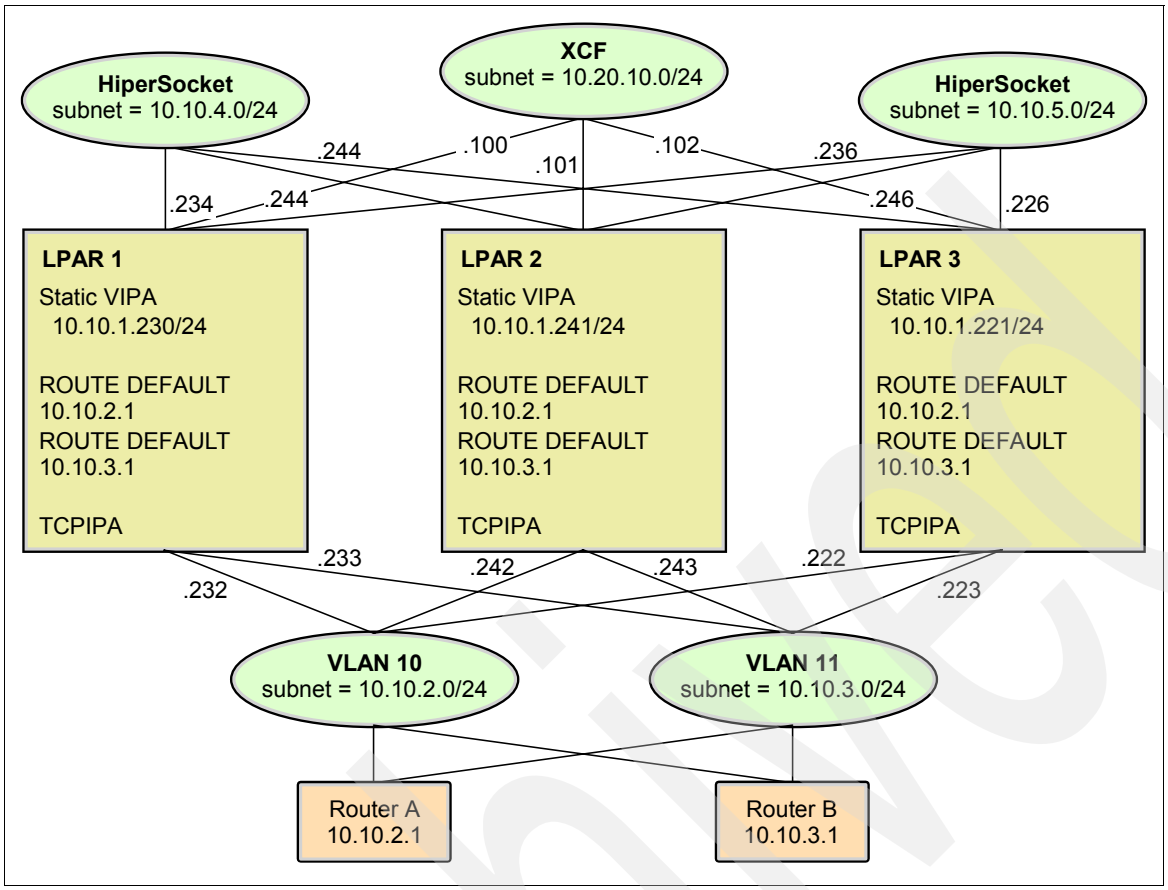


Figure 5-3 Static routing scenario

Dependencies

All subnetworks being used by the application servers, including static and dynamic VIPAs, must have static routing definitions in the routers.

Advantages

There are circumstances when static routing is useful, for example:

- To define a route that is not automatically advertised within a network
- When line utilization or tariffs make it undesirable to send routing advertisements through lower capacity connections
- To ensure traffic for a specific destination takes a specific route through the network
- Used in conjunction with the OMPROUTE routing daemon, to provide backup routes when OMPROUTE cannot find a dynamic route to a destination (replaceable static routes)
- To define a default route that is used to forward traffic when the routing table does not contain a more specific route to a destination
- To provide a secure network by defining routes to authorized (sub)networks only

Considerations

Some issues must be carefully considered when deciding to implement a static routing environment:

- The only way to remove static routes from the routing table is for the network administrator to update the routing table.
- If a destination (sub)network becomes unreachable, the static routes for that (sub)network will remain in the routing table, and packets will still be forwarded to the destination.
- The routing table's management is manual, increasing the possibilities of outages caused by definition errors.

5.4.2 Design scenario 2: OSPF routing with OMPROUTE

The most common and recommended way to use dynamic routing in the z/OS environment is to define the stack as a OSPF Stub Area or even better as a Totally Stubby Area.

Stub Areas minimize storage and CPU processing at the nodes that are part of the Stub Area because they maintain less knowledge about the topology of the Autonomous System (AS) than do other types of non-backbone routers. They maintain knowledge only of intra-area destinations—summaries of inter-area destinations and default routes within the AS in order to reach external destinations.

A Totally Stubby Area receives less routing information than a Stub Area. It only knows of intra-area destinations and default routes within the Stub Area to reach external destinations.

Our OSPF scenario is represented by Figure 5-4 on page 159.

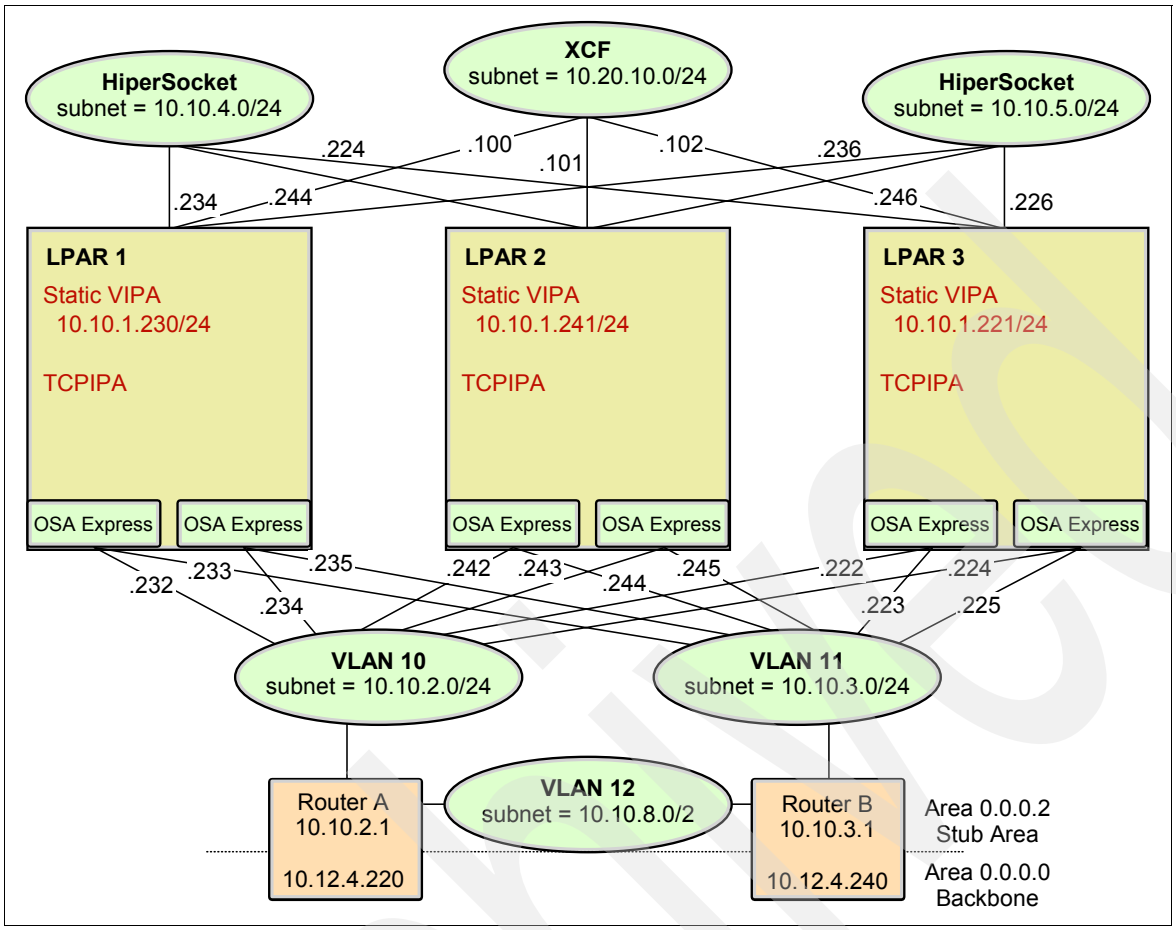


Figure 5-4 OSPF totally stub area scenario

Dependencies

The IP routers that will be involved in establishing access to the external network must support OSPF and the configuration parameters set in OMPROUTE must match with those defined to the IP routers.

Advantages

The primary way to reduce the routing burdens on the z/OS Communications Server host is by use of OSPF areas. A z/OS Communications Server application host or sysplex can be placed into a non-backbone area with dedicated routers acting as area-border routers. The area-border routers would advertise the z/OS Communications Server's resources to other attached areas (for example, the backbone) and would summarize the network outside the local area to the z/OS Communications Server hosts.

Considerations

A z/OS Communications Server host is usually used as an application server and the routing daemon is running primarily to provide access to network resources and vice versa. With this in mind, care must be taken to ensure that the z/OS Communications Server host is not overly burdened with routing work.

The z/OS Communications Server should not be configured as a backbone router, either intentionally or inadvertently. Careful network design can minimize the routing burdens on the z/OS Communications Server (application host), without compromising the accessibility.

5.4.3 Recommendations

If your network environment is small and manageable with few to no network changes anticipated, then we recommend using static routes (keeping in mind that your z/OS system is basically an application server environment). A good practice would be to define only the default gateways to the exterior networks, and let the routers do the exterior routing.

If your network environment is medium to large, or has high availability requirements, then we recommend using OMPROUTE to define the z/OS Communications Server environment as an OSPF Stub Area. A Stub Area can be configured so that route summaries from other areas are not flooded into the stub area by the area border routers. When this is done, only routes to destinations within the Stub Area are shared among the hosts. Default routes are used to represent all destinations outside the Stub Area. The Stub Area's resources are still advertised to the network at large by the area-border routers. You can use this optimization, sometimes referred to as a Totally Stubby Area.

5.5 How IP routing is implemented

Based upon our recommendations above, we show implementation details for the following scenarios:

- Static routing scenario
- Using OMPROUTE to implement an OSPF routing scenario

5.5.1 Static routing scenario

In this section we implement the static routing scenario, as shown in Figure 5-5 on page 161, and discussed in "Design scenario 1: Static routing" on page 156.

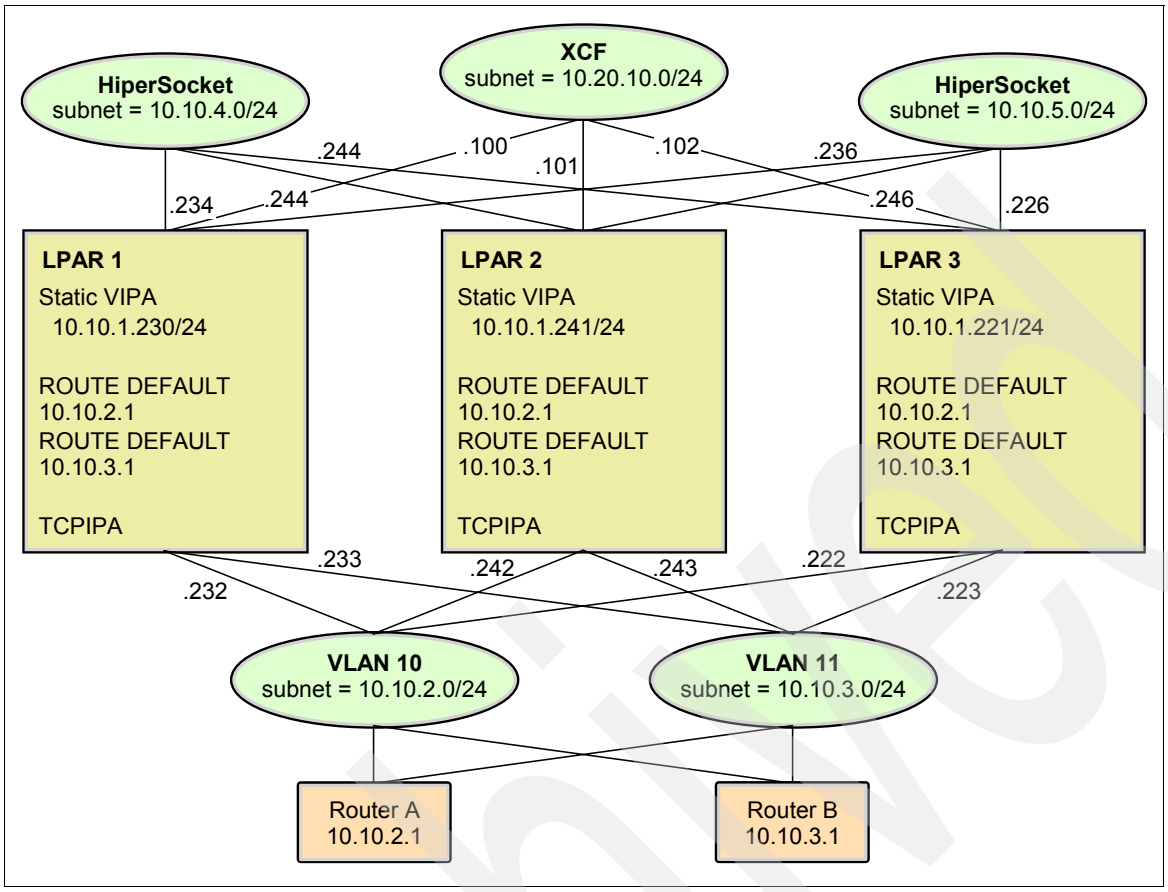


Figure 5-5 Static routing implementation scenario

Implementation tasks

To implement the static routing scenario, we will follow these implementation steps:

- In a TCP/IP profile:
  - Add the BeginRoutes statements to define all interfaces direct route.
  - Example 5-2 on page 162.
- In the backbone routers, define the static routes to reach the VIPA hosts and subnetworks.

Configuration examples

In this section we review some configuration examples.

Add the BeginRoutes statements to define all interfaces direct route

We added all direct route statements that represents our physical links as shown in Example 5-1. Note that the first hop parameter is defined as an equal sign (=) **1** to identify this as a direct route. The direct routes are also created to define the subnet mask for each link will have. It can be configured either by specifying the entire mask **2**, or by specifying the number of left most significant bits **3**.

Example 5-1 Direct routes configuration

```
;
BEGINRoutes
; Direct Routes - Routes that are directly connected to my interfaces
```

```
;     Destination     Subnet Mask   First Hop Link Name     Packet Size ;
 ROUTE 10.10.2.0       255.255.255.0 = 1       OSA2080LNK    mtu defaultsize
 ROUTE 10.10.3.0       255.255.255.0 =         OSA20A0LNK    mtu defaultsize
 ROUTE 10.10.2.0       255.255.255.0 =         OSA20c0LNK    mtu defaultsize
 ROUTE 10.10.3.0       255.255.255.0 =         OSA20e0LNK    mtu defaultsize
 ROUTE 10.10.4.0     2 255.255.255.0 =         IUTIQDF4LNK   mtu defaultsize
 ROUTE 10.10.4.0/24 3                =         IUTIQDF5LNK   mtu defaultsize
 ROUTE 10.10.5.0       255.255.255.0 =         IUTIQDF6LNK   mtu defaultsize
```

Add the BeginRoutes statements to define the indirect routes

After we created the direct route statements, the next step was to add the indirect routes needed to reach external networks, as shown in Example 5-2. Note that the first hop now shows the address of the router we used to reach the defined subnetwork or host 1.

Example 5-2 Indirect routes implementation

```
; Indirect Routes - Routes that are not directly connected to my interfaces
;     Destination     Subnet Mask    First Hop    Link Name     Packet Size ;
 ROUTE 10.12.4.0      255.255.255.0   10.10.2.1 1 OSA2080LNK    mtu defaultsize
 ROUTE 10.12.4.221    255.255.255.255 10.10.2.1   OSA20A0LNK    mtu defaultsize
```

Add the BeginRoutes statements to define the default routes

The last step to create our routing table was to add one or more default gateway statements to route all packets being sent to unknown destinations. When multiple default routes are defined, the traffic will be sent to the first default route defined. If the MULTIPATH parameter is specified on the IPCONFIG statement, then all default routes will be used.

Example 5-3 Default gateway configuration

```
; Default Route - All packets to an unknown destination are routed
;through this route.
;     Destination     First Hop    Link Name   Packet Size
 ROUTE DEFAULT         10.10.2.1    OSA2080LNK  mtu defaultsize
 ROUTE DEFAULT         10.10.3.1    OSA20A0LNK  mtu defaultsize
 ROUTE DEFAULT         10.10.2.2    OSA20C0LNK  mtu defaultsize
```

Managing the static routing table

After all routes have been defined, the static routing table will be created when the stack is started. Once it is up and running the static routing table can only be modified in these situations:

- Replace the routing table using the VARY TCPIP,,OBEYFILE command.
- If a static route is defined on a BEGINROUTES statement as being replaceable, it can be replaced if a dynamic route is discovered by OMPROUTE.

Important: When using the OBEYFILE command to change a static routing table, you must execute the command with the entire routing table, even if only one route will be added or changed, because OBEYFILE replaces the table, it does not change it.

Verification

To verify if the static routing table is built as expected, the following commands can be used.

Note: The commands shown in this section can be executed as a TSO command, or USS command, or, if it is a **netstat**, from a console display command. Our output samples are the results of TSO commands.

- **Netstat, devlinks**

  Use this command to review the status of all devices defined in the TCP/IP environment. If a device is not ready, there will be no routing through this device. Example 5-4 shows the resulting display of this command, for one device only.

Example 5-4 Netstat DEVlink command results

```
DevName: OSA2080           DevType: MPCIPA
  DevStatus: Ready
  LnkName: OSA2080LNK        LnkType: IPAQENET   LnkStatus: Ready
    NetNum: n/a  QueSize: n/a  Speed: 0000000100
    IpBroadcastCapability: No
    CfgRouter: Non                   ActRouter: Non
    ArpOffload: Yes                  ArpOffloadInfo: Yes
    ActMtu: 1492
    VLANid: 10                       VLANpriority: Disabled
            DynVLANRegCfg: No                DynVLANRegCap: Yes
    ReadStorage: GLOBAL (4096K)      InbPerf: Balanced
    ChecksumOffload: Yes             SegmentationOffload: Yes
    SecClass: 255
  BSD Routing Parameters:
    MTU Size: 1492             Metric: 10
    DestAddr: 0.0.0.0          SubnetMask: 255.255.255.0
  Multicast Specific:
    Multicast Capability: Yes
    Group             RefCnt
    -----             ------
    224.0.0.1         0000000001
  Link Statistics:
    BytesIn                          = 297809
    Inbound Packets                  = 1522
    Inbound Packets In Error         = 2620
    Inbound Packets Discarded        = 0
    Inbound Packets With No Protocol = 0
    BytesOut                         = 358
    Outbound Packets                 = 4
    Outbound Packets In Error        = 0
    Outbound Packets Discarded       = 0
```

- **Netstat, ROUTe**

  Use the **Netstat ROUTe** command to display the active routing information in this stack. A sample of the command is shown in Example 5-5. The default routes must have a first hop IP address **1**. The direct interfaces has no gateway (first hop) address **2**, while the indirect routes shows its gateway address **3**.

Example 5-5 Netstat ROUTe resulting display

```
MVS TCP/IP NETSTAT CS V1R7       TCPIP Name: TCPIPA          20:35:51
IPv4 Destinations
Destination         Gateway          Flags     Refcnt  Interface
-----------         -------          -----     ------  ---------
Default             10.10.2.1        UGZ       000000  OSA2080LNK 1
Default             10.10.3.1        UGZ       000000  OSA20A0LNK
Default             10.10.2.2        UGZ       000000  OSA20C0LNK
```

```
Default            10.10.3.2        UGZ       000000  OSA2OE0LNK
10.10.1.221/32     10.20.10.102     UGHO      000000  IQDIOLNKOA14OA64 3
10.10.1.230/32     0.0.0.0          UH        000000  STAVIPA1LNK 2
10.10.1.241/32     10.20.10.101     UGHO      000000  IQDIOLNKOA14OA64
10.10.2.0/24       0.0.0.0          UC        000000  OSA2OC0LNK
10.10.2.0/24       0.0.0.0          UC        000000  OSA2O80LNK
10.10.2.232/32     0.0.0.0          UH        000000  OSA2O80LNK
10.10.2.234/32     0.0.0.0          UH        000001  OSA2OC0LNK
10.10.3.0/24       0.0.0.0          UC        000000  OSA2OE0LNK
10.10.3.0/24       0.0.0.0          UC        000000  OSA2OA0LNK
10.10.3.233/32     0.0.0.0          UH        000000  OSA2OA0LNK
10.10.3.235/32     0.0.0.0          UH        000000  OSA2OE0LNK
10.10.4.0/24       0.0.0.0          UO        000000  IUTIQDF5LNK
10.10.4.0/24       0.0.0.0          UO        000000  IUTIQDF4LNK
10.10.4.234/32     0.0.0.0          UH        000000  IUTIQDF4LNK
10.10.4.235/32     0.0.0.0          UH        000000  IUTIQDF5LNK
10.10.5.0/24       0.0.0.0          UO        000000  IUTIQDF6LNK
10.10.5.236/32     0.0.0.0          UH        000000  IUTIQDF6LNK
10.10.20.130/32    0.0.0.0          UHS       000000  EZASAMEMVS
```

Problem determination

The problem determination in a static routing scenario is done by executing the same display commands we used in the verification section.

If the routing table and the devices seem to be OK, a PING command will help us to check whether the address being requested is reachable.

If not, we can trace the routing path using the command TRACERTE to discover which paths or hops are being used to reach the destination address. Example 5-6 shows the command being executed.

Example 5-6 Tracerte command results

```
CS V1R7: Traceroute to 10.12.4.221 (10.12.4.221):
1 router1 (10.10.2.1)  0 ms  0 ms  0 ms
2 tot177 (10.12.4.221)  0 ms  0 ms  0 ms
```

If it is necessary to go further in the problem determination, a packet trace must be taken to analyze the problem based on the interface traffic.

5.5.2 Using OMPROUTE to implement an OSPF routing scenario

In this section we implement the recommended dynamic routing scenario with OMPROUTE. We show the steps needed to configure the OMPROUTE and define an OSPF STUB Area in a z/OS TCP/IP environment. The OSPF Area is shown in Figure 5-6 on page 165.

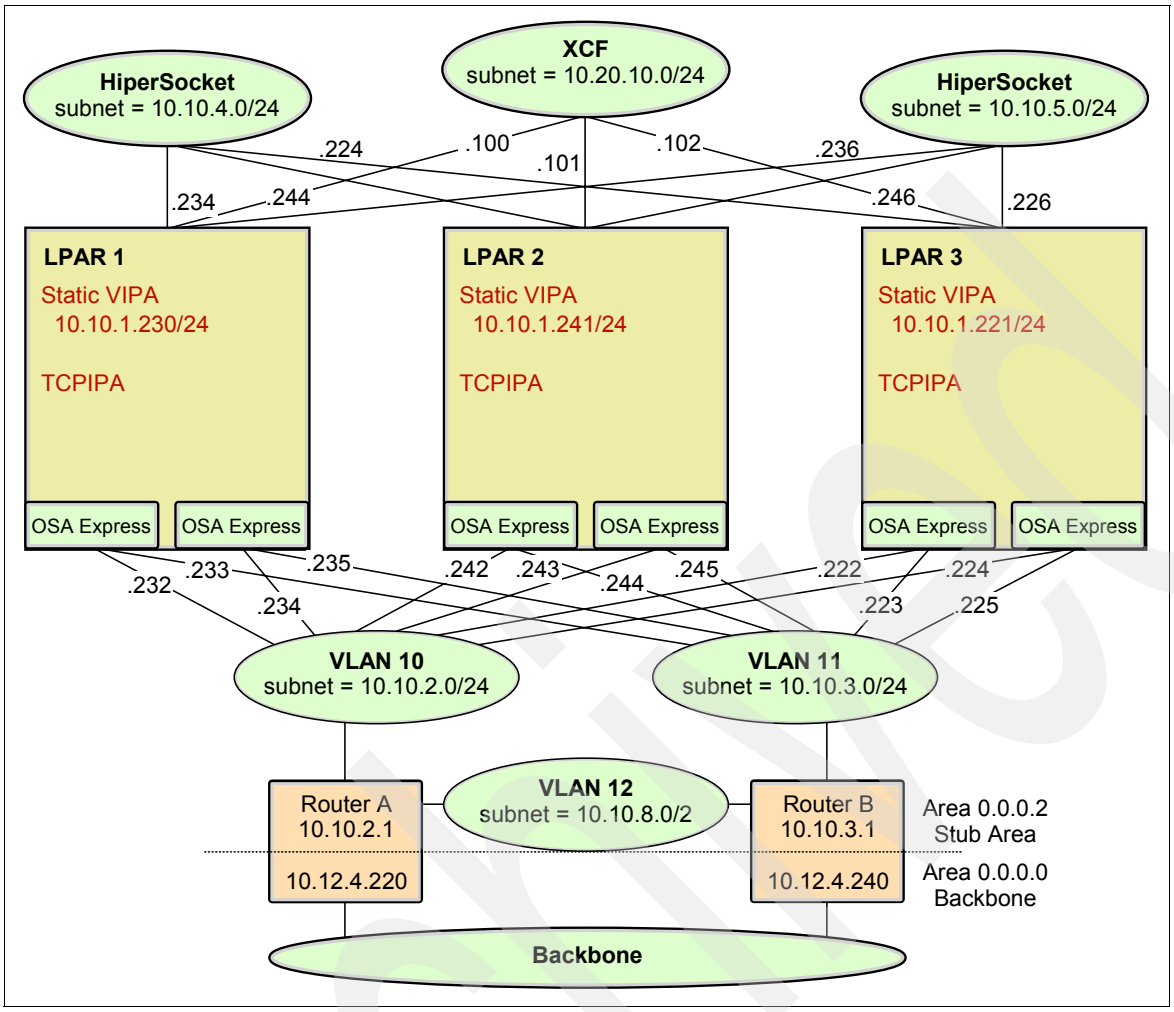


Figure 5-6 z/OS network environment with OSPF dynamic routing topology

Implementation tasks

The steps to configure OMPROUTE with OSPF are:

1. Create the OMPROUTE configuration file.
2. Change port 520 and 521 definitions to NOAUTOLOG.
3. Update the Resolver configuration file.
4. Update the OMPROUTE cataloged procedure.
5. RACF authorize user IDs for starting OMPROUTE.
6. Start syslogd.
7. Update the OMPROUTE environment variables.

   In our Cisco routers we created a router service 100, where we defined the OSPF configuration.

8. Router configuration statements

Create the OMPROUTE configuration file

In the OMPROUTE configuration file (see Example 5-7) we defined a Stub Area, the interfaces as part of the Stub Area, and other parameters.

Example 5-7 OMPROUTE configuration file

```
Area Area_Number=0.0.0.2 a
```

```
     Stub_Area=YES                                        b
     Authentication_type=None
     Import_Summaries=No;                                 c
OSPF RouterID=10.10.1.230;                                d
Routesa_Config Enabled=No;
; Static vipa
OSPF_Interface IP_address=10.10.1.230                     e
          Subnet_mask=255.255.255.252
          Name=STAVIPA1LNK                                f
          Attaches_To_Area=0.0.0.2 a
          Advertise_VIPA_Routes=HOST_ONLY                 g
          Cost0=10
          MTU=65535;
; OSA Qdio VLAN10
OSPF_Interface IP_address=10.10.2.*                       h
           Subnet_mask=255.255.255.0
           Router_Priority=0                              i
           Attaches_To_Area=0.0.0.2 a
           Cost0=10
           MTU=1492;
; OSA Qdio VLAN11
OSPF_Interface IP_address=10.10.3.*                       h
           Subnet_mask=255.255.255.0
           Router_Priority=0                              i
           Attaches_To_Area=0.0.0.2 a
           Cost0=10
           MTU=1492;
; hipersocket
OSPF_Interface IP_address=10.10.5.*                       h
           Subnet_mask=255.255.255.0
           Attaches_To_Area=0.0.0.2 a
           Cost0=5
           MTU=8192;
; xcf
OSPF_Interface IP_address=10.20.*.*                       h
           Subnet_mask=255.255.255.0
           Attaches_To_Area=0.0.0.2 a
           Cost0=15
           MTU=8192;
;
AS_Boundary_routing
  Import_Direct_Routes=yes; j
```

The descriptions for the tags shown in Example 5-7 on page 165 are as follows:

a Identifies the OSPF Area (Area 2).

b Indicates Area 2 is a Stub Area.

c We only want to receive default information.

d Defines the router internal IP address.

> **Attention:** With the advent of Dynamic VIPAs (DVIPAs) that can move between z/OS hosts within a sysplex, it is highly recommended to code the RouterID statement (d), either with the static VIPA address or an interface address.

e Defines a specific interface to be an OSPF interface (IP address). When this is used, the NAME statement must also be configured (see f).

f The NAME statement identifies the link as an OSPF interface.

g If the OSPF_Interface is a VIPA link, you can use this parameter to tell OMPROUTE how you want the VIPA address to be advertised. With this option set to HOST_ONLY, only the VIPA host route is advertised.

h When defining OSPF_Interface IP_address with a wild card (*), all interfaces within the defined range will be seen as OSPF interfaces, simplifying our OSPF configuration file.

i The z/OS Communications Server should be prevented from becoming the designated router in the LAN environment, when routers that are present can perform this function. To do this, define statement Router_Priority with value 0.

j Stub Areas do not permit importation of OSPF external, direct, or static routes; although the z/OS Communications Server learns about them, they will not be advertised.

Change port 520 and 521 definitions to NOAUTOLOG

If OMPROUTE is being started with AUTOLOG and only the OSPF protocol is being used, it is important to do one of the following:

- Ensure that the RIP UDP port (520) and the IPv6 RIP UDP port (521) are not reserved by the PORT statement in the PROFILE.TCPIP.
- Add the NOAUTOLOG parameter to the PORT statement, as shown in Example 5-8.

Example 5-8 Ports 520 and 521 defined as NOAUTOLOG

```
520 UDP OMPROUTE NOAUTOLOG  ; OMPROUTE RIPV2 port
521 UDP OMPROUTE NOAUTOLOG  ; OMPROUTE RIPV2 port
```

Important: If you fail to take one of the above actions, OMPROUTE will be periodically canceled and restarted by TCP/IP.

Update the Resolver configuration file

The Resolver configuration file contains keywords (DATASETPREFIX 2 and TCPIPjobname) used by OMPROUTE. The value assigned to TCPIPjobname 1 will be used as the name of the TCP/IP stack with which OMPROUTE establishes a connection. In our test environment we defined the TCPIP.DATA file shown in Example 5-9.

Example 5-9 TCPIP.DATA file contents

```
TCPIPJOBNAME TCPIPA             1
HOSTNAME SC30A
DOMAINORIGIN  ITSO.IBM.COM
DATASETPREFIX TCPIPA            2
MESSAGECASE MIXED
NSINTERADDR  10.12.6.7
NSPORTADDR 53
RESOLVEVIA UDP
RESOLVERTIMEOUT 10
RESOLVERUDPRETRIES 1
```

Update the OMPROUTE cataloged procedure

We created the cataloged procedure by copying the sample in SEZAINST(OMPROUTE) to our PROCLIB. We Specified OMPROUTE parameters and changed the data set names to suit our local configuration. In Our test environment we created the procedure shown in Example 5-10 on page 168.

Example 5-10 OMPROUTE cataloged procedure

```
//OMP3OA PROC STDENV=STDENV&SYSCLONE a
//OMP3OA EXEC PGM=OMPROUTE,REGION=4096K,TIME=NOLIMIT,                    .
//        PARM=('POSIX(ON) ALL31(ON)',                                   .
//            'ENVAR("_BPXK_SETIBMOPT_TRANSPORT=TCPIPA"',                .
//            '"_CEE_ENVFILE=DD:STDENV")/-t2') b
//*                                                                      .
//STDENV   DD DISP=SHR,DSN=TCPIPA.OMPROUTE.&STDENV c
//*                                                                      .
//CEEDUMP  DD SYSOUT=*,DCB=(RECFM=FB,LRECL=132,BLKSIZE=132)
```

The descriptions for the tags shown in Example 5-10 are as follows:

a We can use a common procedure for all images within the same server environment by specifying the &sysclone variable.

b We can also use the PARM statement to define our environment variable and startup parameters, such as the trace and debug levels.

c Each OMPROUTE procedure in the same server will have its own environment variables based on this DD.

Important: When using _CEE_ENVFILE with a z/OS data set, the data set must be allocated with RECFM=V. RECFM=F must not be used, because it enables padding with blanks for the environment variables.

RACF authorize user IDs for starting OMPROUTE

To reduce risk of an unauthorized user starting OMPROUTE and affecting the contents of the routing table, users who start OMPROUTE must be RACF-authorized to the entity MVS.ROUTEMGR.OMPROUTE and require a UID of zero. In our test environment we execute the command shown in Example 5-11. The ID parameter defines the procedure name being authorized to start OMPROUTE 1.

Example 5-11 RACF commands to authorize the stating of OMPROUTE

```
RDEFINE OPERCMDS (MVS.ROUTEMGR.OMPROUTE) UACC(NONE)
PERMIT MVS.ROUTEMGR.OMPROUTE ACCESS(CONTROL) CLASS(OPERCMDS) ID(OMPA)1
SETROPTS RACLIST(OPERCMDS) REFRESH
```

Start syslogd

Syslogd can and should be used to receive the specified messages from OMPROUTE. It can be configured to receive all OMPROUTE non-critical messages. To do that, the syslogd.conf file, which will be used to isolate all OMPROUTE messages, must be changed to a specific syslog z/OS UNIX file system file. Example 5-12 shows how we configure the syslogd to receive all error, warning, info, and notice messages in a syslog file 1.

Example 5-12 Syslogd configuration file

```
##********************************************************************
#*                                                                   *
#* syslog.conf - Defines the actions to be taken for the specified   *
#*               facilities/priorities by the syslogd daemon.        *
#*                                                                   *
*.OMPA.*.err /etc/syslog/syslog.ompa.err.log 1
```

Update the OMPROUTE environment variables

To define our OMPROUTE environment variables we used an stdenv file, pointed to by the STDENV DD card in our OMPROUTE procedure. The stdenv file contents we used are shown in Example 5-13.

Example 5-13 OMPROUTE environment variables

```
RESOLVER_CONFIG=//'TCPIPA.TCPPARMS(DATAA30)'
OMPROUTE_FILE=//'TCPIPA.TCPPARMS(OMPA30S)'
OMPROUTE_DEBUG_FILE=/etc/omproute/debug30a
OMPROUTE_DEBUG_FILE_CONTROL=100000,5
```

Important: If defining the STDENV file to be a z/OS data set, the data set must be allocated with RECFM=V. RECFM=F is not recommended, because RECFM=F enables padding with blanks for the environment variables.

Router configuration statements

To configure Cisco router A, we use the configuration statements shown in Example 5-14.

Example 5-14 Router A configuration statements

```
router ospf 100 a
 router-id 10.10.2.1 b
 log-adjacency-changes
 area 2 stub c
 network 10.10.0.0 0.0.255.255 area 2 d
 network 10.200.1.0 0.0.0.255 area 0 e
 default-information originate always metric-type 1 f
```

To configure Cisco router B, we use the configuration statements shown in Example 5-15.

Example 5-15 Router B configuration statements

```
router ospf 100 a
 router-id 10.10.3.1 b
 log-adjacency-changes
 area 2 stub c
 network 10.10.0.0 0.0.255.255 area 2 d
 network 10.200.1.0 0.0.0.255 area 0 e
 default-information originate always metric-type 1 f
```

The descriptions for the tags shown in Example 5-14 and Example 5-15 are as follows:

a Designates the process 100 to be an OSPF routing service

b Defines the router ID of this process (100).

c Creates an OSPF area to this process (Area 2) and defines it to be a STUB Area.

d Defines the network range designated to Area 2. All interfaces within this IP address range (10.10.0.0) will belong to Area 2.

e Defines the network range designated to the backbone Area 0. All interfaces within this IP address range (10.200.0.0) will belong to Area 0.

OMPROUTE management

You can manage OMPROUTE from a z/OS operator's console. Commands are available to perform the following:

- Starting OMPROUTE

- Stopping OMPROUTE
- Modifying OMPROUTE

Starting OMPROUTE

OMPROUTE can be started from an z/OS procedure, from the z/OS shell, or from AUTOLOG.

In our test environment we started OMPROUTE by starting the OMPROUTE start procedure shown in Example 5-16.

Example 5-16 OMPROUTE z/OS procedure

```
//OMP3OA PROC STDENV=STDENV&SYSCLONE
//OMP3OA EXEC PGM=OMPROUTE,REGION=4096K,TIME=NOLIMIT,
//        PARM=('POSIX(ON) ALL31(ON)',
//            'ENVAR("_BPXK_SETIBMOPT_TRANSPORT=TCPIPA"',
//            '"_CEE_ENVFILE=DD:STDENV")/')
//STDENV   DD DISP=SHR,DSN=TCPIPA.OMPROUTE.&STDENV
//CEEDUMP  DD SYSOUT=*,DCB=(RECFM=FB,LRECL=132,BLKSIZE=132)
```

You can use the AUTOLOG statement to start OMPROUTE automatically during TCP/IP initialization. Insert the name of the OMPROUTE start procedure in the AUTOLOG statement of the PROFILE.TCPIP data set (see Example 5-17).

Example 5-17 AUTOLOG statements

```
AUTOLOG 5
   OMPA JOBNAME ; OMPROUTE procedure name
ENDAUTOLOG
```

Stopping OMPROUTE

OMPROUTE can be stopped from the z/OS console by issuing **STOP <procname> or MODIFY <procname>,KILL.**

You can also stop OMPROUTE from a z/OS shell superuser ID by issuing issue the **kill** command to the process ID (PID) associated with OMPROUTE. To determine the PID, use one of the following methods:

- From the z/OS console, issue **D OMVS,U=*userid*** (where *userid* is the user ID that started omproute from the shell). From the resulting display, look at the PID number related to OMPROUTE, as shown in Example 5-18 (1).
- From a TSO user issue **/D OMVS,U=*userid*** at the SDSF LOG window on TSO (where *userid* is the user ID that started OMPROUTE from the shell).

Example 5-18 D OMVS,U=TCPIP

```
D OMVS,U=TCPIP
BPXO040I 14.56.39 DISPLAY OMVS 617
OMVS     000E ACTIVE          OMVS=(7A)
USER     JOBNAME  ASID        PID        PPID STATE   START      CT_SECS
TCPIP    OMPA     00EF        50397483 1 1    HS----  14.43.13   243.843
LATCHWAITPID=         0 CMD=OMPROUTE
```

From the z/OS shell issue the **ps -ef** command, as shown in Example 5-19 (1).

Example 5-19 ps -ef command in z/OS shell

```
CS03 @ SC30:/u/cs03>ps -ef
```

```
     UID        PID       PPID  C    STIME TTY       TIME CMD
BPXROOT          1          0  -   Oct 11 ?        0:14 BPXPINPR
BPXROOT   16842754          1  -   Oct 11 ?        1:07 BPXVCMT
BPXROOT   50397483          1  - 14:43:14 ?        4:10 OMPROUTE 1
```

Modifying OMPROUTE

We can use the MODIFY command to do the following:

- Display OMPROUTE information: We can use the MODIFY (F) command instead of the DISPLAY TCPIP command to display information for OMPROUTE. Both commands provide the same information and use the same statements we can see in the samples that follow:
  - **F procname,RTTABLE** command: The resulting display provides us with the same contents as though we were using **D TCPIP,*tcpipprocname*,OMP,RTTABLE**.
  - **F *procname*,OSPF,LIST ALL** command: The resulting display provides us the same contents as though we were using **D TCPIP,*tcpipprocname*,OMP,OSPF,LIST ALL**.
- Change OMPROUTE configuration: We can use the MODIFY (F) command to change some configuration statements and to start, stop, or change the level of OMPROUTE tracing and debugging as follows:
  - **F *procname*,RECONFIG** command: Used to reread the OMPROUTE configuration file, adding new OSPF_interfaces
  - **F *procname*,ROUTESA=ENABLE/DISABLE** command: To enable or disable the OMPROUTE subagent
  - **F *procname*,OSPF,WEIGHT,NAME=<if_name>,COST=<cost>** command: Changes dynamically the cost of an OSPF interface
- Stop OMPROUTE: We can use the MODIFY command to stop OMPROUTE using the KILL statement, as follows:

  ```
  F procname,KILL
  ```
- Start, stop, or change the level of OMPROUTE tracing and debugging as follows:
  - **F *procname*,TRACE=n**: For OMPROUTE tracing for initialization and IPv4 routing protocols; n can be 0–2.
  - **F *procname*,DEBUG=n**: For OMPROUTE debugging for initialization and IPv4 routing protocols; n can be 0–4.
  - **F *procname*,SADEBUG=n**: For OMPROUTE subagent debugging; n can be 0 or 1.

For further information about these commands and options, refer to *z/OS V1R7.0 Communications Server: IP System Administrator's Commands*, SC31-8781.

Verification

To verify OMPROUTE behavior and status, we can use either the DISPLAY or the MODIFY command. Any of the DISPLAY options shown here can be used with the MODIFY command, as discussed in "Modifying OMPROUTE" on page 171.

In this section we show the most useful DISPLAY commands, such as:

- D TCPIP,TCPIPA,OMP,OSPF,LIST,ALL

  Displays all OSPF configuration information (see Example 5-20).

Example 5-20 D TCPIP,TCPIPA,OMP,OSPF,LIST,ALL

```
D TCPIP,TCPIPA,OMP,OSPF,LIST,ALL
```

```
 EZZ7831I GLOBAL CONFIGURATION 684
     TRACE: 0, DEBUG: 0, SADEBUG LEVEL: 0
     STACK AFFINITY:          TCPIPA
     OSPF PROTOCOL:           ENABLED
     EXTERNAL COMPARISON:     TYPE 2
     AS BOUNDARY CAPABILITY:  ENABLED
     IMPORT EXTERNAL ROUTES:  STA DIR SUB
     ORIG. DEFAULT ROUTE:     NO
     DEMAND CIRCUITS:         ENABLED

 EZZ7832I AREA CONFIGURATION
AREA ID          AUTYPE          STUB? DEFAULT-COST IMPORT-SUMMARIES?
0.0.0.2          0=NONE           YES         1             NO
0.0.0.0          0=NONE           NO          N/A           N/A

EZZ7833I INTERFACE CONFIGURATION
IP ADDRESS      AREA             COST RTRNS TRDLY PRI HELLO  DEAD
DB_EX
10.20.10.100    0.0.0.2             5     5     1   1    10    40
40
10.20.10.100    0.0.0.2             5     5     1   1    10    40
40
10.10.5.236     0.0.0.2           100     5     1   1    10    40
40
10.10.4.235     0.0.0.2           100     5     1   1    10    40
40
10.10.4.234     0.0.0.2           100     5     1   1    10    40
40
10.10.3.235     0.0.0.2            10     5     1   0    10    40
40
10.10.2.234     0.0.0.2            10     5     1   0    10    40
10.10.3.233     0.0.0.2            10     5     1   0    10    40
40
10.10.2.232     0.0.0.2            10     5     1   0    10    40
40
10.10.1.230     0.0.0.2            10   N/A   N/A N/A   N/A   N/A
N/A

               DEMAND CIRCUIT PARAMETERS
IP ADDRESS       DONOTAGE    HELLO SUPPRESSION   POLL INTERVAL
10.20.10.100     OFF         N/A                   N/A
10.20.10.100     OFF         ALLOW                  60
10.10.5.236      OFF         N/A                   N/A
10.10.4.235      OFF         N/A                   N/A
10.10.4.234      OFF         N/A                   N/A
10.10.3.235      OFF         N/A                   N/A
10.10.2.234      OFF         N/A                   N/A
10.10.3.233      OFF         N/A                   N/A
10.10.2.232      OFF         N/A                   N/A

          ADVERTISED VIPA ROUTES
10.10.1.230    /255.255.255.255
```

- D TCPIP,TCPIPA,OMP,OSPF,LIST,AREAS

  Displays information about configured OSPF Areas (see Example 5-21).

Example 5-21 D TCPIP,TCPIPA,OMP,OSPF,LIST,AREAS

```
D TCPIP,TCPIPA,OMP,OSPF,LIST,AREAS
 EZZ7832I AREA CONFIGURATION 692
```

```
 AREA ID          AUTYPE          STUB? DEFAULT-COST IMPORT-SUMMARIES?
 0.0.0.2          0=NONE           YES         1             NO
 0.0.0.0          0=NONE           NO          N/A           N/A
```

- D TCPIP,TCPIPA,OMP,OSPF,LIST,IFS

  Displays configuration information about configured OSPF interfaces (see Example 5-22).

Example 5-22 D TCPIP,TCPIPA,OMP,OSPF,LIST,IFS

```
D TCPIP,TCPIPA,OMP,OSPF,LIST,IFS
EZZ7833I INTERFACE CONFIGURATION 694
IP ADDRESS      AREA             COST RTRNS TRDLY PRI HELLO  DEAD
DB_EX
10.20.10.100    0.0.0.2             5     5     1   1    10    40
40
10.20.10.100    0.0.0.2             5     5     1   1    10    40
40
10.10.5.236     0.0.0.2           100     5     1   1    10    40
40
10.10.4.235     0.0.0.2           100     5     1   1    10    40
40
10.10.4.234     0.0.0.2           100     5     1   1    10    40
40
10.10.3.235     0.0.0.2            10     5     1   0    10    40
40
10.10.2.234     0.0.0.2            10     5     1   0    10    40
40
10.10.3.233     0.0.0.2            10     5     1   0    10    40
40
10.10.2.232     0.0.0.2            10     5     1   0    10    40
40
10.10.1.230     0.0.0.2            10   N/A   N/A N/A   N/A   N/A
N/A

                DEMAND CIRCUIT PARAMETERS
IP ADDRESS       DONOTAGE    HELLO SUPPRESSION   POLL INTERVAL
10.20.10.100     OFF         N/A                   N/A
10.20.10.100     OFF         ALLOW                  60
10.10.5.236      OFF         N/A                   N/A
10.10.4.235      OFF         N/A                   N/A
10.10.4.234      OFF         N/A                   N/A
10.10.3.235      OFF         N/A                   N/A
10.10.2.234      OFF         N/A                   N/A
10.10.3.233      OFF         N/A                   N/A
10.10.2.232      OFF         N/A                   N/A

            ADVERTISED VIPA ROUTES
10.10.1.230    /255.255.255.255
```

- D TCPIP,TCPIPA,OMP,OSPF,LSA,LSTYPE=1,LSID=10.10.2.1,ORIG=10.10.2.1, AREAID=0.0.0.2

  Displays the contents of a single OSPF link state advertisement (see Example 5-23).

Example 5-23 D TCPIP,TCPIPA,OMP,OSPF,LSA resulting display

```
D TCPIP,TCPIPA,OMP,OSPF,LSA,LSTYPE=1,LSID=10.10.2.1,AREAID=0.0.0.2
EZZ7880I LSA DETAILS 706
        LS AGE:          1327
        LS OPTIONS:      DC (0X20)
        LS TYPE:         1
```

```
LS DESTINATION (ID): 10.10.2.1
LS ORIGINATOR:   10.10.2.1
LS SEQUENCE NO:  0X8000005A
LS CHECKSUM:     0XC988
LS LENGTH:       48
ROUTER TYPE:      (0X00)
# ROUTER IFCS:   2
        LINK ID:          10.10.8.0
         LINK ID:          10.10.8.0
         LINK DATA:        255.255.255.0
         INTERFACE TYPE:   3
                NO. OF METRICS: 0
                TOS 0 METRIC:   1
         LINK ID:          10.10.2.1
         LINK DATA:        10.10.2.1
         INTERFACE TYPE:   2
                NO. OF METRICS: 0
                TOS 0 METRIC:   1 (0)
```

- D TCPIP,TCPIPA,OMP,OSPF,AREASUM

  Displays statistics and parameters for all OSPF areas attached to the router (see Example 5-24).

Example 5-24 D TCPIP,TCPIPA,OMP,OSPF,AREASUM

```
D TCPIP,TCPIPA,OMP,OSPF,AREASUM
EZZ7848I AREA SUMMARY 708
AREA ID        AUTHENTICATION   #IFCS  #NETS  #RTRS  #BRDRS DEMAND
0.0.0.2          NONE             10      9      9      0  ON
```

- D TCPIP,TCPIPA,OMP,OSPF,EXTERNAL

  Displays a list of AS external advertisements that are in the OSPF link state database (see Example 5-25).

Example 5-25 D TCPIP,TCPIPA,OMP,OSPF,EXTERNAL

```
D TCPIP,TCPIPA,OMP,OSPF,EXTERNAL
EZZ7853I AREA LINK STATE DATABASE 710
TYPE LS DESTINATION     LS ORIGINATOR     SEQNO      AGE   XSUM
  5 @0.0.0.0            10.10.1.230     0X80000007 1321  0XE8B0
  5 @10.10.20.130       10.10.1.230     0X80000007 1321  0X4831
  5 @10.20.10.0         10.10.1.230     0X80000007 1321  0X57A4
  5 @10.20.10.101       10.10.1.230     0X80000007 1321  0X6135
  5 @10.20.10.102       10.10.1.230     0X80000007 1321  0X573E
  5 @10.20.20.102       10.10.1.230     0X80000007 1321  0XE8A2
  5 @10.20.40.100       10.10.1.230     0X80000007 1321  0X2059
  5 @10.20.40.102       10.10.1.230     0X80000007 1321  0XC6B
  5 @10.30.20.100       10.10.1.230     0X80000007 1321  0X84FE
  5 @10.30.20.101       10.10.1.230     0X80000007 1321  0X7A08
  5 @10.30.20.102       10.10.1.230     0X80000007 1321  0X7011
            # ADVERTISEMENTS:       11
            CHECKSUM TOTAL:         0X4C575
```

- D TCPIP,TCPIPA,OMP,OSPF,INTERFACE

  Displays current run-time statistics and parameters for OSPF interfaces (see Example 5-26 on page 175).

Example 5-26 D TCPIP,TCPIPA,OMP,OSPF,INTERFACE

```
D TCPIP,TCPIPA,OMP,OSPF,INTERFACE
EZZ7849I INTERFACES 712
IFC ADDRESS     PHYS         ASSOC. AREA     TYPE   STATE  #NBRS
#ADJS
10.20.10.100    IQDIOLNKOA1* 0.0.0.2         BRDCST   32      2      2
10.20.10.100    EZASAMEMVS   0.0.0.2         P-2-MP   16      3      1
10.10.5.236     IUTIQDF6LNK  0.0.0.2         BRDCST   32      2      2
10.10.4.235     IUTIQDF5LNK  0.0.0.2         BRDCST   32      2      2
10.10.4.234     IUTIQDF4LNK  0.0.0.2         BRDCST    2      0      0
10.10.3.235     OSA2OE0LNK   0.0.0.2         BRDCST   32      2      0
10.10.2.234     OSA2OC0LNK   0.0.0.2         BRDCST   32      3      1
10.10.3.233     OSA2OA0LNK   0.0.0.2         BRDCST    2      0      0
10.10.2.232     OSA2080LNK   0.0.0.2         BRDCST    2      0      0
10.10.1.230     STAVIPA1LNK  0.0.0.2         VIPA    N/A    N/A    N/A
* -- LINK NAME TRUNCATED
```

- D TCPIP,TCPIPA,OMP,OSPF,NBR

  Displays current run-time statistics and parameters for OSPF neighbors (see Example 5-27).

Example 5-27 D TCPIP,TCPIPA,OMP,OSPF,NBR

```
D TCPIP,TCPIPA,OMP,OSPF,NBR
EZZ7851I NEIGHBOR SUMMARY 714
NEIGHBOR ADDR   NEIGHBOR ID     STATE  LSRXL DBSUM LSREQ HSUP IFC
10.20.10.101    10.10.1.241       128      0     0     0  OFF
IQDIOLNK*
10.20.10.102    10.10.1.221       128      0     0     0  OFF
IQDIOLNK*
10.30.20.100    0.0.0.0             1      0     0     0  OFF
EZASAMEM*
10.20.40.100    10.40.1.230       128      0     0     0  OFF
EZASAMEM*
10.10.20.130    0.0.0.0             1      0     0     0  OFF
EZASAMEM*
10.10.5.226     10.10.1.221       128      0     0     0  OFF
IUTIQDF6*
10.10.5.246     10.10.1.241       128      0     0     0  OFF
IUTIQDF6*
10.10.4.245     10.10.1.241       128      0     0     0  OFF
IUTIQDF5*
10.10.4.225     10.10.1.221       128      0     0     0  OFF
IUTIQDF5*
10.10.3.245     10.10.1.241         8      0     0     0  OFF
OSA2OE0L*
10.10.3.225     10.10.1.221         8      0     0     0  OFF
OSA2OE0L*
10.10.2.244     10.10.1.241         8      0     0     0  OFF
OSA2OC0L*
10.10.2.1       10.10.2.1         128     12     0     0  OFF
OSA2OC0L*
10.10.2.224     10.10.1.221         8      0     0     0  OFF
OSA2OC0L*
* -- LINK NAME TRUNCATED
```

- D TCPIP,TCPIPA,OMP,OSPF,ROUTERS

  Displays routes to other routers that have been calculated by OSPF (see Example 5-28 on page 176).

Example 5-28 D TCPIP,TCPIPA,OMP,OSPF,ROUTERS

```
EZZ7855I OSPF ROUTERS 718
DTYPE RTYPE DESTINATION       AREA          COST       NEXT HOP(S)
FADD  SPF   10.10.2.1        0.0.0.2        10        OSA2080LNK
*
FADD  SPF   10.20.10.100     0.0.0.2         0        EZASAMEMVS
```

- D TCPIP,TCPIPA,OMP,RTTABLE

  Displays all of the routes in the OMPROUTE routing table (see Example 5-29).

Example 5-29 D TCPIP,TCPIPA,OMP,RTTABLE

```
D TCPIP,TCPIPA,OMP,RTTABLE
EZZ7847I ROUTING TABLE 720
TYPE   DEST NET         MASK      COST    AGE     NEXT HOP(S)

RSTA*  0.0.0.0                0  0       13323   10.10.2.1        (4)
 SPF   10.10.1.221      FFFFFFFF  15      13319   10.20.10.102
 DIR*  10.10.1.228      FFFFFFFC  1       13326   10.10.1.230
 DIR*  10.10.1.230      FFFFFFFF  1       13326   STAVIPA1LNK
 SPF   10.10.1.241      FFFFFFFF  15      13319   10.20.10.101
 SPF*  10.10.2.0        FFFFFF00  10      13314   OSA2080LNK       (2)
 DIR*  10.10.3.0        FFFFFF00  1       13324   10.10.3.233      (2)
 SPF*  10.10.4.0        FFFFFF00  100     13319   IUTIQDF4LNK      (2)
 SPF*  10.10.5.0        FFFFFF00  100     13314   IUTIQDF6LNK
 SPF   10.10.8.0        FFFFFF00  11      13314   10.10.2.1        (2)
STAT*  10.10.20.130     FFFFFFFF  0       13324   10.20.10.100
STAT*  10.20.10.0       FFFFFF00  0       13324   10.20.10.100
 SPF   10.20.10.100     FFFFFFFF  0       13326   EZASAMEMVS
STAT*  10.20.10.101     FFFFFFFF  0       13324   10.20.10.100
STAT*  10.20.10.102     FFFFFFFF  0       13324   10.20.10.100
STAT*  10.20.20.102     FFFFFFFF  0       13324   10.20.10.100
 SPF   10.20.40.0       FFFFFF00  10      13319   10.20.40.100
STAT*  10.20.40.100     FFFFFFFF  0       13324   10.20.10.100
STAT*  10.20.40.102     FFFFFFFF  0       13324   10.20.10.100
STAT*  10.30.20.100     FFFFFFFF  0       13324   10.20.10.100
STAT*  10.30.20.101     FFFFFFFF  0       13324   10.20.10.100
STAT*  10.30.20.102     FFFFFFFF  0       13324   10.20.10.100
 SPF   10.40.1.221      FFFFFFFF  20      13319   10.20.10.102     (2)
 SPF   10.40.1.230      FFFFFFFF  15      13319   10.20.40.100
 SPF   10.40.1.241      FFFFFFFF  25      13319   10.20.40.100
 SPF   10.40.2.0        FFFFFF00  15      13319   10.20.40.100
 SPF   10.40.3.0        FFFFFF00  15      13319   10.20.40.100
 SPF   10.40.4.0        FFFFFF00  105     13319   10.20.40.100
 SPF   10.40.5.0        FFFFFF00  105     13319   10.20.40.100

DEFAULT GATEWAY IN USE.

TYPE COST     AGE       NEXT HOP
RSTA 0        13323     10.10.2.1        (4)
                      0 NETS DELETED, 1 NETS INACTIVE
```

To see other display command options and to get more detailed information about specific commands, refer to *z/OS V1R7.0 Communications Server: IP System Administrator's Commands*, SC31-8781.

5.5.3 Problem determination

When implementing a network environment with indirect access to external hosts or networks using routing definitions, it is important to understand how to isolate networking problems. This means that using the correct diagnostic tools and techniques is essential. In this section we describe the tools and techniques needed to debug routing problems, including OMPROUTE (OSPF) problems.

To debug a network problem in a z/OS environment we suggest to follow the flow shown in Figure 5-7.

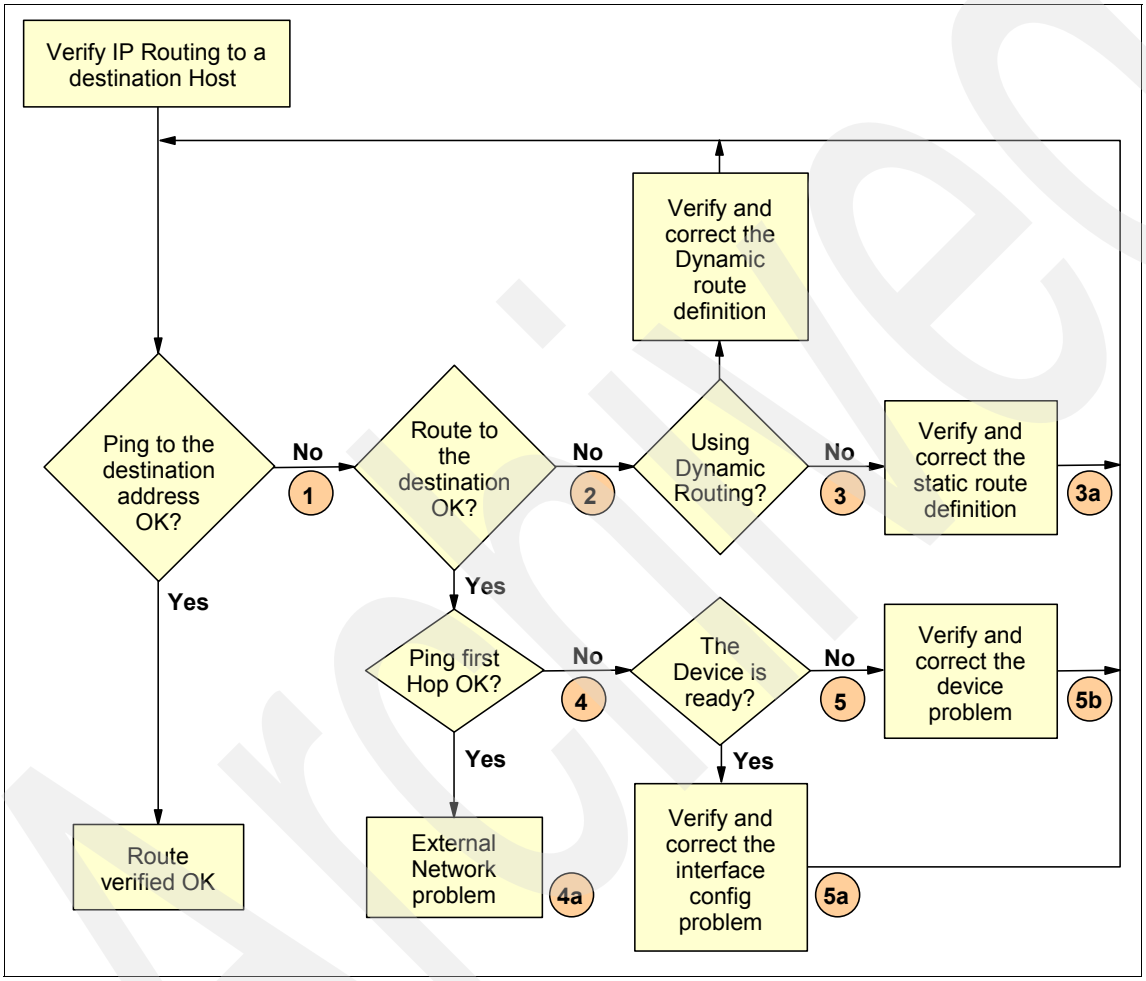


Figure 5-7 Routing problem determination flow

The descriptions for the tags shown in Figure 5-7 are as follows:

1. Use the Ping command to determine whether there is connectivity to the destination IP address. More details about the Ping command can be found in “PING command (TSO or z/OS Unix)” on page 178.
2. If the Ping command fails immediately, there might not be a route to the destination host or subnet. Use the **`Netstat ROUTE/ -r`** command to display routes to the network, as shown in “Netstat command (TSO or z/OS Unix)” on page 178. Verify that TCP/IP has a route to the destination address. If there is no route, proceed to step 3. If a route exists, proceed to step 4.

3. If there is no route to the destination, problem resolution depends on whether static or dynamic routing is being used. In either case, do the following:
   a. If TCP/IP is configured using Static Routing, review and correct the configuration.
   b. If OMPROUTE is being used to generate dynamic routes, verify and correct the configuration and, if it seems OK, diagnose the problem using the debugging tools described in “Diagnosing a OMPROUTE problem” on page 180.
4. If a route exists, verify that the route is correct for the destination. Determine whether the gateway identified for the route to the destination is reachable. To verify this, use the PING command to confirm connectivity to the gateway. Do one of the following:
   a. If the gateway responds to a ping, then there is a network problem at the gateway or beyond. To get further debug information, use the **traceroute** command with the final destination address to determine which hop in the route is failing.
   b. If the gateway does not respond to a ping, proceed to step 5.
5. Determine which network interface is associated with the route to the destination. Verify it is operational issuing the **Netstat Devlink** command, as shown in “Netstat command (TSO or z/OS Unix)” on page 178. Based on the resulting display, do one of the following:
   a. If the device is ready, the problem might be in the interface configuration. Check the network configuration (VLAN ID, IP address, subnet mask, and so on). Correct this and resume testing. Otherwise, a packet trace should be taken to verify that the packets are being sent to the network. A LAN Analyzer could also be used to verify the network traffic in the switch port where the OSA-Express port is connected.
   b. If the device is not ready, the problem might be that the device is not varied online to z/OS, or there is an error in the device configuration. Also verify the VTAM TRLE definitions, HCD/IOCP configuration, as well as the physical connection, cable, and switch port.

Commands to diagnose networking connectivity problems

In this section we describe briefly the commands that can be used to diagnose connectivity problems. For additional help and information about diagnosing problems, refer to *z/OS V1R7.0 Communications Server: IP System Administrator’s Commands*, SC31-8781.

PING command (TSO or z/OS Unix)

We ping a host in a remote network to verify:

- Whether the route to the remote network is defined correctly
- Whether the router is able to forward packets to the remote network
- Whether the remote host is able to send and receive packets on the network
- Whether the remote host has a route back to the local host

Tip: The use of names instead of IP address would need the Resolver or DNS to do the translation, adding more variables to the problem determination, and should be avoided when diagnosing network problems. Use the host IP address instead.

Netstat command (TSO or z/OS Unix)

We can use the **Netstat** command to verify the status of the TCP/IP configuration. To verify our routing-related configuration, we use the following **Netstat** command options:

- **D TCPIP,*tcpipproc*,NETSTAT, DEVLINK**: Use this command to display the status and associated configuration values for a device and its defined interfaces, as shown in Example 5-30 on page 179.

Example 5-30 D TCPIP,tcpipproc,NETSTAT, DEVLINK

```
D TCPIP,TCPIPA,N,DEVLINK
EZD0101I NETSTAT CS V1R7 TCPIPA 331
LNKNAME: OSA2080LNK        LNKTYPE: IPAQENET   LNKSTATUS: NOT ACTIVE
     NETNUM: N/A  QUESIZE: N/A
     IPBROADCASTCAPABILITY: NO
     CFGROUTER: NON                    ACTROUTER: UNKNOWN
     ACTMTU: UNKNOWN
     SECCLASS: 255
   BSD ROUTING PARAMETERS:
     MTU SIZE: 1492              METRIC: 10
     DESTADDR: 0.0.0.0           SUBNETMASK: 255.255.255.0
   MULTICAST SPECIFIC:
     MULTICAST CAPABILITY: UNKNOWN
     GROUP             REFCNT
     -----             ------
     224.0.0.1         0000000001
   LINK STATISTICS:
     BYTESIN                          = 13997061
     INBOUND PACKETS                  = 79260
     INBOUND PACKETS IN ERROR         = 150310
     INBOUND PACKETS DISCARDED        = 0
     INBOUND PACKETS WITH NO PROTOCOL = 0
     BYTESOUT                         = 2092
     OUTBOUND PACKETS IN ERROR        = 4
     OUTBOUND PACKETS DISCARDED       = 0
```

- **D TCPIP,*tcpipproc*,NETSTAT,ROUTE**: This command displays the current routing tables for TCP/IP, as shown in Example 5-31.

Example 5-31 D TCPIP,tcpipproc,NETSTAT, ROUTE

```
D TCPIP,TCPIPA,N,ROUTE
EZD0101I NETSTAT CS V1R7 TCPIPA 336
IPV4 DESTINATIONS
DESTINATION        GATEWAY         FLAGS     REFCNT  INTERFACE
DEFAULT            10.10.2.1       GZ        000000  OSA2080LNK
DEFAULT            10.10.3.1       UGZ       000000  OSA20A0LNK
DEFAULT            10.10.2.2       GZ        000000  OSA20C0LNK
DEFAULT            10.10.3.2       UGZ       000000  OSA20E0LNK
10.10.1.230/32     0.0.0.0         UH        000000  STAVIPA1LNK
10.10.1.241/32     10.20.10.101    UGHO      000000  IQDIOLNK0A140A64
10.10.2.0/24       0.0.0.0         Z         000000  OSA2080LNK
10.10.2.0/24       0.0.0.0         Z         000000  OSA20C0LNK
10.10.2.232/32     0.0.0.0         H         000000  OSA2080LNK
10.10.2.234/32     0.0.0.0         H         000000  OSA20C0LNK
10.10.3.0/24       0.0.0.0         UC        000000  OSA20E0LNK
10.10.3.0/24       0.0.0.0         UC        000000  OSA20A0LNK
10.10.3.233/32     0.0.0.0         UH        000000  OSA20A0LNK
10.10.3.235/32     0.0.0.0         UH        000000  OSA20E0LNK
10.10.4.0/24       0.0.0.0         UO        000000  IUTIQDF5LNK
10.10.4.0/24       0.0.0.0         UO        000000  IUTIQDF4LNK
```

TRACEROUTE command

Traceroute can be invoked by either the TSO TRACERTE command or the z/OS UNIX shell **traceroute/otracert** command.

Traceroute displays the route that a packet takes to reach the requested target. Traceroute starts at the first router and uses a series of UDP probe packets with increasing IP time-to-live (TTL) or hop count values to determine the sequence of routers that must be traversed to reach the target host. The output generated by this command can be seen in Example 5-32.

Example 5-32 Tracerte command results

```
CS V1R7: Traceroute to 10.12.4.221 (10.12.4.221):
1 router1 (10.10.2.1)  0 ms  0 ms  0 ms
2 tot177 (10.12.4.221)  0 ms  0 ms  0 ms
```

Diagnosing a OMPROUTE problem

To diagnose an OMPROUTE problem the first thing we should do is to verify whether we have any error messages related to OMPROUTE during the startup process. To do so, examine SYSLOGD, JES MSG output and the console log for errors.

If there is no apparent error message that could help us to solve the problem, the next thing to do is to prepare OMPROUTE to generate more detailed information, using the debug tools available in OMPROUTE, which can be activated by coding the Debug and Trace options in the startup procedure or by using the MODIFY command to implement these options.

A trace from startup is ideal because some information is only shown in the trace at startup and because the time for problem determination and resolution is quicker when the trace captures the entire flow of events rather than just a small subset of events.

An OMPROUTE trace from startup can be enabled by coding the trace options after the forward slash in the PARM field of the OMPROUTE catalogued procedure, as shown in Example 5-33.

Example 5-33 Trace options defined in the OMPROUTE startup procedure

```
//OMP30A PROC STDENV=STDENV&SYSCLONE
//OMP30A EXEC PGM=OMPROUTE,REGION=4096K,TIME=NOLIMIT,
//        PARM=('POSIX(ON) ALL31(ON)',
//             'ENVAR("_BPXK_SETIBMOPT_TRANSPORT=TCPIPA"',
//             '"_CEE_ENVFILE=DD:STDENV")/-t2 -d1')
//*
//STDENV   DD DISP=SHR,DSN=TCPIPA.OMPROUTE.&STDENV
```

If a trace cannot be enabled from startup, then the following commands can dynamically enable and disable tracing:

- To enable:
  - /F omproute,TRACE=2 (TRACE6=2 for IPv6)
  - /F omproute,DEBUG=1 (DEBUG6=1 for IPv6)
- To disable:
  - /F omproute,TRACE=0 (TRACE6=0 for IPv6)
  - /F omproute,DEBUG=0 (DEBUG6=0 for IPv6)

Trace output is sent to one of the following locations:

- Destination referenced by OMPROUTE_DEBUG_FILE environment variable (which is coded in the STDENV DD dataset).
- STDOUT DD, but trace output is only output to this location if OMPROUTE_DEBUG_FILE is *not* defined, and the trace is started at initialization.
- /{TMPDIR}/omproute_debug (TMPDIR is usually /tmp.)

By default OMPROUTE will create five debug files, each 200 KB in size, for a total of 1 MB worth of trace data. The size and number of trace files can be controlled with the OMPROUTE_DEBUG_CONTROL environment variable. For further information about the TRACE and DEBUG options, refer to *z/OS V1R7.0 Communications Server: IP Diagnosis Guide*, GC31-8782.

Important: Using the OMPROUTE Trace and Debug options and directing the output to z/OS UNIX file system files generates additional overhead that may cause OSPF adjacency failures or other routing problems. To prevent that, change the output destination to the CTRACE Facility.

Using OMPROUTE CTRACE to get debugging information

As mentioned, the overhead problems that can happen when using z/OS UNIX file system files to save the trace and debug output data can be solved by using the CTRACE facility. To do so, we highly recommend the use of a new OMPROUTE option (DEBUGTRC) in the startup procedure, which changes the output destination of the OMPROUTE trace. In this section we briefly describe how to define and use CTRACE to debug OMPROUTE problems.

The OMPROUTE CTRACE can be started anytime you need by using the command TRACE CT, or it can be activated during OMPROUTE initialization. If not defined, OMPROUTE component trace is started with a buffer size of 1 MB and the MINIMUM tracing option.

A parmlib member can be used to customize the parameters and to initialize the trace. The default OMPROUTE Component Trace parmlib member is the SYS1.PARMLIB member CTIORA00. The parmlib member name can be changed by using the OMPROUTE_CTRACE_MEMBER environment variable.

Besides specifying the trace options, you can also change the OMPROUTE trace buffer size. The buffer size can be changed only at OMPROUTE initialization. The maximum OMPROUTE trace buffer size is 100 MB. The OMPROUTE REGION size in the OMPROUTE catalog procedure must be large enough to accommodate a large buffer size.

When OMPROUTE is initialized using the DEBUGTRC option, we recommend that you use a larger internal CTRACE buffer or an external writer. When using the internal CTRACE buffer, we must get a DUMP of OMPROUTE in order to see the trace output.

We are going to show next the necessary steps to start the CTRACE for OMPROUTE, directing the trace output to an external writer:

1. Create a CTWTR procedure in your SYS1.PROCLIB, as shown in Example 5-34.

Example 5-34 CTWTR procedure

```
//CTWTR    PROC
//IEFPROC  EXEC PGM=ITTTRCWR
//TRCOUT01 DD  DSNAME=CSO3.CTRACE1,VOL=SER=COMST2,UNIT=3390,
//         SPACE=(CYL,10),DISP=(NEW,KEEP),DSORG=PS
//TRCOUT02 DD  DSNAME=CSO3.CTRACE2,VOL=SER=COMST2,UNIT=3390,
//         SPACE=(CYL,10),DISP=(NEW,KEEP),DSORG=PS
//*
```

2. Prepare the SYS1.PARMLIB member CTIORA00 to get the desired output data. Example 5-35 shows a sample of CTIORA00 contents.

Example 5-35 CTIORA00 sample

```
/*****************************************************************/
/*                                                               */
```

```
/*  DESCRIPTION = This parmlib member causes component trace for     */
/*                the TCP/IP OMPROUTE application to be initialized  */
/*                with a trace buffer size of 1M                     */
/*                                                                   */
/*                This parmlib member only lists those TRACEOPTS     */
/*                values specific to OMPROUTE.  For a complete list  */
/*                of TRACEOPTS keywords and their values see         */
/*                z/OS MVS INITIALIZATION AND TUNING REFERENCE.      */
/*                                                                   */
/*                                                                   */
/* $MAC(CTIORA00),COMP(OSPF  ),PROD(TCPIP ):  Component Trace        */
/*                                            SYS1.PARMLIB member    */
/*                                                                   */
/*********************************************************************/
TRACEOPTS
/* ------------------------------------------------------------------ */
/*   Optionally start external writer in this file (use both          */
/*   WTRSTART and WTR with same wtr_procedure)                        */
/* ------------------------------------------------------------------ */
/*        WTRSTART(CTWTR)                 a                           */
/* ------------------------------------------------------------------ */
/*   ON OR OFF: PICK 1                                                */
/* ------------------------------------------------------------------ */
         ON
/*       OFF                                                         */
/* ------------------------------------------------------------------ */
/*   BUFSIZE: A VALUE IN RANGE 128K TO 100M                           */
/*            CTRACE buffers reside in OMPROUTE Private storage       */
/*            which is in the regions address space.                  */
/* ------------------------------------------------------------------ */
         BUFSIZE(50M)                    b
         WTR(CTWTR)                      a
/* ------------------------------------------------------------------ */
/*   OPTIONS: NAMES OF FUNCTIONS TO BE TRACED, OR "ALL"               */
/* ------------------------------------------------------------------ */
/*        OPTIONS(                        c                           */
/*                'ALL     '                                          */
/*                ,'MINIMUM '                                         */
/*                ,'ROUTE '                                           */
/*                ,'PACKET '                                          */
/*                ,'OPACKET '                                         */
/*                ,'RPACKET '                                         */
/*                ,'IPACKET '                                         */
/*                ,'SPACKET '                                         */
                  ,'DEBUGTRC'           d
/*                              )                                    */
```

The descriptions for the tags shown in Example 5-35 on page 181 are as follows:

a Define whether we are going to use an external writer to save the output trace data.

b Define the CTRACE buffer size allocated in the OMPROUTE private storage.

c Define the trace options to be used to get specific debug information. MINIMUM is the default option.

d This option indicates the output data generated from Debug and Trace options activated in the OMPROUTE startup procedure will be sent to CTRACE.

3. Start the OMPROUTE procedure using the desired Debug and Trace options as shown in Example 5-36 on page 183.

Example 5-36 OMPROUTE procedure

```
//OMP3OA PROC STDENV=STDENV&SYSCLONE
//OMP3OA EXEC PGM=OMPROUTE,REGION=4096K,TIME=NOLIMIT,
//        PARM=('POSIX(ON) ALL31(ON)',
//            'ENVAR("_BPXK_SETIBMOPT_TRANSPORT=TCPIPA"',
//            '"_CEE_ENVFILE=DD:STDENV")/-t2 -d1')        a
//*
//STDENV   DD DISP=SHR,DSN=TCPIPA.OMPROUTE.&STDENV
```

The description for the tag shown in Example 5-36 is as follows:

a The parameters -t (trace) and -d (debug) define how detailed we want the output data to be. We recommend using -t2 and -d1.

To verify whether CTRACE has been started as expected, display the CTRACE status, issuing the console command shown in Example 5-37.

Example 5-37 Displaying OMPROUTE CTRACE status

```
D TRACE,COMP=SYSTCPRT,SUB=(OMPA)
IEE843I 17.01.01  TRACE DISPLAY 820
        SYSTEM STATUS INFORMATION
 ST=(ON,0256K,00512K) AS=ON  BR=OFF EX=ON  MT=(ON,024K)
  TRACENAME
  =========
  SYSTCPRT
                      MODE BUFFER HEAD SUBS
                      ====================
                      OFF         HEAD    3
     NO HEAD OPTIONS
  SUBTRACE            MODE BUFFER HEAD SUBS
 ----------------------------------------------------------------
  OMPA                ON   0010M
     ASIDS      *NONE*
     JOBNAMES   *NONE*
     OPTIONS    MINIMUM ,DEBUGTRC
     WRITER     CTWTR
```

4. We can also use TRACE CT command to define the options we want after OMPROUTE has been initialized, as seen in Example 5-38.

Example 5-38 TRACE CT command flow

```
  TRACE CT,ON,COMP=SYSTCPRE,SUB=(RESOLV32)
 *189 ITT006A SPECIFY OPERAND(S) FOR TRACE CT COMMAND.
  R 189,OPTIONS=(ALL),END
  IEE600I REPLY TO 189 IS;OPTIONS=(ALL),END
  ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
  WERE SUCCESSFULLY EXECUTED.
  IEE839I ST=(ON,0256K,00512K) AS=ON  BR=OFF EX=ON  MT=(ON,024K) 497
          ISSUE DISPLAY TRACE CMD FOR SYSTEM AND COMPONENT TRACE STATUS
          ISSUE DISPLAY TRACE,TT CMD FOR TRANSACTION TRACE STATUS
```

5. Reproduce the problem.
6. Stop the CTRACE by issuing the command in Example 5-39.

Example 5-39 Stopping CTRACE

```
  TRACE CT,OFF,COMP=SYSTCPRE,SUB=(RESOLV32)
  ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
```

```
  WERE SUCCESSFULLY EXECUTED.
  IEE839I ST=(ON,0256K,00512K) AS=ON  BR=OFF EX=ON  MT=(ON,024K) 506
          ISSUE DISPLAY TRACE CMD FOR SYSTEM AND COMPONENT TRACE STATUS
          ISSUE DISPLAY TRACE,TT CMD FOR TRANSACTION TRACE STATUS
```

7. Save the trace contents into the trace file created by the CTWRT procedure, by executing the command in Example 5-40.

Example 5-40 Saving the trace contents

```
 TRACE CT,ON,COMP=SYSTCPRT,SUB=(OMPA)
*018 ITT006A SPECIFY OPERAND(S) FOR TRACE CT COMMAND.
 R 18,WTR=DISCONNECT,END
 IEE600I REPLY TO 018 IS;WTR=DISCONNECT,END
 ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
 WERE SUCCESSFULLY EXECUTED.
 IEE839I ST=(ON,0256K,00512K) AS=ON  BR=OFF EX=ON  MT=(ON,024K) 828
         ISSUE DISPLAY TRACE CMD FOR SYSTEM AND COMPONENT TRACE STATUS
         ISSUE DISPLAY TRACE,TT CMD FOR TRANSACTION TRACE STATUS
```

8. Stop the external writer procedure CTWTR by issuing the command shown in Example 5-41.

Example 5-41 Stopping CTWTR

```
TRACE CT,WTRSTOP=CTWTR
ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
WERE SUCCESSFULLY EXECUTED.
IEF196I AHL904I THE FOLLOWING TRACE DATASETS CONTAIN TRACE DATA :
IEF196I           CS03.CTRACE1
IEF196I           CS03.CTRACE2
IEE839I ST=(ON,0256K,00512K) AS=ON  BR=OFF EX=ON  MT=(ON,024K) 868
        ISSUE DISPLAY TRACE CMD FOR SYSTEM AND COMPONENT TRACE STATUS
        ISSUE DISPLAY TRACE,TT CMD FOR TRANSACTION TRACE STATUS
AHL904I THE FOLLOWING TRACE DATASETS CONTAIN TRACE DATA : 864
          CS03.CTRACE1
          CS03.CTRACE2
ITT111I CTRACE WRITER CTWTR TERMINATED BECAUSE OF A WTRSTOP REQUEST.
IEF196I IEF142I CTWTR CTWTR - STEP WAS EXECUTED - COND CODE 0000
IEF196I IEF285I   CS03.CTRACE1                                KEPT
IEF196I IEF285I   VOL SER NOS= COMST2.
IEF196I IEF285I   CS03.CTRACE2                                KEPT
IEF196I IEF285I   VOL SER NOS= COMST2.
```

9. Change the OMPROUTE Debug and Trace level to avoid performance problems using the MODIFY command, as shown in Example 5-42.

Example 5-42 Modifying Debug and Trace level

```
F OMPA,TRACE=0
EZZ7866I OMPROUTE MODIFY COMMAND ACCEPTED
F OMPA,DEBUG=0
EZZ7866I OMPROUTE MODIFY COMMAND ACCEPTED     g
```

After these steps, the trace file must be formatted, using the following IPCS command, into the IPCS Subcommand screen (option 6), as shown in Example 5-43 on page 185.

Example 5-43 Formatting the OMPROUTE CTRACE

```
CTRACE COMP(SYSTCPRT) FULL
```

The resulting display will show the Resolver process entries, as shown in Example 5-44.

Example 5-44 Sample of formatted OMPROUTE CTRACE

```
 SC30      DEBUGTRC  00060001  21:39:23.679072   Start Trace Message
EZZ7838I Using configuration file: //'TCPIPA.TCPPARMS(OMPA30S)'
================================================================================0000000F
 SC30      DEBUGTRC  00060001  21:39:23.680208   Start Trace Message
11/23 21:39:23 post-checking on OSPF interface 10.10.1.230(STAVIPA1LNK)
================================================================================00000010
 SC30      DEBUGTRC  00060001  21:39:23.680254   Start Trace Message
11/23 21:39:23 post-checking on OSPF interface 10.10.2.0(NO NAME)
================================================================================00000011
 SC30      DEBUGTRC  00060001  21:39:23.680299   Start Trace Message
11/23 21:39:23 post-checking on OSPF interface 10.10.3.0(NO NAME)
================================================================================00000012
 SC30      DEBUGTRC  00060001  21:39:23.680343   Start Trace Message
11/23 21:39:23 post-checking on OSPF interface 10.10.4.0(NO NAME)
================================================================================00000013
 SC30      DEBUGTRC  00060001  21:39:23.680382   Start Trace Message
11/23 21:39:23 post-checking on OSPF interface 10.10.5.0(NO NAME)
================================================================================00000014
 SC30      DEBUGTRC  00060001  21:39:23.680421   Start Trace Message
11/23 21:39:23 post-checking on OSPF interface 10.20.0.0(NO NAME)
```

To get more information about OMPROUTE diagnosis, refer to *z/OS V1R7.0 Communications Server: IP Diagnosis Guide*, GC31-8782.

6

IPv6 support

Within the past few years IPv6 has gained much greater acceptance in the industry. What makes IPv6 so attractive is that it resolves some of the key deficiencies of IPv4 in the areas of:

- IP address space
- Auto-configuration
- Security
- Quality of service
- Anycast and multicast

This chapter discusses the following.

Section	Topic
6.1, “Overview IPv6” on page 188	Discusses the basic concepts of IPv6
6.2, “Why IPv6 is important” on page 188	Discusses key characteristics of IPv6 and why it may be important in your environment
6.3, “The common design scenarios for IPv6” on page 188	Presents commonly implemented IPv6 design scenarios, their dependencies, advantages, considerations, and our recommendations
6.4, “How IPv6 is implemented in z/OS Communications Server” on page 191	Presents selected implementation scenarios, tasks, configuration examples, and problem determination suggestions

6.1 Overview IPv6

Internet Protocol version 6 (IPv6) is the next generation of the Internet protocol designed to replace the current Internet Protocol version 4 (IPv4).

IPv6 was created to resolve the impending problems due to the shortcomings of IPv4 and the rapid demands for IP resources and functionality. The most significant issue is the diminishing supply and expected shortages of IPv4 addresses. Using IPv4 32-bit addressing allows for over 4 million nodes, each with a globally unique address. This current IPv4 space will be unable to satisfy the huge expected increase in the number of users on the Internet. The expected shortage will be exacerbated by the requirements of emerging technologies such as PDAs, HomeArea Networks, Internet-connected commodities such as the automobile and integrated telephone services. IPv6 uses 128-bit addressing and will generate a space large enough to last for the forseeable future, although that expression could have been made more than a decade ago about IPv4.

6.2 Why IPv6 is important

IPv6 is important because it addresses shortcomings of IPv4, such as:

- 128-bit addressing

  This quadruples the network address bits from 32 to 128 thereby significantly increasing the number of possible unique IP addresses to comfortably accommodate on the internet. This huge address space obviates the need for private addresses and Network Address Translators (NATs)

- Simplified Header formats

  This allows for more efficient packet handling and reduced bandwidth cost

- Hierarchical addressing and routing

  This keeps routing tables small and backbone routing efficient by using address prefixes rather than address classes

- Improved support for options

  This changes the way IP header options are encoded, allowing more efficient forwarding and greater flexibility

- Address auto-configuration

  This allows stateless IP address configuration without a configuration server

- Security

  IPv6 brings greater authentication and privacy capabilities through the definition of extensions

- Quality of Service (QoS)

  QoS is provided through a traffic class byte in the header

6.3 The common design scenarios for IPv6

Although there are predictable improvements over IPv4 as explained above, the success of any IPv6 implementation depends on the ability to have IPv6 coexist with IPv4. Because of the pervasiveness of IPv4 this coexistence will be around for sometime. Therefore, the development of technologies and mechanisms to facilitate coexistence is as important as the

deployment strategy for IPv6. In the following sections we will discuss some of the *coexistence* technologies that is available today.

6.3.1 Tunneling

Tunneling is the transmission of IPv6 traffic encapsulated within IPv4 packets over an IPv4 connection. They are used primarily to connect remote IPv6 networks or just connecting an IPv6 network over an IPv4 network infrastructure.

Dependencies

All Tunnel mechanisms require that the endpoints of the tunnel run in dual-stack mode. A dual-stack router is a router running both versions of IP. There are other dependencies based on the tunneling mechanism used. For example, an IPv6 *Manually* configured tunnel requires an ISP-registered IP address, while the *Automatic* tunnel mechanism requires IPv6 prefixes. ISATAP tunnels only require a dual-stack router, but it is not yet commercially available and *6over4* tunnels are not supported by vendor router software.

Advantages

Tunneling allows the implementation of IPv6 without any significant upgrades to the existing infrastructure and therefore does not risk interrupting the existing services provided by the IPv4 network.

Considerations

There are various tunneling mechanisms designed to do primarily different things and careful consideration must be given to the mechanism chosen. Some are primarily used for stable and secure links for regular communications while others are primarily used for single hosts or small sites, with low data traffic volumes.

6.3.2 Dedicated data links

Network architects may choose a separate ATM or frame relay PVCs or separate optical media to run IPv6 traffic across. The only requirement is the reconfiguration of the routers (with IPv6 support). These links can only be used for IPv6 traffic.

Dependencies

Dual-stack routers with IPv6 and IPv4 addresses are required to provide access to the WAN. Access to a DNS to resolve IPv6 names/addresses.

Advantages

The use of existing Layer 2 infrastructure makes this implementation less complex and immediate. This implementation is not disruptive apart from a schedule change for router configuration and therefore little impact to the status quo.

Considerations

All routers on the WAN need to support IPv6 over dedicated data links. Additional costs for the those links will be incurred until the environment is completely migrated over to IPv6.

6.3.3 MPLS backbones

An MPLS IPv4 core network may enable IPv6 domains to communicate over MPLS backbones. It is therefore primarily used by enterprises and service providers. There are various implementations of this strategy ranging from no changes and no impact to changes

and risks. This means, the closer the IPv6 implementation is to the client edge, the least expensive it becomes. While the closer the IPv6 implementation is to the service provider edge, the more expensive it becomes.

Dependencies

The dependencies vary from router configuration to specific hardware requirements to software upgrades depending on the service provider solution.

Advantages

Requires little modifications to the infrastructure and little reconfigurations of the core routers. It is therefore a strategy that could have little or no impact to your environment with low costs and low risks.

Considerations

The considerations also vary depending on the strategy chosen. For example, using the Circuit Transport over MPLS strategy does not support a mix of IPv4 and IPv6 traffic, while IPv6 on service provider edge routers do not support VPNs or VRF currently.

6.3.4 Dual-stack backbones

A dual-stack backbone is a core network with all routers configured to support dual-stacks. It is essentially two network types existing side by side. The IPv4 stack routes the IPv4 traffic through the IPv4 network, while the IPv6 stack routes the IPv6 traffic through the IPv6 network. This is a very basic approach to routing both IPv4 and IPv6 traffic through a network.

Dependencies

Each site has the appropriate entries in a DNS to resolve both IPv4 and IPv6 names and IP addresses.

Advantages

This is a basic and simple strategy for routing IPv4 and IPv6 traffic in a network.

Considerations

All routers in the network require software upgrade to support dual-stack. Having dual-stack requires that everything from router management of a dual addressing scheme to router memory being doubled (theoretically).

6.3.5 Dual-mode stack

A dual-mode stack is a stack configured to support both IPv4 and IPv6 protocols. It is a single stack (not two) configured to support IPv4 and IPv6 simultaneously. Both IPv4 and IPv6 interfaces are capable of receiving and sending IPv4 and IPv6 packets over corresponding interfaces.

Dependencies

z/OS Communications Server V1R4 or later that is configured to support IPv6 requires OSA-Express ports to be running in QDIO mode.

Advantages

There are no additional software or hardware requirements for users in a z/OS environment configured with OSA-Express features. Dual-mode allows IPv4 and IPv6 applications to

coexist indefinitely. However, any application may be migrated one at a time or at the user's convenience from IPv4 to IPv6. This is therefore an inexpensive, low risk, low impact deployment strategy.

Considerations

The only link layer protocol that supports IPv6 is MPC+. The devices that use the MPC+ protocol are XCF, MPCPTP, and MPCIPA (for example, OSA-Express in QDIO mode and HiperSockets on the System z9).

6.3.6 Recommendations

Dual-mode stacks is the recommended strategy for application migration from IPv4 to IPv6. Dual-mode is implemented in z/OS Communication Server V1R4 or later.

6.4 How IPv6 is implemented in z/OS Communications Server

IPv6 is implemented in the z/OS Communications Server through a series of configuration tasks. We configure the stack to support IPv6 in a similar fashion to the steps performed for IPv4 configuration. However, before we start to configure the stack to support IPv6 traffic we need to understand a few things about IPv6.

6.4.1 IPv6 addressing

An IPv6 address is a 128-bit number written in colon hexadecimal notation. This scheme is hexadecimal and consists of eight 16-bit pieces of the address.

Alternate notations described in RFC 2373 are acceptable; for example:

```
FEDC:BA98:7654:3210:FEDC:BA98:7654:3210
There are three conventional forms for representing IPv6 addresses as text strings:
```

- The preferred form is *xxxxxxxx*, where the *x*'s indicate the hexadecimal value 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: 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 item 2.

- Due to some methods of allocating certain styles of IPv6 addresses, it is common for addresses to contain long strings of zero bits. To simplify the writing of addresses that contain zero bits, a special syntax is available to compress the zeros. The use of **::** indicates multiple groups of 16 bits of zeros. The **::** can appear only once in an address. The **::** can also be used to compress the leading or trailing zeros in an address.

  Consider the following addresses:

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

  They can be represented as:

  ```
  1080::8:800:200C:417A (unicast address)
  FF01::101 (multicast address)
  ::1 (loopback address)
  ```

```
:: (unspecified addresses)
```

- An alternative form that is sometimes more convenient when dealing with a mixed environment of IPv4 and IPv6 nodes is to use *x:x:x:x:x:x:d.d.d.d*. Here, the *x*'s are the hexadecimal values of the six high-order 16-bit pieces of the address. The *d*'s are the decimal values of the four low-order 8-bit pieces of the address (standard IPv4 representation).

  Consider this example:

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

  In compressed form, it is written as:

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

6.4.2 IPv6 TCP/IP Network part (prefix)

Designers have defined some address types and have left room for future definitions, since unknown requirements may arise. RFC 2373: IP version 6 Addressing Architecture (July 1998) defines the current addressing scheme. It defines different types of prefixes (and therefore, address types) as follows:

- Link local address type

  These are special addresses that are only valid on a link of an interface. Using this address as the destination, the packet never passes through a router. It is used for link communications such as:

  – Anyone else here on this link?
  – Anyone here with a special address (for example, looking for a router)?

  Consider this example:

  ```
  fe80:  (currently the only one in use)
  fe90:
  ```

  An address with this prefix is found on each IPv6-enabled interface after stateless auto-configuration, which is normally used.

  Example: An address of FE80 : : interface ID was added to the OSA Address Table (OAT) of our OSA-Express feature at start of the device. The interface ID is derived from the MAC address of the OSA-Express feature.

- Site local address type

  These addresses are similar to the RFC 1918/Address Allocation for Private Internets in IPv4 today. The added advantage is that everyone who uses this address type has the ability to use the given 16 bits for a maximum number of 65536 subnetworks. This is comparable to the 10.0.0.0/8 address in IPv4 today.

  Another advantage is that, since it is possible to assign more than one address to an interface with IPv6, you can also assign a site local address in addition to a global one.

  It begins with the following prefixes:

  ```
  fec0:  (most commonly used)
  fed0:
  fee0:
  ```

- 6bone test addresses

These addresses were the first global addresses defined and used for testing purposes. They all begin with the following prefix:

```
3ffe:
```

- 6to4 addresses

  These addresses were designed for a special tunneling mechanism (RFC 3056/Connection of IPv6 Domains via IPv4 Clouds and RFC 2893/Transition Mechanisms for IPv6 Hosts and Routers). They encode a given IPv4 address and a possible subnet. They begin with the following prefix:

  ```
  2002:
  ```

  Consider this example, representing 192.168.1.1/5:

  ```
  2002:c0a8:0101:5::1
  ```

- Assigned by a provider for hierarchical routing

  These addresses are delegated to Internet service providers (ISP) and begin with the following prefix:

  ```
  2001:
  ```

- Multicast addresses

  Multicast addresses are used for related services and always begin with the prefix `ff`*xx*`:`. Here, *xx* is the scope value.

- Anycast addresses

  Anycast addresses are special addresses used to cover such items as the nearest Domain Name System (DNS) server, nearest Dynamic Host Configuration Protocol (DHCP) server, or similar dynamic groups. Addresses are taken out of the unicast address space, and can be aggregated globally or site-local at the moment. The anycast mechanism (client view) is handled by dynamic routing protocols.

  Note: Anycast addresses cannot be used as source addresses. They are used only as destination addresses.

  A simple example of an anycast address is the *subnet-router anycast address*. Assuming that a node has the following global assigned IPv6 address:

  ```
  3ffe:ffff:100:f101:210:a4ff:fee3:9566/64
  ```

  The *subnet-router anycast address* is created blanking the suffix (least significant 64 bits) completely:

  ```
  3ffe:ffff:100:f101::/64
  ```

6.4.3 IPv6 implementation in z/OS

The z/OS Communications Server provides support for both IPv4 and IPv6 protocols to coexist. We will review how this strategy may be implemented in a z/OS networking environment. Further details on the configuration options applied or not referenced here are available in the *z/OS V1R7.0 Communications Server: IP Configuration Reference,* SC31-8776, and *z/OS V1R7.0 CS: IPv6 Network and Application Design Guide*, SC31-8885.

Table 6-1 on page 194 summarizes the z/OS TCP/IP stack-related functions and the level of support, based on the current release of the Communications Server for z/OS. You can use this table to determine whether a given function is applicable and supported.

Table 6-1 z/OS TCP/IP stack function support

z/OS TCP/IP stack function	IPv4 support	IPv6 support	Comments
Link-layer device support	Y	Y	IPv4 devices are defined with the DEVICE and LINK configuration statements. In IPv6, interfaces are defined with the INTERFACE statement.
Ethernet LAN connectivity using OSA-Express in QDIO mode	Y	Y	To define an MPCIPA device for IPv4, use the DEVICE statement with the MPCIPA parameter and the LINK statement with the IPAQENET parameter. For IPv6 traffic, you must configure OSA-Express2 and OSA-Express features, running in QDIO mode, by using an INTERFACE statement of type IPAQENET6.
Virtual IP addressing support			
Virtual device/interface configuration	Y	Y	With IPv4, a static virtual device is configured using DEVICE and LINK statements with the VIRTUAL parameter. An IPv6 virtual interface is configured with an INTERFACE statement of type VIRTUAL6.
Sysplex support	Y	Y	IP Sysplex functions: ► Dynamic VIPA ► Sysplex Distributor ► SourceVIPA ► SNMP MIBs for dynamic VIPA ► SysplexPorts
IP routing functions			
Dynamic routing - OSPF and RIP	Y	Y	IPv6-related changes were made to OMPROUTE: ► RIPng support with Communications Server for z/OS V1R5 ► OSPFv3 support with Communications Server for z/OS V1R6
Dynamic routing - Auto Configuration	N	Y	
Static route configuration using BEGINROUTES statement	Y	Y	Related configuration statements: BEGINROUTES
Static route configuration using GATEWAY statement	Y	N	Related configuration statements: GATEWAY
Multipath routing groups	Y	Y	Related configuration statements: IPCONFIG IPCONFIG6
Miscellaneous stack functions			
Path MTU discovery	Y	Y	Path MTU discovery is mandatory in IPv6. Related configuration statements: IPCONFIG IPCONFIG6
Link-layer address resolution	Y	Y	In IPv4, this is performed using Address Resolution Protocol (ARP). In IPv6, this is performed using the neighbor discovery protocol. Related configuration statements: DEVICE and LINK (LAN Channel Station and OSA devices) INTERFACE (IPAQENET6 interfaces)

z/OS TCP/IP stack function	IPv4 support	IPv6 support	Comments
ARP/Neighbor cache PURGE capability	Y	Y	Use the V TCPIP,PURGECACHE command. For information see *z/OS V1R6.0 Communications Server: IP Systems Administration Commands*, SC31-8781.
Datagram forwarding enable/disable	Y	Y	Related configuration statements: IPCONFIG IPCONFIG6
Transport-layer functions			
Fast response cache accelerator	Y	N	
Enterprise extender	Y	N	
Server-BIND control	Y	Y	Related configuration statement: PORT
UDP Checksum disablement option	Y	N	UDP checksum is required when operating over IPv6. Related configuration statement: UDPCONFIG
Network management and accounting functions			
SNMP	Y	Y	SNMP applications can, in z/OS V1R5, communicate over an IPv6 connection. IPv6 management data includes added support for the version-neutral (both IPv4 and IPv6) MIB data in the following new, IETF Internet drafts: ► IP-MIB: draft-ietf-ipv6-rfc2011-update-01.txt ► IP-FORWARD-MIB: draft-ietf-ipv6-rfc2096-update-02.txt ► TCP-MIB: draft-ietf-ipv6-rfc2012-update-01.txt
Policy-based networking	Y	Y	IPv6 support in Policy Agent in z/OS V1R5: ► IPv6 source and destination IP addresses are allowed to be specified in policy rules (LDAP and configuration files). ► Interfaces in policy rules and subnet priority TOS masks are allowed to be specified by name. – Allowed for both IPv4 and IPv6 interfaces – IPv6 interfaces *must* be specified by name ► TOS in policy definitions means IPv4 Type of Service or IPv6 Traffic Class.
SMF SMFCONFIG	Y	Y	New Type 119 records are used to collect IPv6-related information. Related configuration statements: SMFCONFIG
IPSec	Y	N	Related configuration statement: IPCONFIG
IP filtering	Y	N	
Network access control	Y	N	Related configuration statement: NETACCESS
Stack and port access control	Y	Y	Related configuration statement: PORT DELETE
Intrusion detection services	Y	N	

Based on the discussion and recommendation in “The common design scenarios for IPv6” on page 188, will we concentrate on a single stack environment running in dual-mode. A single stack environment is one TCPIP stack running in an LPAR.

Dual-mode stack

A TCP/IP stack that supports both IPv4 and IPv6 interfaces and is capable of receiving and sending IPv4 and IPv6 packets over the corresponding interfaces is referred to as a dual-mode stack. A dual-mode stack is a single stack supporting IPv4 and IPv6 protocols, which is different from dual-stack mode that uses two TCP/IP stacks running side-by-side, each supporting only one of the protocols (either IPv4 or IPv6).

The z/OS Communications Server can be configured to support an IPv4-only stack or a dual-mode stack (IPv4 and IPv6). There is no support for an IPv6-only stack. By default, IPv6-enabled applications can communicate with both IPv4 and IPv6 peers in a dual-mode environment.

A z/OS dual-mode stack is enabled when both AF_INET and AF_INET6 are coded in SYS1.PARMLIB(BPXPRMxx). You cannot code AF_INET6 without specifying AF_INET, doing so will cause the TCP/IP stack initialization to fail.

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.

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 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 address to the native IPv4 addresses and send IPv4 packets.

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.

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 stack). 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 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.

IPv6-enabling both sockets libraries and applications on dual-mode stack therefore becomes a migration concern. As soon as IPv6-only hosts are being deployed in a network, applications on those IPv6-only partners cannot communicate with the IPv4-only applications on the dual mode hosts, unless one of the multiple migration technologies is implemented either on intermediate nodes in the network or directly on the dual mode hosts.

Table 6-2 summarizes the application communication rules when running in dual-mode.

Table 6-2 Dual-mode communication

Partner	Application communication on a dual-mode TCP/IP stack	
	IPv4 only	**IPv6 enabled**
IPv4-only	Yes	Yes
IPv6-only	No	Yes

Figure 6-1 depicts a dual-mode stack, which is the IPv6 configuration we implemented in our networking environment. The following sections walk you through the setup.

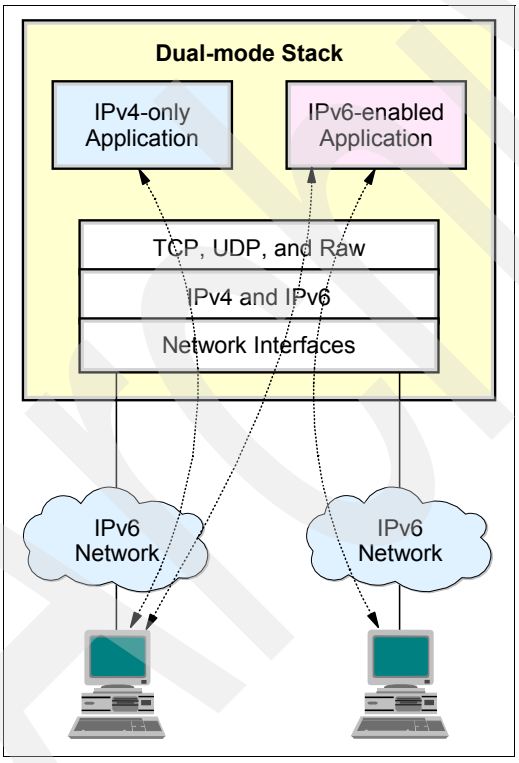


Figure 6-1 Dual-mode TCP/IP stack

Implementation tasks for a dual-mode stack

To implement a dual-mode stack in our networking environment, we modified the following:

- BPXPRMxx definitions
- VTAM definitions
- TCP/IP definitions

BPXPRMxx definitions

IPv6 is not enabled, by default. You must specify a NETWORK statement with AF_INET6 in your BPXPRMxx member.

To support our dual-mode stack (IPv4 and IPv6), we added the NETWORK statement, as shown in Example 6-1, to our BPXPRMxx member.

Example 6-1 BPXPRMxx NETWORK statement

```
NETWORK DOMAINNAME (AF_INET6)
DOMAINNUMBER(19)
MAXSOCKETS(2000)
TYPE(INET)
```

The TYPE option in our case is INET, because we are using a single stack.

Note: The BPXPRMxx member can be updated dynamically using the z/OS command SETOMVS RESET=(xx). After the reset, we received the message `BPXF203I DOMAIN AF_INET6 WAS SUCCESSFULLY ACTIVATED`. We then *recycled* the TCP/IP stack to pick up the change.

For more details about the definitions required in BPXPRMxx to provide a dual- stack, refer to *z/OS V1R7 CS: IP Configuration Guide*, SC31-8775.

VTAM definitions

As mentioned in the introduction, one of the protocols that CS for z/OS TCP/IP supports is MPC+, and the MPC+ protocols are used to define the DLCs for OSA-Express devices in QDIO. OSA-Express QDIO connections are configured via a TRLE definition. Since VTAM provides the DLCs for TCP/IP, all TRLEs are defined as VTAM major nodes (see Example 6-2).

Example 6-2 TRLE definition

```
OSA2080  VBUILD TYPE=TRL
OSA2080P TRLE  LNCTL=MPC,
               READ=2080,
               WRITE=2081,
               DATAPATH=(2082-2087),
               PORTNAME=OSA2080,    1
               MPCLEVEL=QDIO
```

The PORTNAME (1) is identical to the device name defined in the TCP/IP PROFILE data set on the INTERFACE statement.

TCP/IP definitions

We added one INTERFACE statement for the OSA-Express2 1000BASE-T port to support IPv6. This statement merges the DEVICE, LINK, and HOME definitions into a single statement. Several different parameters are associated with the INTERFACE statement. To determine which of them best fits your requirements, refer to *z/OS Communications Server IP Configuration Reference Version 1 Release 7,* SC31-8776.

We used the following syntax:

```
INTERFace interfname DEFINE linktype PORTNAME portname IPADDR ipaddr
```

Note the following explanation:

- *interfname* specifies a name for the interface with no more than 16 characters in length.
- *linktype* must be IPAQENET6, the only DLC that currently supports IPv6.
- *portname* is specified in the VTAM TRLE definition for the QDIO interface.
- *ipaddr* is optional for link type IPAQENET6. If not specified, TCP/IP enables auto-configuration for the interface. If used, one or more prefixes or full IPv6 addresses can be specified.

Note: To configure a single physical device for both IPv4 and IPv6 traffic, you must use DEVICE/LINK/HOME for the IPv4 definition and INTERFACE for the IPv6 definition, so that the PORTNAME value on the INTERFACE statement matches the device name on the DEVICE statement.

The TCP/IP IPv6 PRFOFILE for a single stack is illustrated in the following example. The INTERFACE statement defines the configuration of the OSA-Express device (OSA2080) that we use for network connectivity. The PORTNAME must be identical to the PORTNAME defined in the TRLE. The TRLE is defined as a VTAM major node in the VTAM definition data set.

Example 6-3 shows the TCP/IP profile for our environment, using SYSTEM SYMBOLS and INCLUDE statements. The &sysclone that you see throughout the example will result in a two-digit value (30 in our example, for system SC30) being inserted. By doing this we can use the same profile for each of several systems, each time translating to the appropriate system value (systems 30, 31, and 32). The &sysclone value is defined in SYS1.PARMLIB.

Example 6-3 Profile definition with the use of SYSTEM SYMBOLS and INCLUDE

```
ARPAGE 20
;
GLOBALCONFIG TCPIPSTATISTICS
;
IPCONFIG                                    1
  DATAGRAMFWD
  SYSPLEXROUTING
  SOURCEVIPA
;
DYNAMICXCF 10.10.20.1&sysclone 255.255.255.0 8
;
IPCONFIG6                                   2
  DATAGRAMFWD
  SOURCEVIPA
;
SOMAXCONN 240
;
TCPCONFIG TCPSENDBFRSIZE 16K TCPRCVBUFRSIZE 16K SENDGARBAGE FALSE
TCPCONFIG RESTRICTLOWPORTS
;
UDPCONFIG RESTRICTLOWPORTS
;
INCLUDE TCPIPE.TCPPARMS(HOME&SYSCLONE.6)
;
INCLUDE TCPIPE.TCPPARMS(STAT&SYSCLONE.6)
;
AUTOLOG 5
   FTPDE&sysclone  JOBNAME FTPDE&sysclone.1
;  OMP&sysclone             ; OSPF daemon
```

```
;  SMTP                     ; SMTP Server
ENDAUTOLOG
;
PORT
    20 TCP *  NOAUTOLOG     ; FTP Server
    21 TCP OMVS    BIND 10.10.10.210; control port
    23 TCP INTCLIEN            ; MVS Telnet Server
    23 TCP OMVS BIND 10.10.10.210 ;Telnet Server
    25 TCP SMTP                ; SMTP Server
;  111 TCP PORTMAP             ; Portmap Server
;  111 UDP PORTMAP             ; Portmap Server
;  443 TCP HTTPS               ; http protocol over TLS/SSL
;  443 UDP HTTPS               ; http protocol over TLS/SSL
   514 UDP OMVS                ; UNIX Syslogd daemon
; 3389 TCP MSYSLDAP            ; LDAP Server for Msys
;
SACONFIG ENABLED COMMUNITY public AGENT 161
;
SMFCONFIG
  FTPCLIENT TN3270CLIENT
  TYPE119 FTPCLIENT TN3270CLIENT
;
INCLUDE TCPIPE.TCPPARMS(TELN&SYSCLONE)
```

1 defines the IPv4 environment.

2 defines the IPv6 environment.

Example 6-4 shows the DEVICE, LINK, HOME, INTERFACE, and IPADDR definitions we used to support IPv4 and IPv6 and their addressing schemes.

Example 6-4 Interface and address definitions

```
DEVICE OSA2080  MPCIPA          1
 LINK   OSA2080LNK  IPAQENET      OSA2080 VLANID 10
 INTERFACE LNK62080 DEFINE IPAQENET6 PORTNAME OSA2080      1
 IPADDR FEC0:0:0:1::3302
        FEC0:0:0:1001::3302
;
; Static VIPA definitions
 DEVICE STAVIPA1    VIRTUAL 0
 LINK   STAVIPA1LNK VIRTUAL 0     STAVIPA1
;
HOME
   10.10.10.210   STAVIPA1LNK
   10.10.2.212    OSA2080LNK
;
START OSA2080
START LNK62080
```

1 defines the same device (OSA2080) to support IPv4 and IPv6 addresses.

Example 6-5 show static routes in a flat network (no dynamic routing protocol).

Example 6-5 Static route definitions

```
BEGINRoutes
; Direct Routes - Routes that are directly connected to my interfaces
;     Destination     Subnet Mask   First Hop Link Name     Packet Size ;
 ROUTE FEC0::0/10                     =        LNK62080      mtu 1492
```

```
 ROUTE 10.10.2.0       255.255.255.0 =        OSA2080LNK    mtu 1492
; Default Route - All packets to an unknown destination are routed
;through this route.
;     Destination     First Hop    Link Name   Packet Size
 ROUTE DEFAULT         10.10.2.1    OSA2080LNK  mtu 1492
ENDRoutes
```

Example 6-6 shows our VTAM and Telnet (TN3270) definitions.

Example 6-6 Our definitions for TN3270

```
TelnetParms
  Port 23                          ; Port number 23 (std.)
  TELNETDEVICE 3278-3-E NSX32703 ; 32 line screen -
                                   ; default of NSX3270 is 24
  TELNETDEVICE 3279-3-E NSX32703 ; 32 line screen -
                                   ; default of NSX3270 is 24
  TELNETDEVICE 3278-4-E NSX32704 ; 48 line screen -
                                   ; default of NSX3270 is 24
  TELNETDEVICE 3279-4-E NSX32704 ; 48 line screen -
                                   ; default of NSX3270 is 24
  TELNETDEVICE 3278-5-E NSX32705 ; 132 column screen-
                                   ; default of NSX3270 is 80
  TELNETDEVICE 3279-5-E NSX32705 ; 132 column screen -
                                   ; default of NSX3270 is 80
  LUSESSIONPEND     ; On termination of a Telnet server connection,
                    ; the user will revert to the DEFAULTAPPL
                    ; instead of having the connection dropped
MSG07             ; Sends a USS error message to the client if an
                    ; error occurs during session establishment
                    ; instead of dropping the connection
  CodePage ISO8859-1 IBM-1047   ; Linemode ASCII, EBCDIC code pages
  Inactive 0                    ; Let connections stay around
  PrtInactive 0                 ; Let connections stay around
  TimeMark 600
  ScanInterval 120
; SMFinit std
; SMFterm std
; Define logon mode tables to be the defaults shipped with the
  ; latest level of VTAM
EndTelnetParms
;
BeginVTAM
  Port 23
  ; Define the LUs to be used for general users.
  DEFAULTLUS
    SC&SYSCLONE.TSO1..SC&SYSCLONE.TS30
  ENDDEFAULTLUS
LINEMODEAPPL TSO  ; Send all line-mode terminals directly to TSO.
ALLOWAPPL SC*  DISCONNECTABLE ; Allow all users access to TSO
  ALLOWAPPL TSO* DISCONNECTABLE ; Allow all users access to TSO
             ; applications.
             ; TSO is multiple applications all beginning with TSO,
             ; so use the * to get them all.  If a session is closed,
             ; disconnect the user rather than log off the user.
ALLOWAPPL *      ; Allow all applications that have not been
                   ; previously specified to be accessed.
RESTRICTAPPL IMS ; Only 3 users can use IMS.
    USER USER1     ; Allow user1 access.
      LU TCPIMS01  ; Assign USER1 LU TCPIMS01.
```

```
    USER USER2     ; Allow user2 access from the default LU pool.
    USER USER3     ; Allow user3 access from 3 Telnet sessions,
                   ; each with a different reserved LU.
      LU TCPIMS31 LU TCPIMS32 LU TCPIMS33
USSTCP USSTEST1
EndVTAM
```

The following messages were written to the z/OS console when the TCP/IP stack was initializing (see Example 6-7).

Example 6-7 TCP/IP stack initialization

```
J E S 2  J O B  L O G  --  S Y S T E M  S C 3 0  --  N O D E  W T S C P L X 5

--- FRIDAY,    21 OCT 2005 ----
IEF695I START TCPIPE   WITH JOBNAME TCPIPE   IS ASSIGNED TO USER TCPIP   , GROUP
$HASP373 TCPIPE   STARTED
IEE252I MEMBER CTIEZB00 FOUND IN SYS1.IBM.PARMLIB
IEE252I MEMBER CTIIDS00 FOUND IN SYS1.IBM.PARMLIB
EZZ7450I FFST SUBSYSTEM IS NOT INSTALLED
EZZ0300I OPENED INCLUDE FILE 'TCPIPE.TCPPARMS(HOME306)'
EZZ0300I OPENED INCLUDE FILE 'TCPIPE.TCPPARMS(STAT306)'
EZZ0300I OPENED INCLUDE FILE 'TCPIPE.TCPPARMS(TELN30)'
EZZ0300I OPENED PROFILE FILE DD:PROFILE
EZZ0309I PROFILE PROCESSING BEGINNING FOR DD:PROFILE
EZZ0309I PROFILE PROCESSING BEGINNING FOR TCPIPE.TCPPARMS(HOME306)
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'TCPIPE.TCPPARMS(HOME306)'
EZZ0304I RESUMING PROCESSING OF FILE DD:PROFILE
EZZ0309I PROFILE PROCESSING BEGINNING FOR TCPIPE.TCPPARMS(STAT306)
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'TCPIPE.TCPPARMS(STAT306)'
EZZ0304I RESUMING PROCESSING OF FILE DD:PROFILE
EZZ0309I PROFILE PROCESSING BEGINNING FOR TCPIPE.TCPPARMS(TELN30)
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'TCPIPE.TCPPARMS(TELN30)'
EZZ0304I RESUMING PROCESSING OF FILE DD:PROFILE
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE DD:PROFILE
EZZ0641I IP FORWARDING NOFWDMULTIPATH SUPPORT IS ENABLED
EZZ0350I SYSPLEX ROUTING SUPPORT IS ENABLED
EZZ0351I SOURCEVIPA SUPPORT IS ENABLED
EZZ0624I DYNAMIC XCF DEFINITIONS ARE ENABLED
EZZ0700I IPV6 FORWARDING NOFWDMULTIPATH SUPPORT IS ENABLED   1
EZZ0702I IPV6 SOURCEVIPA SUPPORT IS ENABLED          1
EZZ0338I TCP PORTS 1 THRU 1023 ARE RESERVED
EZZ0338I UDP PORTS 1 THRU 1023 ARE RESERVED
EZZ0613I TCPIPSTATISTICS IS DISABLED
EZZ4202I Z/OS UNIX - TCP/IP CONNECTION ESTABLISHED FOR TCPIPE
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE OSA2080
EZD1176I TCPIPE HAS SUCCESSFULLY JOINED THE TCP/IP SYSPLEX GROUP
EZB6473I TCP/IP STACK FUNCTIONS INITIALIZATION COMPLETE.
EZAIN11I ALL TCPIP SERVICES FOR PROC TCPIPE ARE AVAILABLE.
EZZ4340I INITIALIZATION COMPLETE FOR INTERFACE LNK62080       2
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE IUTIQDIO
EZZ0400I TELNET/VTAM (SECOND PASS) BEGINNING FOR FILE: DD:PROFILE
EZZ0309I PROFILE PROCESSING BEGINNING FOR TCPIPE.TCPPARMS(HOME306)
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'TCPIPE.TCPPARMS(HOME306)'
EZZ0304I RESUMING PROCESSING OF FILE DD:PROFILE
EZZ0309I PROFILE PROCESSING BEGINNING FOR TCPIPE.TCPPARMS(STAT306)
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'TCPIPE.TCPPARMS(STAT306)'
EZZ0304I RESUMING PROCESSING OF FILE DD:PROFILE
EZZ0309I PROFILE PROCESSING BEGINNING FOR TCPIPE.TCPPARMS(TELN30)
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'TCPIPE.TCPPARMS(TELN30)'
```

```
EZZ0304I RESUMING PROCESSING OF FILE DD:PROFILE
EZZ6003I TELNET LISTENING ON PORT    23
EZZ0403I TELNET/VTAM (SECOND PASS) COMPLETE FOR FILE: DD:PROFILE
EZD0011I USE OF SITE LOCAL ADDRESS FEC0::1001:0:0:0:3302 IS NOT RECOMMENDED    3
EZD0011I USE OF SITE LOCAL ADDRESS FEC0::1:0:0:0:3302 IS NOT RECOMMENDED       3
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE IUTSAMEH
EZZ4324I CONNECTION TO 10.30.20.100 ACTIVE FOR DEVICE IUTSAMEH
EZZ4324I CONNECTION TO 10.20.10.100 ACTIVE FOR DEVICE IUTSAMEH
EZZ4324I CONNECTION TO 10.20.40.100 ACTIVE FOR DEVICE IUTSAMEH
S FTPDE30
```

1 through 3 indicate that IPv6 support is enabled and that the interface is initialized with IPv6 addresses.

6.4.4 Verification

We now verify our environment. Since the TRLE must be active before the interface is started, we ensure that the TRLE is in an active state.

The results are shown in Figure 6-4. You can also verify the OSA-Express code level with this command.

Example 6-8 OSA-Express status and code level

```
D NET,TRL,TRLE=OSA2080P
IST097I DISPLAY ACCEPTED
IST075I NAME = OSA2080P, TYPE = TRLE 384
IST1954I TRL MAJOR NODE = OSA2080
IST486I STATUS= ACTIV, DESIRED STATE= ACTIV   1
IST087I TYPE = LEASED           , CONTROL = MPC , HPDT = YES
IST1715I MPCLEVEL = QDIO       MPCUSAGE = SHARE
IST1716I PORTNAME = OSA2080    LINKNUM =   0   OSA CODE LEVEL = 0804  4
IST1577I HEADER SIZE = 4096 DATA SIZE = 0 STORAGE = ***NA***
IST1221I WRITE DEV = 2081 STATUS = ACTIVE     STATE = ONLINE     2
IST1577I HEADER SIZE = 4092 DATA SIZE = 0 STORAGE = ***NA***
IST1221I READ  DEV = 2080 STATUS = ACTIVE     STATE = ONLINE      2
IST1221I DATA  DEV = 2082 STATUS = ACTIVE     STATE = N/A           3
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1717I ULPID = TCPIPE
```

1 indicates the state of the TRLE major node.

2 and 3 are the desired and required states.

4 indicates the OSA code level, which is a four-digit number that relates to a specific microcode engineering change (EC) and patch level (MCL).

Example 6-9 displays the HOME addresses after initialization.

Example 6-9 HOME addresses displayed

```
D TCPIP,TCPIPE,N,HOME
EZD0101I NETSTAT CS V1R7 TCPIPE 617   617
HOME ADDRESS LIST:
LINKNAME:   STAVIPA1LNK
  ADDRESS:  10.10.10.210
    FLAGS:  PRIMARY
LINKNAME:   OSA2080LNK      1
  ADDRESS:  10.10.2.212
```

```
    FLAGS:
LINKNAME:   EZASAMEMVS
  ADDRESS:  10.10.20.130
    FLAGS:
LINKNAME:   IQDIOLNKOA0A1482
  ADDRESS:  10.10.20.130
    FLAGS:
LINKNAME:   LOOPBACK
  ADDRESS:  127.0.0.1
    FLAGS:
INTFNAME:   LNK62080              2
  ADDRESS:  FEC0:0:0:1::3302
    TYPE:   SITE_LOCAL
    FLAGS:
  ADDRESS:  FEC0:0:0:1001::3302     2
    TYPE:   SITE_LOCAL
    FLAGS:
  ADDRESS:  FE80::9:6B00:21A:7490     3
    TYPE:   LINK_LOCAL
    FLAGS:  AUTOCONFIGURED
INTFNAME:   LOOPBACK6             4
  ADDRESS:  ::1
    TYPE:   LOOPBACK
    FLAGS:
9 OF 9 RECORDS DISPLAYED
```

1 This is the IPv4 address assigned to the OSA Express device (OSA2080).

2 These are the IPv6 addresses assigned to the same OSA Express device (OSA2080) defined with the INTERFACE statement. However, these SITE_LOCAL addresses are not generally recommended.

3 This is an auto-configured LINK_LOCAL address for the same OSA Express device.

4 This is the IPv6 Loopback address.

Example 6-10 shows the output of NETSTAT DEV, with the IPv6 Loopback and device interfaces shown as READY.

Example 6-10 Device display, using Netstat

```
D TCPIP,TCPIPE,N,DEV
EZD0101I NETSTAT CS V1R7 TCPIPE 627   627
DEVNAME: LOOPBACK          DEVTYPE: LOOPBACK
  DEVSTATUS: READY
  LNKNAME: LOOPBACK          LNKTYPE: LOOPBACK   LNKSTATUS: READY
    NETNUM: N/A  QUESIZE: N/A
    ACTMTU: 65535
  BSD ROUTING PARAMETERS:
    MTU SIZE: N/A               METRIC: 00
    DESTADDR: 0.0.0.0           SUBNETMASK: 0.0.0.0
  MULTICAST SPECIFIC:
    MULTICAST CAPABILITY: NO
  LINK STATISTICS:
    BYTESIN                          = 17662
    INBOUND PACKETS                  = 257
    INBOUND PACKETS IN ERROR         = 0
    INBOUND PACKETS DISCARDED        = 0
OMMAND INPUT ===>                                            SCR
  INBOUND PACKETS WITH NO PROTOCOL  = 0
```

```
  BYTESOUT                          = 17662
  OUTBOUND PACKETS                  = 257
  OUTBOUND PACKETS IN ERROR         = 0
  OUTBOUND PACKETS DISCARDED        = 0
INTFNAME: LOOPBACK6        INTFTYPE: LOOPBACK6 INTFSTATUS: READY    1
  NETNUM: N/A  QUESIZE: N/A
  ACTMTU: 65535
MULTICAST SPECIFIC:
  MULTICAST CAPABILITY: NO
INTERFACE STATISTICS:
  BYTESIN                           = 0
  INBOUND PACKETS                   = 0
  INBOUND PACKETS IN ERROR          = 0
  INBOUND PACKETS DISCARDED         = 0
  INBOUND PACKETS WITH NO PROTOCOL  = 0
  BYTESOUT                          = 0
    OUTBOUND PACKETS                  = 0
    OUTBOUND PACKETS IN ERROR         = 0
    OUTBOUND PACKETS DISCARDED        = 0
DEVNAME: OSA2080           DEVTYPE: MPCIPA
  DEVSTATUS: READY
  LNKNAME: OSA2080LNK        LNKTYPE: IPAQENET   LNKSTATUS: READY     2
    NETNUM: N/A  QUESIZE: N/A  SPEED: 0000000100
    IPBROADCASTCAPABILITY: NO
    CFGROUTER: NON                  ACTROUTER: NON
    ARPOFFLOAD: YES                 ARPOFFLOADINFO: YES
    ACTMTU: 1492
    VLANID: 10                      VLANPRIORITY: DISABLED
    DYNVLANREGCFG: NO               DYNVLANREGCAP: YES
    READSTORAGE: GLOBAL (4096K)     INBPERF: BALANCED
    CHECKSUMOFFLOAD: YES            SEGMENTATIONOFFLOAD: YES
    SECCLASS: 255
  BSD ROUTING PARAMETERS:
  MTU SIZE: N/A              METRIC: 00
  DESTADDR: 0.0.0.0          SUBNETMASK: 255.255.255.0
MULTICAST SPECIFIC:
  MULTICAST CAPABILITY: YES
  GROUP            REFCNT
  -----            ------
  224.0.0.1        0000000001
LINK STATISTICS:
  BYTESIN                           = 52142
  INBOUND PACKETS                   = 294
  INBOUND PACKETS IN ERROR          = 552
  INBOUND PACKETS DISCARDED         = 0
  INBOUND PACKETS WITH NO PROTOCOL  = 0
  BYTESOUT                          = 366
  OUTBOUND PACKETS                  = 4
  OUTBOUND PACKETS IN ERROR         = 0
  OUTBOUND PACKETS DISCARDED        = 0
INTFNAME: LNK62080         INTFTYPE: IPAQENET6 INTFSTATUS: READY     3
  NETNUM: N/A  QUESIZE: 0    SPEED: 0000000100
  MACADDRESS: 00096B1A7490
  DUPADDRDET: 1
  CFGROUTER: NON                  ACTROUTER: NON
  CFGMTU: NONE                    ACTMTU: 1500
  READSTORAGE: GLOBAL (4096K)     INBPERF: BALANCED
MULTICAST SPECIFIC:
  MULTICAST CAPABILITY: YES
  REFCNT      GROUP
```

```
  ------      -----
  0000000002  FF02::1:FF00:3302
  0000000001  FF02::1:FF1A:7490
  0000000001  FF01::1
  0000000001  FF02::1
INTERFACE STATISTICS:
  BYTESIN                           = 0
    INBOUND PACKETS                   = 0
    INBOUND PACKETS IN ERROR          = 0
    INBOUND PACKETS DISCARDED         = 0
    INBOUND PACKETS WITH NO PROTOCOL  = 0
    BYTESOUT                          = 1304
    OUTBOUND PACKETS                  = 13
    OUTBOUND PACKETS IN ERROR         = 0
    OUTBOUND PACKETS DISCARDED        = 0
```

If the device does not have a LnkStatus or IntfStatus of *Ready* (as with 1, 2, and 3) you must resolve this before you continue. There are several factors that may cause the LnkStatus or IntfStatus to not be ready. For example, the device may not be varied online or defined to z/OS correctly, or the device may not be defined in the TCP/IP profile correctly, and so on.

In Figure 6-11 we see the FTPD server 1 and the Internal Telnet server (TN3270) 2 bound to the IPv6 address. They may now be accessed by another IPv6-enabled client across an IPv4 network.

Example 6-11 Sockets in the stack

```
D TCPIP,TCPIPE,N,SOCKETS
EZD0101I NETSTAT CS V1R7 TCPIPE 643
SOCKETS INTERFACE STATUS:
NAME: DCD1DIST  SUBTASK: 007D0988
TYPE: STREAM  STATUS:  LISTEN    CONN: 00000045
BOUNDTO: 0.0.0.0..38241
CONNTO:  0.0.0.0..0
NAME: DCD1DIST  SUBTASK: 007D21C0
TYPE: STREAM  STATUS:  LISTEN    CONN: 00000043
BOUNDTO: 0.0.0.0..38240
CONNTO:  0.0.0.0..0
NAME: FTPDE301  SUBTASK: 007FF540
TYPE: STREAM  STATUS:  LISTEN    CONN: 00000026
BOUNDTO: ::FFFF:10.10.10.210..21      1
CONNTO:  ::FFFF:0.0.0.0..0
NAME: TCPIPE    SUBTASK: 007D2D80
TYPE: STREAM  STATUS:  LISTEN    CONN: 0000001E
BOUNDTO: ::..23          2
CONNTO:  ::..0
NAME: TCPIPE    SUBTASK: 007E4BE8
TYPE: STREAM  STATUS:  ESTBLSH   CONN: 00000019
BOUNDTO: 127.0.0.1..1025
CONNTO:  127.0.0.1..1024
NAME: TCPIPE    SUBTASK: 007E6370
TYPE: STREAM  STATUS:  ESTBLSH   CONN: 0000001A
BOUNDTO: 127.0.0.1..1024
CONNTO:  127.0.0.1..1025
TYPE: STREAM  STATUS:  LISTEN    CONN: 00000014
BOUNDTO: 127.0.0.1..1024
CONNTO:  0.0.0.0..0
7 OF 7 RECORDS DISPLAYED
END OF THE REPORT
```

We used the **ping** command from various hosts within the network to verify connectivity for IPv4 and IPv6 (see Example 6-12).

Example 6-12 Results of the ping command

```
TSO PING FEC0:0:0:1001::3302
CS V1R7: Pinging host FEC0:0:0:1001::3302
Ping #1 response took 0.000 seconds.

TSO PING FE80::9:6B00:21A:7490
CS V1R7: Pinging host FE80::9:6B00:21A:7490
Ping #1 response took 0.000 seconds.

TSO PING 10.10.2.232 (TCP tcpipe
CS V1R7: Pinging host 10.10.2.232
Ping #1 response took 0.000 seconds.
```

A

Component trace (CTRACE)

The MVS component trace can be used for diagnosis of most TCP/IP problems. Some components of TCP/IP continue to maintain their own tracing mechanisms, for example, the FTP server. Consult the *z/OS V1R7.0 Communications Server: IP Diagnosis Guide*, GC31-8782, and *z/OS V1R7.0 MVS Diagnosis: Tools and Service Aids*, GA22-7589, for more information about the various trace options.

The following TCP/IP traces are available using the component trace:

- Event trace for TCP/IP stacks and TN3270 Telnet server in its own space
- TCP/IP packet trace
- OMPROUTE trace
- Resolver trace
- TCP/IP intrusion detection service trace
- Configuration profile trace

Information APAR II12014 is a useful source of information about the TCP/IP component and packet trace.

For general information about the MVS component trace, see *z/OS V1R7.0 MVS Diagnosis: Tools and Service Aids*, GA22-7589.

Taking a component trace

Component trace data is written to either an external writer or the TCP/IP data space TCPIPDS1 (the default is to write trace data to the data space). The following command sequence starts a component trace that uses the external writer; this allows you to store trace data in data sets, which can later be used as input to IPCS.

In the following commands, *component* is one of the following:

- SYSTCPIP for the TCP/IP event trace and TN3270 Telnet server in its own space
- SYSTCPDA for the TCP/IP packet trace
- SYSTCPRT for the OMPROUTE trace
- SYSTCPRE for the Resolver trace
- SYSTCPIS for the TCP/IP intrusion detection service trace
- Configuration profile trace

proc_name is the name of the TCP/IP, OMPROUTE, or RESOLVER procedure.

1. Start the external writer (CTRACE writer):

   ```
   TRACE CT,WTRSTART=ctwrt
   ```

   Note: Before starting the external writer, ensure that you have the ctwrt procedure in the SYS1.PROCLIB library.

   Figure A-1 shows a sample of an external writer. GDGs are used and are ideal for creating multiple trace data sets, if the need arises, while keeping traces of previous conditions or errors. Other examples can be found in II12014 informational APAR.

```
//PTTCP  PROC
//*******************************************************************
//*THIS PROCEDURE IS THE IPCS CTRACE EXTERNAL
//*PROCEDURE USED BY TCP/IP DATA TRACING
//*WITH GDGS
//*******************************************************************
//IEFPROC EXEC PGM=ITTTRCWR
//TRCOUT01 DD DSN=SYS2.TCPIP.TRACED.DATA(+1),UNIT=DISK,
//         DCB=SYS.MODEL,DISP=(,CATLG),
//         SPACE=(CYL,200,,CONTIG)
//SYSPRINT  DD SYSOUT=*
```

Figure A-1 External writer

2. Start the CTRACE and connect to the external writer:

   ```
   TRACE CT,ON,COMP=component,SUB=(proc_name)
   R xx,OPTION=(valid_options),WTR=ctwrt,END
   ```

3. Display the active component trace options with:

   ```
   DISPLAY TRACE,COMP=component,SUB=(proc_name)
   ```

4. Perform the operation you want to trace.

5. Disconnect the external writer:

   ```
   TRACE CT,ON,COMP=component,SUB=(proc_name)
   R xx,WTR=DISCONNECT,END
   ```

6. Stop the component trace:

   ```
   TRACE CT,OFF,COMP=component,SUB=(proc_name)
   ```

7. Stop the external writer:

```
TRACE CT,WTRSTOP=ctwrt
```

Event Trace for TCP/IP stacks (SYSTCPIP)

The TCP/IP event trace, SYSTCPIP, traces individual TCP/IP components, such as storage. It is automatically started at TCP/IP initialization using the default trace options set in the SYS1.PARMLIB member CTIEZB00 for SYSTCPIP and CTIEZBTN for TN3270 Telnet server. Alternatively, you may create a different member with new options (for example, CTIEZBXX) and override CTIEZB00 by means of the CTRACE keyword in your TCP/IP PROC.

```
//TCPIPC   PROC PARMS='CTRACE(CTIEZBXX)'
//*
//TCPIP    EXEC PGM=EZBTCPIP,REGION=0M,TIME=1440,
//             PARM='&PARMS'
```

Figure A-2 Overriding CTIEZB00 with CTIEZBXX

If you wish to specify different trace options after TCP/IP initialization, you may execute the **TRACE CT** MVS command, and either specify a component trace options file or respond to prompts from the command. However, you might want to edit the buffer size in this member and increase its size from the default of 8 MB. Buffer sizes, unlike many of the other CTRACE options, cannot be reset without restarting the TCP/IP address space.

Figure A-3 shows the status of the component trace for TCP/IP procedure TCPIPC as it has been initialized using SYS1.PARMLIB member CTIEZB01. Note that we have changed the default value for BUFSIZE to 4M.

```
DISPLAY TRACE,COMP=SYSTCPIP,SUB=(TCPIPC)
IEE843I 10.31.34  TRACE DISPLAY 003
        SYSTEM STATUS INFORMATION
 ST=(ON,0064K,00128K) AS=ON  BR=OFF EX=ON  MT=(ON,064K)
  TRACENAME
  =========
  SYSTCPIP
                     MODE BUFFER HEAD SUBS
                     =====================
                     OFF         HEAD    3
     NO HEAD OPTIONS
  SUBTRACE           MODE BUFFER HEAD SUBS
 ------------------------------------------------------------
  TCPIPC             ON   0004M
     ASIDS      *NONE*
     JOBNAMES   *NONE*
     OPTIONS    MINIMUM
     WRITER     *NONE*
```

Figure A-3 DISPLAY TRACE,COMP=SYSTCPIP,SUB=(TCPIPC) output

The MINIMUM trace option is always active. During minimum tracing, certain exceptional conditions are being traced so the trace records for these events will be available for easier debugging in case the TCP/IP address space should encounter an abend condition.

Socket API trace

The SOCKAPI option for the TCP/IP CTRACE component SYSTCPIP is intended to be used for application programmers to debug problems in their application. The SOCKAPI option captures trace information related to the socket API calls that an application may issue. However, the SOCKET option is primarily intended for use by TCP/IP Service and provides information meant to be used to debug problems in the TCP/IP socket layer, UNIX System Services, or the TCP/IP stack. Please refer to *z/OS V1R7.0 CS: IP Diagnosis,* GC31-8782, for further details on the SOCKAPI option.

Sample SYSTCPIP trace

Figure A-4 on page 213 shows the trace activation process (steps 1, 2, and 3) and Figure A-5 on page 214 shows the deactivation process (steps 5, 6, and 7). The TN3270 server running in its own address space also uses the SYSTCPIP event trace; therefore, all discussions that follow here where TCP/IP is used also pertain to the Telnet server, with the following exceptions:

- The Telnet server does not use a dataspace for trace data collection, but its own private storage.
- A subset of the trace commands are use by Telnet so a new default member, CTIEZBTN, is created, which provides an indication of the trace options available. This member may also be overwritten in the same manner as the TCP/IP parmlib member can be overwritten.
- A subset of IPCS commands are used by Telnet.

Note: If using the Telnet option, do not specify the JOBNAME parm when starting CTRACE.

Tracing steps

Seven steps to problem determination are:

1. Start the external writer (CTRACE writer).
2. Connect to the CTRACE external writer and specify trace options.
3. Display the active component trace options.
4. Reproduce the failure you want to trace.
5. Disconnect the external writer.
6. Stop the component trace.
7. Stop the external writer.

Instead of specifying options individually as we did in step 2, we could have created a SYS1.PARMLIB member that would have started the external writer for us and included the desired options. We could use the following command for that:

```
TRACE CT,ON,COMP=SYSTCPIP,SUB=(tcpproc),PARM=(CTIEZBXX)
```

```
TRACE CT,WTRSTART=CTVLAD
ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
WERE SUCCESSFULLY EXECUTED.
IEE839I ST=(ON,0064K,00128K) AS=ON  BR=OFF EX=ON  MT=(ON,064K) 413
     ISSUE DISPLAY TRACE CMD FOR SYSTEM AND COMPONENT TRACE STATUS
     ISSUE DISPLAY TRACE,TT CMD FOR TRANSACTION TRACE STATUS
IRR813I NO PROFILE WAS FOUND IN THE STARTED CLASS FOR 414
     CTVLAD WITH JOBNAME CTVLAD. RACF WILL USE ICHRIN03.
IEF196I        1 //CTVLAD   JOB MSGLEVEL=1
IEF196I        2 //STARTING EXEC CTVLAD
IEF196I  STMT NO. MESSAGE
IEF196I        2 IEFC001I PROCEDURE CTVLAD WAS EXPANDED USING SYSTEM
IEF196I LIBRARY SYS1.PROCLIB
IEF196I        3 XXCTRACE   PROC
IEF196I          XX*         EXTERNAL WRITER USED WITH CS V2R10
IEF196I        4 XXIEFPROC  EXEC  PGM=ITTTRCWR
IEF196I        5 XXTRCOUT01 DD DSNAME=LUNA.CTRACE,DISP=OLD
IEF403I CTVLAD - STARTED - TIME=16.26.58
IEF196I IEF236I ALLOC. FOR CTVLAD CTVLAD
IEF196I IEF237I 0829 ALLOCATED TO TRCOUT01
IEF196I AHL906I THE OUTPUT BLOCK SIZE OF    27998 WILL BE USED FOR
IEF196I OUTPUT
IEF196I         DATA SETS:
IEF196I           LUNA.CTRACE
AHL906I THE OUTPUT BLOCK SIZE OF  27998 WILL BE USED FOR OUTPUT 441
        DATA SETS:
          LUNA.CTRACE
ITT110I INITIALIZATION OF CTRACE WRITER CTVLAD COMPLETE.
...
TRACE CT,ON,COMP=SYSTCPIP,SUB=(TCPIPC)
*90 ITT006A SPECIFY OPERAND(S) FOR TRACE CT COMMAND.
 R 90,JOBNAME=(TCPIPC),OPTIONS=(XCF)
 IEE600I REPLY TO 90 IS;JOBNAME=(TCPIPC),OPTIONS=(XCF)
*91 ITT006A SPECIFY OPERAND(S) FOR TRACE CT COMMAND.
 R 91,WTR=CTVLAD,END
 IEE600I REPLY TO 91 IS;WTR=CTVLAD,END
 ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
 WERE SUCCESSFULLY EXECUTED.
 IEE839I ST=(ON,0064K,00128K) AS=ON  BR=OFF EX=ON MT=(ON,064K) 515
      ISSUE DISPLAY TRACE CMD FOR SYSTEM AND COMPONENT TRACE STATUS
      ISSUE DISPLAY TRACE,TT CMD FOR TRANSACTION TRACE STATUS
...
DISPLAY TRACE,COMP=SYSTCPIP,SUB=(TCPIPC)
IEE843I 16.29.59  TRACE DISPLAY 526
        SYSTEM STATUS INFORMATION
 ST=(ON,0064K,00128K) AS=ON  BR=OFF EX=ON  MT=(ON,064K)
  TRACENAME
  =========
  SYSTCPIP
                     MODE BUFFER HEAD SUBS
                     =====================
                     OFF         HEAD    3
     NO HEAD OPTIONS
  SUBTRACE           MODE BUFFER HEAD SUBS
 ----------------------------------------------------------------
  TCPIPC             ON   0004M
     ASIDS      *NONE*
     JOBNAMES   TCPIPC
     OPTIONS    XCF
     WRITER     CTVLAD
```

Figure A-4 CTRACE activation sample

```
 TRACE CT,ON,COMP=SYSTCPIP,SUB=(TCPIPC)
*92 ITT006A SPECIFY OPERAND(S) FOR TRACE CT COMMAND.
 R 92,WTR=DISCONNECT
 IEE600I REPLY TO 92 IS;WTR=DISCONNECT
*93 ITT006A SPECIFY OPERAND(S) FOR TRACE CT COMMAND.
 R 93,END
 IEE600I REPLY TO 93 IS;END
 ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
 WERE SUCCESSFULLY EXECUTED.
 IEE839I ST=(ON,0064K,00128K) AS=ON  BR=OFF EX=ON  MT=(ON,064K) 597
         ISSUE DISPLAY TRACE CMD FOR SYSTEM AND COMPONENT TRACE STATUS
         ISSUE DISPLAY TRACE,TT CMD FOR TRANSACTION TRACE STATUS
...

TRACE CT,OFF,COMP=SYSTCPIP,SUB=(TCPIPC)
ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
WERE SUCCESSFULLY EXECUTED.
IEE839I ST=(ON,0064K,00128K) AS=ON  BR=OFF EX=ON  MT=(ON,064K) 613
        ISSUE DISPLAY TRACE CMD FOR SYSTEM AND COMPONENT TRACE STATUS
        ISSUE DISPLAY TRACE,TT CMD FOR TRANSACTION TRACE STATUS
...

TRACE CT,WTRSTOP=CTVLAD
ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
WERE SUCCESSFULLY EXECUTED.
IEF196I AHL904I THE FOLLOWING TRACE DATASETS CONTAIN TRACE DATA :
IEF196I           LUNA.CTRACE
AHL904I THE FOLLOWING TRACE DATASETS CONTAIN TRACE DATA : 619
          LUNA.CTRACE
IEE839I ST=(ON,0064K,00128K) AS=ON  BR=OFF EX=ON  MT=(ON,064K) 623
        ISSUE DISPLAY TRACE CMD FOR SYSTEM AND COMPONENT TRACE STATUS
        ISSUE DISPLAY TRACE,TT CMD FOR TRANSACTION TRACE STATUS
ITT111I CTRACE WRITER CTVLAD TERMINATED BECAUSE OF A WTRSTOP REQUEST.
..
-JOBNAME  STEPNAME PROCSTEP    RC   EXCP   CONN    TCB    SRB  CLOCK
 SERV  PG  PAGE  SWAP   VIO SWAPS
-CTVLAD           IEFPROC     00    208    414    .00    .00   10.9
 4286   0     0     0     0     0
IEF404I CTVLAD - ENDED - TIME=16.37.36
-CTVLAD   ENDED.  NAME-                  TOTAL TCB CPU TIME=   .00
 TOTAL ELAPSED TIME=  10.9
IEF196I IEF142I CTVLAD CTVLAD - STEP WAS EXECUTED - COND CODE 0000
...

DISPLAY TRACE,COMP=SYSTCPIP,SUB=(TCPIPC)
IEE843I 18.58.52  TRACE DISPLAY 384
        SYSTEM STATUS INFORMATION
 ST=(ON,0064K,00128K) AS=ON  BR=OFF EX=ON  MT=(ON,064K)
  TRACENAME
  =========
  SYSTCPIP
                     MODE BUFFER HEAD SUBS
                     =====================
                     OFF         HEAD    3
     NO HEAD OPTIONS
  SUBTRACE           MODE BUFFER HEAD SUBS
 ------------------------------------------------------------
  TCPIPC             MIN  0004M
     ASIDS      *NONE*
     JOBNAMES   *NONE*
     OPTIONS    MINIMUM
     WRITER     *NONE*
```

Figure A-5 CTRACE deactivation sample

Packet trace (SYSTCPDA)

Packet tracing captures IP packets as they enter or leave TCP/IP. You select what you want to trace via the PKTTRACE statement within the PROFILE.TCPIP or via the VARY PKTTRACE command entered from the MVS console. RACF authorization is required to execute this command.

With the VARY PKTTRACE command or PKTTRACE statement in PROFILE.TCPIP, you can specify options such as IP address, port number, and protocol type. If you are planning to gather a trace for relatively long hours, or if your system experiences heavy traffic, it is recommended that you specify these filtering options so that TCP/IP does not have to gather unnecessary packets.

The following steps run a packet trace, and the data is written to an external writer:

1. Start the CTRACE external writer:

   ```
   TRACE CT,WTRSTART=ctwtr
   ```

2. Start the CTRACE and connect the external writer to the TCP/IP address space:

   ```
   TRACE CT,ON,COMP=SYSTCPDA,SUB=(tcpprocname)
   R nn,WTR=ctwtr,END
   ```

3. Check that the trace started successfully:

   ```
   D TRACE,COMP=SYSTCPDA,SUB=(tcpprocname)
   ```

4. Start the trace through the PROFILE.TCPIP statement and the VARY OBEYFILE command, or through the V TCPIP,,PKT command.

   ```
   VARY TCPIP,tcpprocname,PKT,ON
   ```

   The trace options may modify the data being captured using the VARY command. If, for example, you wish to abbreviate the data collected you may use:

   ```
   VARY TCPIP,tcpprocname,PKT,ABBREV
   ```

   Also issuing multiple VARY commands may be used to OR filters together. You may use it for multiple addresses or if, for example, you wish to record all packets with destination ports xxxx or source ports yyyy you may use the commands:

   ```
   VARY TCPIP,tcpprocname,PKT,DEST=xxxx
   VARY TCPIP,tcpprocname,PKT,SRCP=yyyy
   ```

5. Perform the operation that you want to trace.
6. Stop the trace:

   ```
   VARY TCPIP,tcpprocname,PKT,OFF
   ```

7. Disconnect the external writer from TCP/IP:

   ```
   TRACE CT,ON,COMP=SYSTCPDA,SUB=(tcpprocname)
   R nn,WTR=DISCONNECT,END
   ```

8. Stop the CTRACE:

   ```
   TRACE CT,OFF,COMP=SYSTCPDA,SUB=(tcpprocname)
   ```

9. Stop the external writer.

   ```
   TRACE CT,WTRSTOP=ctwtr
   ```

Examine the data with IPCS or send it to the Support Center if they have requested it.

Socket data trace

The data trace is used to trace socket data into and out of the Physical File System (PFS).

Follow the packet trace steps detailed above, but use the following commands to start and stop the data trace:

```
V TCPIP,tcpproc,DATTRACE,ON
V TCPIP,tcpproc,DATTRACE,OFF
```

OMPROUTE trace (SYSTCPRT)

OS/390 V2R6 IP and later provides component trace support for the OMPROUTE application. A default minimum component trace is always started during OMPROUTE initialization. To customize the parameters used to initialize the trace, update the SYS1.PARMLIB member CTIORA00. Besides specifying the trace options, you can also change the OMPROUTE trace buffer size. The buffer size can be changed only at OMPROUTE initialization.

After OMPROUTE initialization, you must use the TRACE CT command to change the component trace options.

To gather the component trace for OMPROUTE, use the commands listed in “Taking a component trace” on page 210, specify the component name of SYSTCPRT and your OMPROUTE proc_name, and follow the same procedures outlined.

Examine the data with IPCS using the steps identified in 5.6 “Analyzing a trace” or send it to the Support Center if they have requested it.

Resolver trace (SYSTCPRE)

CS for z/OS V1R2 provides component trace support for the Resolver. A default minimum component trace is always started during Resolver initialization. To customize the parameters used to initialize the trace, update SYS1.PARMLIB member CTIRES00. Besides specifying the trace options, you can also change the Resolver trace buffer size. The buffer size can be changed only at Resolver initialization.

After Resolver initialization, you must use the TRACE CT command to change component trace options.

To gather the component trace for the Resolver, use the commands listed in “Taking a component trace” on page 210 and specify a component name of SYSTCPRE and your Resolver proc_name.

Examine the data with IPCS or send it to the Support Center if they have requested it.

Intrusion detection services trace (SYSTCPIS)

When the TCP/IP stack starts, it reads SYS1.PARMLIB member CTIIDS00, which contains trace options for the SYSTCPIS trace. Packets are traced based on the IDS policy defined in LDAP.

Please see *z/OS V1R7.0 CS: IP Diagnosis*, GC31-8782, for details on the intrusion detection services trace, and *z/OS V1R7.0 CS: IP Configuration Guide*, SC31-8775, for information about defining policy.

Obtaining component trace data with a dump

If the TCP/IP or user’s address space abends, TCP/IP recovery dumps the home ASID, the primary ASID, the secondary ASID, and the TCPIPDS1 dataspace to the data sets defined within your MVS environment. The TCPIPDS1 dataspace contains the trace data for SYSTCPIP, SYSTCPDA, and SYSTCPIS components.

To obtain a dump of the TCP/IP stack when no abend has occurred, use the DUMP command. Remember to specify the data space name, which is always TCPIPDS1, since it contains the trace data for the SYSTCPIP, SYSTCPDA, and SYSTCPIS components. Be sure to include “region” in the SDATA dump options:

- DUMP COMM=(enter_dump_title_here)
- Rxx,JOBNAME=tcpproc,DSPNAME=(‘tcpproc’.TCPIPDS1),CONT
- Rxx,SDATA=(CSA,LSQA,NUC,PSA,RGN,SQA,SUM,SQA,TRT),END

To obtain a dump of the OMPROUTE, RESOLVER, or TELNET address space (which contains the trace table), use the DUMP command as follows:

```
DUMP COMM=(enter_dump _title_here)
Rxx,JOBNAME=proc_started_task_name,SDATA=(RGN,CSA,ALLPSA,SQA,SUM,TRT,ALLNUC),END
```

Analyzing a trace

You can format component trace records using IPCS panels or a combination of IPCS panels and the CTRACE command, either from a dump or from external writer files. You can also use IPCS in batch to print a component trace.

The primary purpose of the component trace is to capture data that the IBM Support Center may use in diagnosing problems. There is little information in the documentation on interpreting trace data. If you wish to analyze the packet trace or data trace by yourself, you can do so with IPCS, an example of the IP packet layout or format, and a booklet that allows you to interpret the hexadecimal data in EBCDIC or ASCII.

Using the IPCS panels

The code for the component trace formatter is in data set SYS1.MIGLIB. Ensure that your TSO logon procedure includes SYS1.MIGLIB in the STEPLIB concatenation.

```
------------------------ IPCS PRIMARY OPTION MENU  ----------------------------

 OPTION  ===>  0
                                                      ********************
    0  DEFAULTS    - Specify default dump and options    * USERID  - USER1
    1  BROWSE      - Browse dump data set                * DATE    - 98/03/26
    2  ANALYSIS    - Analyze dump contents               * JULIAN  - 98.085
    3  UTILITY     - Perform utility functions           * TIME    - 16:14
    4  INVENTORY   - Inventory of problem data           * PREFIX  - USER1
    5  SUBMIT      - Submit problem analysis job to batch* TERMINAL- 3278
    6  COMMAND     - Enter subcommand, CLIST or REXX exec* PF KEYS - 12
    T  TUTORIAL    - Learn how to use the IPCS dialog    ********************
    X  EXIT        - Terminate using log and list defaults

 Enter END command to terminate IPCS dialog
   Message Control ==> CONFIRM VERIFY FLAG(WARNING)
   Display Content ==> NOMACHINE REMARK REQUEST NOSTORAGE SYMBOL

 Press ENTER to update defaults.

Command ===>
```

Figure A-6 IPCS main panel

First we specify the name of our trace data set on the IPCS defaults panel. See **1** in Figure A-7 on page 218.

```
------------------------ IPCS Default Values ------------------ LOCAL

Command ===>

  You may change any of the defaults listed below.  The defaults shown
  any changes are LOCAL.  Change scope to GLOBAL to display global def

  Scope   ==> LOCAL   (LOCAL, GLOBAL, or BOTH)

  If you change the Source default, IPCS will display the current defa
  Address Space for the new source and will ignore any data entered in
  the Address Space field.

  Source  ==> DSNAME('TCP.CTRACE1.TRACE01') 1
  Address Space   ==> RBA
  Message Routing ==> NOPRINT TERMINAL
  Message Control ==> NOCONFIRM VERIFY FLAG(WARNING)
  Display Content ==> NOMACHINE REMARK REQUEST NOSTORAGE SYMBOL

Press ENTER to update defaults.

Use the END command to exit without an update.
```

Figure A-7 Changing IPCS defaults

After we return to the primary IPCS panel, we enter:

- 2 for ANALYSIS
- 7 for TRACES
- 1 for CTRACE

Once you arrive at the CTRACE primary option menu, you simply proceed through the menu selections, specifying Q first, then on the parameter panel displayed in Figure A-8 on page 219 enter report type FULL and start formatting with S. See the trace shown in Figure A-9 on page 219.

```
--------------------- CTRACE QUERY PARAMETERS  ----------------------

COMMAND ===>  S

  Enter/verify CTRACE QUERY parameters below:

  System      ===>           (System name or blank)
  Component   ===>           (Blank for all active components)
  Subnames    ===>

  Report type ===> FULL      (Short or Full, short is default)

  GMT/LOCAL   ===> G         (G or L, GMT is default)
  Subname
  entry panel ===>
  Override
    source    ===>

  CTRACE QUERY FULL


  END/PF3 = return to CTRACE primary options panel.
  S = start CTRACE.  R = reset all fields.
```

Figure A-8 Specifying query parameters and starting CTRACE formatting

```
IPCS OUTPUT STREAM ----------------------------------------- Line 0 C

Command ===>                                                SCROLL =
 ****************************** TOP OF DATA ************************



 COMPONENT TRACE QUERY SUMMARY


       SYSNAME  COMPONENT SUB NAME
       -------- --------- --------
 0001. MVSVIC97 SYSTCPDA TCPIPC 1
```

Figure A-9 Displaying the trace formatting parameters

Now that you know the TCP/IP name (TCPIPC 1 in Figure A-9), you are prepared to execute the next step. Back out of CTRACE processing by pressing PF3 twice and select D to enter the formatting parameters in Figure A-10 on page 220. Fill in the name of the TCP/IP proc 1 that was traced and that you want a FULL report 2 for.

```
-------------------- CTRACE DISPLAY PARAMETERS  -----------------------

 COMMAND ===>  S

   System      ===>           (System name or blank)
   Component   ===> SYSTCPDA   4 (Component name (required))
   Subnames    ===> TCPIPC 1

   GMT/LOCAL   ===> l                        (G or L, GMT is default)
   Start time  ===>                          (mm/dd/yy,hh:mm:ss.dddddd
   Stop time   ===>                           mm/dd/yy,hh.mm.ss.dddddd)
   Limit       ===> 0         Exception ===>
   Report type ===> FULL 2  (SHort, SUmmary, Full, Tally)
   User exit   ===>           (Exit program name)
   Override source ===>
   Options         ===>  3

   To enter/verify required values, type any character
   Entry IDs ===>   Jobnames ===>   ASIDs ===>   OPTIONS ===>   SUBS ==




   ENTER = update CTRACE definition.  END/PF3 = return to previous pane
   S = start CTRACE.  R = reset all fields.
```

Figure A-10 Specifying TCP/IP name and full report

In the Component field 4, specify:

- SYSTCPDA for packet trace or data trace
- SYSTCPIP for TCP/IP stack trace and TN3270 Telnet server in its own space
- SYSTCPRT for OMPROUTE trace
- SYSTCPRE for the Resolver trace
- SYSTCPIS for intrusion detection services trace

In Options 3 specify DATATRACE for a data trace or PACKETTRACE for a packet trace. You can use the Options field to filter the display. For example, if you have a packet trace and you only want to display packets for port 23, you can specify PACKETTRACE PORT(23); or if you have a TCP/IP event trace and you only want to display events related to a particular IP address you can specify IPADDR(ip_addr). See *z/OS V1R7.0 CS: IP Diagnosis*, GC31-8782, for a full list of available options.

In CS for z/OS V1R2, the CTRACE packet trace formatter has been rewritten. The example below shows one packet that has been formatted using the new packet trace formatter.

```
COMPONENT TRACE FULL FORMAT
 SYSNAME(SC64)
 COMP(SYSTCPDA)SUBNAME((TCPIPA))
 OPTIONS((PACKETTRACE))
 OS/390 TCP/IP Packet Trace Formatter, (C) IBM 2000, 2001.134
 DSNAME('WOODS.CTRACE1')

**** 2001/09/24
RcdNr Sysname  Mnemonic Entry Id   Time Stamp    Description
----- -------- -------- -------- -------------- ---------------------------
40 SC64     PACKET   00000001 17:55:02.444686 Packet Trace
 To Link           : OSA22E0LINK      Device: QDIO Ethernet    Full=71
  Tod Clock        : 2001/09/24 17:55:02.444686
  Lost Records     : 0                Flags: Pkt Ver2 Adj Out
  Source Port      : 23               Dest Port: 1160  Asid: 004C TCB: 00000000
 IpHeader: Version : 4                Header Length: 20
  Tos              : 00               QOS: Routine Normal Service
  Packet Length    : 71               ID Number: 050C
  Fragment         :                  Offset: 0
  TTL              : 64               Protocol: TCP           CheckSum: F34F FFFF
  Source           : 10.12.6.60
  Destination      : 10.24.105.246

 TCP
  Source Port      : 23    (telnet)   Destination Port: 1160  ()
  Sequence Number  : 2366825366       Ack Number: 2895763969
  Header Length    : 20               Flags: Ack Psh
  Window Size      : 32765            CheckSum: B735 FFFF Urgent Data Pointer: 0000

Telnet: 31
 0000 IAC,WILL,SUPPRESS GO AHEAD,(data=28);

IP Header         : 20     IP: 10.12.6.60, 10.24.105.246
000000 45000047 050C0000 4006F34F 090C063C  091869F6
Protocol Header   : 20     Port:    23,  1160
000000 00170488 8D12E396 AC99DA01 50187FFD  B7350000

Data              : 31     Data Length: 31
000000 FFFB030D 0A494B4A 35363730 30412045 |.......¢........ .....IKJ56700A E|
000010 4E544552 20555345 52494420 3D0D0A   |+..............  NTER USERID =.. |
```

Figure A-11 Formatted packet

Using IPCS and the CTRACE command

If you prefer to bypass most of the IPCS panels, you can enter IPCS, set your defaults (Option 0), and go to command mode (Option 6). Then use the CTRACE command to format the trace data. For example:

```
CTRACE COMP(component) SUB((proc_name)) FULL
CTRACE COMP(SYSTCPDA) SUB((TCPIPA)) FULL OPTIONS((PACKETTRACE PORT(23)))
```

For more information about the CTRACE command see *z/OS V1R7.0 CS: IP Diagnosis*, GC31-8782.

Printing a component trace

If you want to print a component trace, you may do so with IPCS. See Figure A-12 on page 222 for a sample batch IPCS job to format a TCP/IP component trace.

```
//jobname  JOB  ...
//IEFPROC  EXEC PGM=IKJEFT01,REGION=4M,DYNAMNBR=10
//STEPLIB      DD    DSN=SYS1.MIGLIB,DISP=SHR
//IPCSPARM  DD    DSN=SYS1.PARMLIB,DISP=SHR
//IPCSDDIR   DD     DSN=userid.DDIR,DISP=SHR
//SYSPROC   DD    DSN=SYS1.SBLSCLIO,DISP=SHR
//SYSTSPRT DD    SYSOUT=*
//IPCSTOC    DD    SYSOUT=*
//IPCSPRNT  DD    SYSOUT=*
//SYSTSIN     DD     *
 IPCS
 MERGE
 CTRACE DSN('your.trace.dataset1') -
     COMP(component) SUB(procname) FULL -
     OPTIONS((option1,option2))
 CTRACE DSN('your.trace.dataset2') -
     COMP(component) SUB(procname) FULL -
     OPTIONS((option1,option2))
 MERGEEND
 END
/*
```

Figure A-12 Batch IPCS formatting job

The *component* and *procname* are the same values you specified when you took the component trace. The *options* vary depending on the type of component trace. A SNIFFER option exists to allow trace records to be written in a format acceptable for downloading to other trace analysis programs. Please refer to *z/OS V1R7.0 CS: IP Diagnosis*, GC31-8782, for details on the options available for each trace. IPCS requires member IPCSPR00 to be present in the data set referenced by DD name IPCSPARM (usually SYS1.PARMLIB). See *z/OS V1R7.0 MVS Initialization and Tuning Reference*, SA22-7592, for details on what to define in IPCSPR00.

Useful formats

Figure A-13 on page 223, Figure A-14 on page 223, and Figure A-15 on page 223 show the format of an IP diagram, TCP header, and UDP header, respectively. Understanding these formats is important when analyzing trace data.

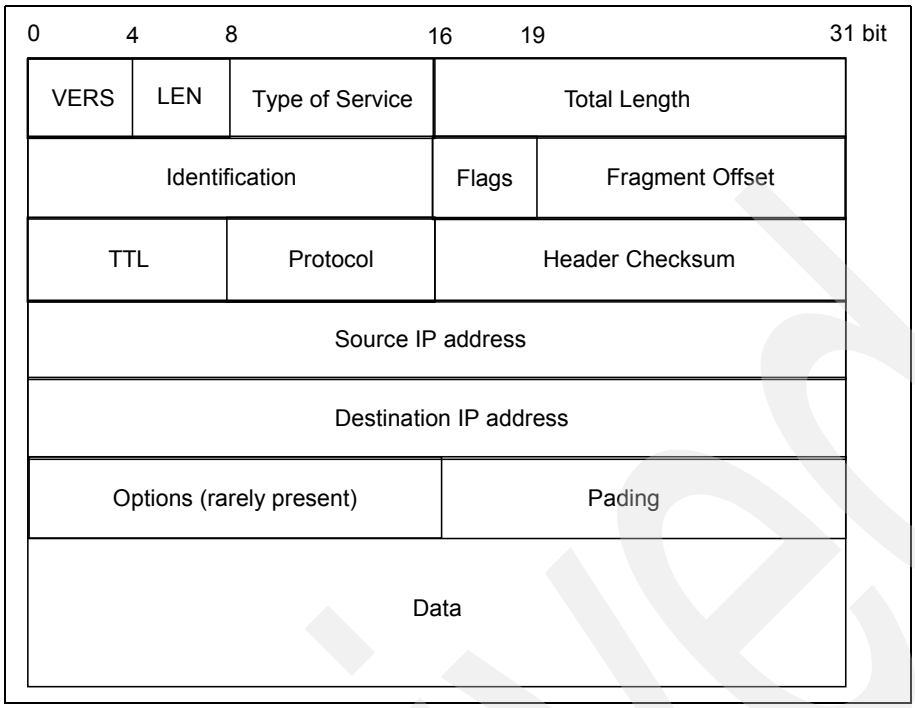


Figure A-13 Format of IP datagram

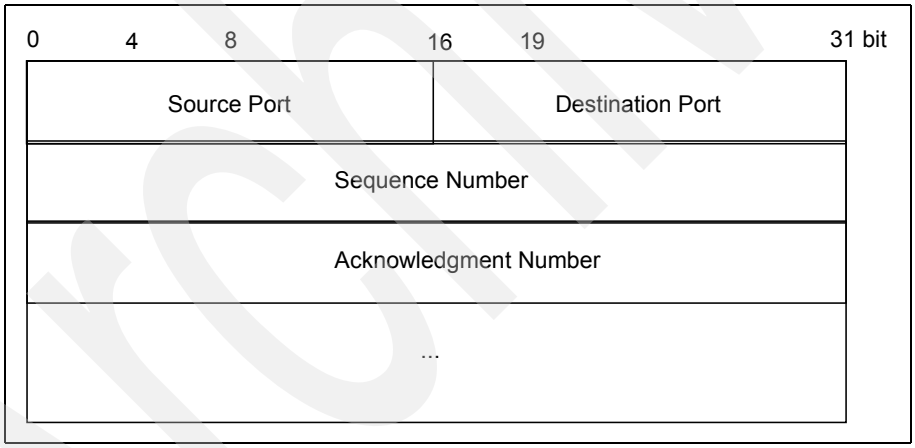


Figure A-14 Format of TCP header

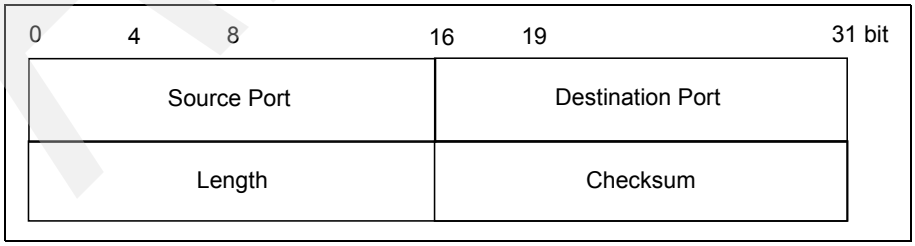


Figure A-15 Format of UPD header

Processing IPCS dumps

z/OS Communications Server IP provides an IPCS subcommand to format an IPCS dump. The TCPIPCS subcommand is documented in *z/OS V1R7.0 CS: IP Diagnosis*, GC31-8782.

Configuration profile trace

You can use the ITRACE statement in the PROFILE.TCPIP data set to activate TCP/IP runtime tracing for configuration, the TCP/IP SNMP subagent, commands, and the autolog subtask. ITRACE should only be set at the direction of an IBM Service representative. For more information, please refer to *z/OS V1R7.0 CS: IP Diagnosis*, GC31-8782.

B

Additional parameters and functions

This appendix contains examples and a discussion of the following:

- System symbolics
- PROFILE.TCPIP parameters
- TCP/IP built-in security functions

System symbolics

One of the many strengths of the z/OS technology is that it allows multiple TCP/IP stacks (instances) to be configured in the same MVS system or across multiple MVS systems. If we need to run many stacks we need to ensure that each profile configuration data set is unique. For example, if you are running your TCP/IP stacks in a sysplex, you would need to maintain one configuration for each stack on each of the systems. As more systems are added to the sysplex, more TCP/IP configuration files need to be maintained and synchronized.

Here in the ITSO we used system symbolics to enable us to share the definitions between the LPARs (MVS images), so all TCPIPAs will share the definitions across SC30, SC31, and SC32. System symbolics are used in creating shared definitions for systems that are in a sysplex. With this facility, you use the symbols defined during system startup as variables in configuring your TCP/IP stack. This means that you only need to create and maintain a template file for all the systems in the sysplex.

System symbols processing

System symbols used in the PROFILE, OBEYFILE, and INCLUDE files are automatically translated during stack initialization. However, if you are planning to use it on the other TCP/IP data sets such as TCPIP.DATA, you need to run a JCL-driven utility called EZACFSM1 located in hlq.SEZAINST(CONVSYM). See Example B-1.

Example: B-1 JCL for EZACFSM1

```
//CONVSYM  JOB (accounting,information),programmer.name,
//            MSGLEVEL=(1,1),MSGCLASS=A,CLASS=A
//*
//STEP1   EXEC  PGM=EZACFSM1,REGION=0K
//SYSIN   DD DSN=TCP.DATA.INPUT,DISP=SHR
//*SYSIN   DD    PATH='/tmp/tcp.data.input'
//*          The input file can be either an MVS file or a z/OS UNIX filesystem file.
//*
//*
//*
//SYSOUT  DD DSN=TCP.DATA.OUTPUT,DISP=SHR
//*SYSOUT  DD
// PATH='/tmp/tcp.data.output',PATHOPTS=(OWRONLY,OCREAT),
//*              PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIWGRP)
```

The input to EZACFSM1 is your template data set that contains the system symbols and the definitions that you need. The output data set will be the parameter files, such as TCPIP.DATA, that the TCP/IP stack or CS for z/OS IP application will use during its startup and operation. You need to run the utility on each of the systems where you need to have the symbols translated.

Symbols definitions

The variable &SYSCLONE is defined in the IEASYMxx member of SYS1.PARMLIB. As shown in Example B-2 on page 227, the value for &SYSCLONE is derived from &SYSNAME. The variable &SYSNAME could be defined either in the IEASYSxx member or in the LOADxx member used during IPL. In our case, &SYSNAME was defined in IEASYSxx, which we used to IPL our MVS images. Refer to Example B-3 on page 227 for a sample of the IEASYSxx that we used for the startup of SC30. You can find further information about system symbols in *z/OS V1R7 MVS Initialization and Tuning Guide*, SA22-7591.

Example: B-2 &SYSCLONE definition in SYS1.PARMLIB

```
SYSDEF          SYSCLONE(&SYSNAME(3:2)) 1
       SYMDEF(&SYSR2='037RZ1')
                 SYMDEF(&SYSR3='&SYSR2(1:5).2')
                 SYMDEF(&SYSR4='&SYSR2(1:5).3')
```

1 The value of SYSCLONE is defined as two characters starting from the third character of SYSNAME.

Example: B-3 IEASYSxx definition

```
Following is the definition in SYS1.PARMLIB(IEASYS00)

COUPLE=00,
OMVS=7A,        z/OS UNIX initialization member BPXPRM7A
PROD=01,
PROG=(A0,S0,D0,C1,L0),  Authorization list
SMF=00,
SMS=00,
SYSNAME=SC30,            same as SID in SMFPRM00
SSN=00,
VAL=00,
SYSNAME is defined as SC30 for system SC30
```

You can also define and use your own variable in configuring CS for z/OS IP aside from &SYSNAME or &SYSCLONE. Refer to the *z/OS Communications Server: IP Configuration Guide, Version 1 Release 7*, SC31-8775-07, for information about creating symbols output data set.

System symbols in TCPIP.PROFILE

In Example B-4 we show the common configuration profile used for the A-stacks in systems SC30, SC31, and SC32. Since &SYSCLONE is unique in each system, it ensures that the files and IDs that will be generated when the stacks initialize are also unique.

Example: B-4 Use of system symbols in our TCP/IP profile

```
;*************************************************************
;This is the common profile used between the E stack on SC30
; SC31 and SC32
;*************************************************************
ARPAGE 20
;
GLOBALCONFIG NOTCPIPSTATISTICS
;
IPCONFIG DATAGRAMFWD SYSPLEXROUTING
;
DYNAMICXCF 10.10.20.1&SYSCLONE 255.255.255.0 8 1
;
SOMAXCONN 240
;
TCPCONFIG TCPSENDBFRSIZE 16K TCPRCVBUFRSIZE 16K SENDGARBAGE FALSE
TCPCONFIG RESTRICTLOWPORTS
;
UDPCONFIG RESTRICTLOWPORTS
;
;*************************************************************
; Include the stack specific device and home definitions
; the devices are also started (home30, home31, home32)
```

```
;*****************************************************************
INCLUDE TCPIPE.TCPPARMS(HOME&SYSCLONE) 4
;
;*****************************************************************
; include the BEGINROUTES atatic definitions for OMPROUTE
; in tcpipe.tcpparms(static30, static31, static32)
;*****************************************************************
INCLUDE TCPIPE.TCPPARMS(STATIC&SYSCLONE) 6
;
;*****************************************************************
; start the ftp daemon in each of the E stack
;*****************************************************************
AUTOLOG 5
   FTPDE&sysclone  JOBNAME FTPDE&sysclone.1 2
;  OMP&sysclone             ; OSPF daemon
;  SMTP                     ; SMTP Server
ENDAUTOLOG
;
PORT 3
    20 TCP *  NOAUTOLOG     ; FTP Server
    21 TCP OMVS    BIND 10.10.10.210; control port
    23 TCP INTCLIEN           ; MVS Telnet Server
    23 TCP OMVS BIND 10.10.10.210 ;Telnet Server
;   25 TCP SMTP               ; SMTP Server
;  443 TCP HTTPS              ; http protocol over TLS/SSL
;  443 UDP HTTPS              ; http protocol over TLS/SSL
; 3389 TCP MSYSLDAP           ; LDAP Server for Msys
;
SACONFIG ENABLED COMMUNITY public AGENT 161
;
; tcpipe.tcpparms(TELN&sysclone) has the VTAM defnitions for
; the various stacks (TELN30, TELN31, TELN32)
;
INCLUDE TCPIPE.TCPPARMS(TELN&SYSCLONE) 5
```

1 sets a unique IP address for the dynamic XCF definitions.

2 autologs the FTPD server daemon.

Note: A dot is needed at the end of &SYSCLONE because the next character is not a space.

3 Port reservation for each server.

4 Include file for system-specific device definitions.

5 Common include file for TELNET and VTAM definitions.

6 Common include for static routes used by SC30, SC31 and SC32.

Important: The system symbols are stored in uppercase by MVS. Because you can code the TCP/IP configuration statements in either uppercase or lowercase, you have to make sure that you code the system symbol name in uppercase. If the symbols defined in the profile are not in uppercase, the symbols will not get translated and, depending on the parameter, you may or may not get an error message, but you would certainly encounter some unusual and unexpected names.

Include files

Together with the system symbolics support, we also used a facility (called INCLUDE) to help us organize and share our stack configuration. By using the include configuration statement, we were able to structure our configuration better by putting different sections of PROFILE.TCPIP in separate files. During the stack's initialization, the contents of the file pointed to by the include statement are read and processed. These include statements are treated as though they were coded in PROFILE.

Because the IP address used by each TCP/IP stack is unique, we used the variable &SYSCLONE (refer to Example B-4 on page 227) to resolve the name of the system-specific device file. We have included all of the DEVICE, LINK, and START statements for each of the devices in this file. Example B-5 shows the device file for system SC30 called HOME30. We also have a separate device file each for systems SC31 and SC32.

Example: B-5 Included device file HOME30 for SC30

```
;************************************************************
;member tcpipe.tcpparms(home30)
;This is the member with device and home definitions for stack
; on SC30 MVS image
;************************************************************
;Osa definitions... relates to sys1.vtamlst trle definitions
;major nodes : osa2080,osa20a0,osa20c0 and osa20e0
 DEVICE OSA2080  MPCIPA
 LINK   OSA2080LNK  IPAQENET      OSA2080 VLANID 10
 DEVICE OSA20A0  MPCIPA
 LINK   OSA20A0LNK  IPAQENET      OSA20A0 VLANID 11
 DEVICE OSA20C0  MPCIPA
 LINK   OSA20C0LNK  IPAQENET      OSA20C0 VLANID 10
 DEVICE OSA20E0  MPCIPA
 LINK   OSA20E0LNK  IPAQENET      OSA20E0 VLANID 11
 ;
 ;Hipersocks definitions dynamically created on vtam
 DEVICE IUTIQDF4 MPCIPA
 LINK   IUTIQDF4LNK  IPAQIDIO      IUTIQDF4
 DEVICE IUTIQDF5 MPCIPA
 LINK   IUTIQDF5LNK  IPAQIDIO      IUTIQDF5
 DEVICE IUTIQDF6 MPCIPA
 LINK   IUTIQDF6LNK  IPAQIDIO      IUTIQDF6
;
; Static VIPA definitions
 DEVICE STAVIPA1    VIRTUAL 0
 LINK   STAVIPA1LNK VIRTUAL 0     STAVIPA1
;
HOME
   10.10.4.214   IUTIQDF4LNK
   10.10.4.215   IUTIQDF5LNK
   10.10.5.216   IUTIQDF6LNK
   10.10.10.210   STAVIPA1LNK
   10.10.2.212    OSA2080LNK
   10.10.3.213    OSA20A0LNK
   10.10.2.214    OSA20C0LNK
   10.10.3.215    OSA20E0LNK
;
START IUTIQDF4
START IUTIQDF5
START IUTIQDF6
START OSA2080
START OSA20A0
START OSA20C0
```

```
START OSA2OE0
```

In our configuration, the three TCP/IP A-stacks were designed to have common Telnet and VTAM characteristics. This decision helped us to use and share the same file (Example B-6) to define the Telnet and VTAM configuration for each of the TCP/IP stack. The variable &SYSCLONE is then used in the VTAM common configuration file to further customize the setting for each stack in the different systems.

Example: B-6 Common Telnet include file

```
;*****************************************************************
;member tcpipe.tcpparms(teln30)
;This is the vtam definitions for stack E on MVS image 30
;*****************************************************************
TelnetParms
  Port 23                         ; Port number 23 (std.)
  TELNETDEVICE 3278-3-E NSX32703 ; 32 line screen -
                                  ; default of NSX32702 is 24
  TELNETDEVICE 3279-3-E NSX32703 ; 32 line screen -
                                  ; default of NSX32702 is 24
  TELNETDEVICE 3278-4-E NSX32704 ; 48 line screen -
                                  ; default of NSX32702 is 24
  TELNETDEVICE 3279-4-E NSX32704 ; 48 line screen -
                                  ; default of NSX32702 is 24
  TELNETDEVICE 3278-5-E NSX32705 ; 132 column screen-
                                  ; default of NSX32702 is 80
  TELNETDEVICE 3279-5-E NSX32705 ; 132 column screen -
                                  ; default of NSX32702 is 80
  LUSESSIONPEND     ; On termination of a Telnet server connection,
                    ; the user will revert to the DEFAULTAPPL
                    ; instead of having the connection dropped

  MSG07             ; Sends a USS error message to the client if an
                    ; error occurs during session establishment
                    ; instead of dropping the connection
  CodePage ISO8859-1 IBM-1047   ; Linemode ASCII, EBCDIC code pages
  Inactive 0                    ; Let connections stay around
  PrtInactive 0                 ; Let connections stay around
  TimeMark 600
  ScanInterval 120
; SMFinit std
; SMFterm std
; Define logon mode tables to be the defaults shipped with the
  ; latest level of VTAM

EndTelnetParms
;
BeginVTAM
  Port 23
  ; Define the LUs to be used for general users.
  DEFAULTLUS
    SC&SYSCLONE.TSO1..SC&SYSCLONE.TS30 1
  ENDDEFAULTLUS
; DEFAULTAPPL TSO   ; Set the default application for all TN3270(E)
                    ; Telnet sessions to TSO

  LINEMODEAPPL TSO  ; Send all line-mode terminals directly to TSO.

  ALLOWAPPL SC*  DISCONNECTABLE ; Allow all users access to TSO
  ALLOWAPPL TSO* DISCONNECTABLE ; Allow all users access to TSO
```

```
              ; applications.
              ; TSO is multiple applications all beginning with TSO,
              ; so use the * to get them all.  If a session is closed,
              ; disconnect the user rather than log off the user.

  ALLOWAPPL *      ; Allow all applications that have not been
                   ; previously specified to be accessed.

  RESTRICTAPPL IMS ; Only 3 users can use IMS.
    USER USER1     ; Allow user1 access.
      LU TCPIMS01  ; Assign USER1 LU TCPIMS01.
    USER USER2     ; Allow user2 access from the default LU pool.
    USER USER3     ; Allow user3 access from 3 Telnet sessions,
                   ; each with a different reserved LU.
      LU TCPIMS31 LU TCPIMS32 LU TCPIMS33

  USSTCP USSTEST1
EndVTAM
```

Take note of 1 in Example B-6 on page 230. For illustration purposes, we used a different case for the &SYSCLONE variable and we got the error 2 in Example B-7 during the startup of the stack in SC30. Changing the case corrected the problem.

Example: B-7 Startup error messages due to wrong case for &SYSCLONE

```
IEF403I  TCPIPE - STARTED - TIME=09.15.05
 IEE252I  MEMBER CTIEZB01 FOUND IN SYS1.PARMLIB
 EZZ7450I FFST SUBSYSTEM IS NOT INSTALLED
 EZZ0300I OPENED PROFILE FILE DD:PROFILE
 EZZ0309I PROFILE PROCESSING BEGINNING FOR DD:PROFILE
 EZZ0323I AUTOLOG STATEMENT ON LINE 72 HAD NO ENTRIES
 EZZ0300I OPENED TCPIPE.TCPPARMS(HOME30) FILE
 EZZ0309I PROFILE PROCESSING BEGINNING FOR TCPIPE.TCPPARMS(HOME30)
 EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'TCPIPE.TCPPARMS(HOME30)'
 EZZ0304I RESUMING PROCESSING OF FILE DD:PROFILE
    EZZ0300I OPENED TCPIPE.TCPPARMS(TELN30) FILE
 EZZ0309I PROFILE PROCESSING BEGINNING FOR TCPIPE.TCPPARMS(TELN30)
 EZZ0401I LIST IS EMPTY IN FILE: 'TCPIPE.TCPPARMS(TELN30) ON LINE: 2
         57 AT: 'SC&SYSCLONE'
 EZZ0401I SYNTAX ERROR IN FILE: 'TCPIPE.TCPPARMS(TELN30)' ON LINE:  2
         57 AT: 'SC&SYSCLONE'
 EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'TCPIPE.TCPPARMS(TELN30)
```

PROFILE.TCPIP parameters

The syntax for the parameters in the PROFILE can be found in *z/OS V1R7.0 CS: IP Configuration Reference,* SC31-8776.

SOURCEVIPA

Applications within the network use the source IP address of a datagram to determine the source application of the datagram. When sourcevipa is set, outbound datagrams use the corresponding virtual IP address (VIPA) in the HOME statement list instead of the physical interface IP address. Sourcevipa has no effect on RIP servers such as OMPROUTE. The order of the HOME list is important if IPCONFIG SOURCEVIPA is specified. Tolerance of device and adapter failures may be achieved by using the sourcevipa option. Refer to *z/OS Communications Server: IP Configuration Reference, Version 1 Release 7*, SC31-8776-08,

for precautions when either the VIPA address or a physical adapter used as a source for the VIPA has an IP address that is the network address.

TCPIPSTATISTICS

Prints the values of several TCP/IP counters to the output data set designated by the CFGPRINT JCL statement. These counters include the number of TCP retransmissions and the total number of TCP segments sent from the MVS TCP/IP system. These TCP/IP statistics are written to the designated output data only during termination of the TCP/IP address space.

The TCPIPSTATISTICS parameter is confirmed by the message:

```
EZZ0613I TCPIPSTATISTICS IS ENABLED
```

This parameter should be specified in the GLOBALCONFIG section. Note that the SMFCONFIG TCPIPSTATISTICS parameter serves a different purpose. It requests that SMF records of subtype 5 containing TCP/IP statistics be created.

Reserving ports

The PORT reservations that are defined in the PROFILE data set are the ports that are used by specific applications. You may decide to explicitly not reserve all well-known ports by defining the UNRESTRICTLOWPORTS option on the TCPCONFIG and UDPCONFIG statements. This would allow any socket application to acquire a well-known port.

Example: B-8 PROFILE.TCPIP UNRESTRICTLOWPORTS statement

```
TCPCONFIG
UNRESTRICTLOWPORTS

UDPCONFIG
UNRESTRICTLOWPORTS
```

If you want the well-known ports to be used only by predefined application processes or superuser authorized application processes, then you can define the RESTRICTLOWPORTS option on the TCPCONFIG and UDPCONFIG statements. This prevents any non-authorized socket application from acquiring a well-known port. You then need to explicitly define PORT statements to reserve each port, or define the process with superuser authority in RACF. You may reserve the PORT or PORTRANGE by using the keyword OMVS, jobname of the process, or a wild card jobname such as *. UNIX applications may fork() another address space with a different name (for example, inetd or FTPD). You would need to reserve the port using the new address space name.

Example: B-9 PROFILE.TCPIP: PORT & PORTRANGE

```
TCPCONFIG
   RESTRICTLOWPORTS

UDPCONFIG
   RESTRICTLOWPORTS

PORT
;   20 TCP FTPDE1   NOAUTOLOG ; FTP Server  2
    20 TCP OMVS      NOAUTOLOG ; FTP Server  3
    21 TCP FTPDE1                           ; FTP Server  4
;   23 TCP INTCLIEN                        ; Telnet Server
    25 TCP SMTP                               ; SMTP Server
;   23 TCP INTCLIEN                        ; Telnet Server
    514 UDP OMVS                            ; UNIX SyslogD Server   5
```

```
;
PORTRANGE 10000 2000 TCP OMVS    ; TCP 10000 - 11999    7
PORTRANGE 10000 2000 UDP OMVS    ; UDP 10000 - 11999    7
```

Normally you can specify either OMVS or the job name in the PORT statement. However, certain daemons have special considerations on this matter.

When FTP starts it forks the listener process to run in the background, requiring that the name of the forked address space (FTPDE1 in this example), not the original procedure name, be used on the PORT statement of the control connection 4. You must specify OMVS as the name on the PORT for FTP's PORT 20 3, which is used for the data connection managed by the child process. If you specify the forked name on the data connection (Port 20, 2), the data connections will fail.

As you see, this can be a confusing issue. We recommend that you research each daemon that you implement; if something does not work, then try the alternative coding. You may read more about this subject in *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 4 - Security*, SG24-7172.

Note that we also reserve UDP port 514 5 to OMVS. This port is used by the SyslogD server in OMVS to receive log messages from other SyslogD servers in the TCP/IP network.

In addition to assigning port numbers to servers, you also need to reserve the same range of ephemeral port numbers that was reserved on the NETWORK statement in BPXPRMxx.

7 These PORTRANGE statements reserve a range of ephemeral TCP and UDP ports for UNIX System Services. In Example B-9 on page 232, ports 10000 to 11999 are reserved. The range must match the INADDRANYPORT and INADDRANYCOUNT in your BPXPRMxx member 8 (see Example B-10).

Example: B-10 What the PORTRANGE value in PROFILE should match

```
NETWORK DOMAINNAME(AF_INET)
        DOMAINNUMBER(2)
        MAXSOCKETS(10000)
        TYPE(INET)
        INADDRANYPORT(10000)   8
        INADDRANYCOUNT(2000)   8

NETWORK DOMAINNAME(AF_INET6)
DOMAINNUMBER(19)
MAXSOCKETS(10000)
TYPE(INET)
```

To display the PORT reservation list you can use the TSO/E command NETSTAT PORTL, MVS command **D TCPIP,procname,NETSTAT PORTL**, or UNIX shell command **onetstat -p tO3atcp -o**.

Example: B-11 Viewing port reservation list

```
D TCPIP,TCPIPE,N,PORTL
EZD0101I NETSTAT CS V1R7 TCPIPE 152
PORT# PROT USER     FLAGS    RANGE       SAF NAME
00020 TCP  OMVS        D
00021 TCP  FTPDE1    DABU
      BINDSPECIFIC: 10.10.10.210
00023 TCP  OMVS     DABU
      BINDSPECIFIC: 10.10.10.210
```

```
00023 TCP  TCPIPE   DAU
00025 TCP  SMTP     DA
00514 UDP  OMVS     DA
6 OF 6 RECORDS DISPLAYED
END OF THE REPOR
```

Port sharing (TCP only)

If you want to run multiple instances of a listener for performance reasons, you can share the same port between them. TCP/IP will select the listener with the fewest connections (both active and in the backlog) at the time when a client request comes in. A typical application using this feature is the Internet Connection Secure Server. If the load gets high, additional servers are started by the Workload Manager.

An example of a shared port is:

```
PORT
    80   TCP  WEBSRV1 SHAREPORT
    80   TCP  WEBSRV2
    80   TCP  WEBSRV3
```

BIND control for INADDR_ANY

A new keyword in the PORT statement was introduced in CS for OS/390 V2R10 IP: the BIND keyword. (CS for OS/390 V2R8 IP provides this support via APAR.) It associates the server job name with a specific IP address when the server binds to INADDR_ANY. This new function can be used to change the BIND for INADDR_ANY to a BIND for a specific IP address.

Telnet, for example, is a server that binds to INADDR_ANY. Previously, an OS/390® client that wants to access both Telnet servers, Telnet 3270 and UNIX Telnet, would connect to different ports or different TCP/IP stacks, depending on which Telnet server it wished to connect to. This led to cases where either one server used a different, nonstandard port or multiple TCP/IP stacks had to be used. With this function you do not need to have two different ports or TCP/IP stacks. You use the same port 23 for both Telnet 3270 and UNIX Telnet. All that is needed is to code the BIND keyword in the PORT statement for each server:

```
PORT
   23 TCP  INTCLIEN BIND 10.10.10.210
   23 TCP  OMVS BIND 10.10.1.211
```

In this case, the INTCLIEN is the special job name associated with the TN3270 server, since it actually runs in the TCP/IP address space. The OMVS job name identifies any UNIX server, including the UNIX Telnet server.

Both IP addresses can be dynamic VIPA addresses, static VIPA addresses, or real interface addresses. You also can code a wild card for the job name. Note that this function will work only for servers that bind to INADDR_ANY and is not valid with the PORTRANGE statement.

ARPTO

IPCONFIG ARPTO and ARPAGE statements have the same function: They specify the time interval between creation or revalidation and deletion of an entry in the ARP table. The value of IPCONFIG ARPTO is specified in seconds, and the value of ARPAGE is specified in minutes. ARP cache entries for MPCIPA and MPCOSA are not affected by ARPTO because they are not managed by the TCP/IP stack. For more information about devices that are affected by ARPTO refer to *z/OS Communications Server: IP Configuration Guide, Version 1 Release 7*, SC31-8775-07.

The UNIX shell command **onetstat -R** displays the current ARP cache entries. The capital R in the option is required for this display. A third parameter may be coded that would specify the IP address of the entry you wish to display, as the **NETSTAT ARP ip_addr** command does from TSO. If you wish to display the entire ARP cache, you can specify the third parameter with the reserved word ALL (again, all in capital letters). If you do not specify in capital letters, the reserved word is not recognized (see Example 6-13).

Example 6-13 ARP display

```
D TCPIP,TCPIPE,N,ARP
EZD0101I NETSTAT CS V1R7 TCPIPE 755
QUERYING ARP CACHE FOR ADDRESS 10.10.2.212
LINK: OSA2080LNK        ETHERNET: 00096B1A7490
QUERYING ARP CACHE FOR ADDRESS 10.10.3.213
LINK: OSA20A0LNK        ETHERNET: 00096B1A74C2
QUERYING ARP CACHE FOR ADDRESS 10.10.3.223
LINK: OSA20A0LNK        ETHERNET: 00096B1A74C2
QUERYING ARP CACHE FOR ADDRESS 10.10.3.245
LINK: OSA20A0LNK        ETHERNET: 00096B1A74C2
QUERYING ARP CACHE FOR ADDRESS 10.10.3.235
LINK: OSA20A0LNK        ETHERNET: 00096B1A74C2
QUERYING ARP CACHE FOR ADDRESS 10.10.3.233
LINK: OSA20A0LNK        ETHERNET: 00096B1A74C2
QUERYING ARP CACHE FOR ADDRESS 10.10.2.232
LINK: OSA20C0LNK        ETHERNET: 00096B1A7454
QUERYING ARP CACHE FOR ADDRESS 10.10.2.234
LINK: OSA20C0LNK        ETHERNET: 00096B1A7454
QUERYING ARP CACHE FOR ADDRESS 10.10.2.242
LINK: OSA20C0LNK        ETHERNET: 00096B1A7454
QUERYING ARP CACHE FOR ADDRESS 10.10.2.244
LINK: OSA20C0LNK        ETHERNET: 00096B1A7454
QUERYING ARP CACHE FOR ADDRESS 10.10.2.214
LINK: OSA20C0LNK        ETHERNET: 00096B1A7454
```

IDYNAMICXCF

You have the option of either defining the DEVICE, LINK, HOME, and START statements for MPC XCF connections to another z/OS or letting TCP/IP dynamically define them for you. Dynamic XCF devices and links, when activated, appear to the stack as though they had been defined in the TCP/IP profile. They can be displayed using standard commands, and they can be stopped and started. For multiple stack environments, IUTSAMEH links are dynamically created for same-LPAR links. Refer to *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 4 - Security*, SG24-7172, for further details.

PATHMTUDISCOVERY

Coding IPCONFIG PATHMTUDISCOVERY prevents the fragmentation of datagrams. It tells TCP/IP to discover dynamically the Path Maximum Transfer Unit (PMTU), which is the smallest of the MTU sizes of each hop in the path between two hosts.

When a connection is established, TCP/IP uses the minimum MTU of the sending host as the starting segment size and sets the Don't Fragment (DF) bit in the IP header. Any router along the route that cannot process the MTU will return an ICMP message requesting fragmentation and will inform the sending host that the destination is unreachable. The sending host can then reduce the size of its assumed PMTU. You can find more information about PMTU discovery in RFC 1191, Path MTU Discovery.

Aside from enabling PMTU during stack initialization, you could also enable or disable PMTU discovery by using VARY OBEYFILE.

MULTIPATH

With the multipath feature of CS for z/OS IP, packets can now be load balanced on routes that have been defined to be of equal cost. These routes could either be learned dynamically or defined statically in your routing program (OMPROUTE). Without multipath support, all connections use the first active route to the destination network or host even if there are other equal-cost routes available. With multipath enabled, TCP/IP will select a route to that destination network or host on a round-robin basis. TCP/IP can select a route on a per-packet basis, but this is not recommended. See Chapter 5, “Routing” on page 139 for details.

IQDIOROUTING

There is improved routing of IP traffic between HiperSockets (also known as Internal Queued Direct I/O or iQDIO) and Queued Direct I/O (QDIO). This type of routing is called *HiperSockets Accelerator* because it allows you to concentrate external network traffic over a single OSA-Express QDIO connection and then accelerates the routing over a HiperSockets link, bypassing the TCP/IP stack. To enable HiperSockets Accelerator, code the IPCONFIG IQDIOROUTING parameter. Refer to Chapter 5, “Routing” on page 139, for details.

For further information about HiperSockets, see *Communcations Server for z/OS V1R2 Implementation Guide Volume 4: Connectivity and Routing*, SG24-6516, and *z/OS V1R2.0 CS: IP Configuration Guide*, SC31-8775.

SACONFIG (SNMP subagent)

The SACONFIG statement provides subagent support for SNMP. Through the subagent support you can manage an ATM OSA network interface. Refer to the SNMP subagent chapter of *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 2 - Standard Applications*, SG24-7170, for further information.

```
SACONFig
  COMMUNity public   ; Community string
  OSASF 760          ;OSASF port number
; AGENT 161          ;Agent port number
  ENABLed
  SETSENAbled
  ATMENabled
```

For more information about SNMP and subagent support, see *IBM Communications Server for OS/390 V2R10 TCP/IP Implementation Guide Volume 2: UNIX Applications*, SG24-5228.

SMFCONFIG

TCP/IP SMF records written using record type 118 do not contain the identity of the TCP/IP stack. If you run multiple TCP/IP stacks, it is not easy to determine which SMF records relate to which TCP/IP stack. A new SMF record type 119 provides that option. Type 119 records contain additional values that identify the TCP/IP stack. Each type 119 record consists of an SMF header, a TCP/IP identification section, and a data section. The following type 119 records are available:

- TCP connection initiation and termination
- UDP socket close
- TCP/IP, interface, and server port statistics
- TCP/IP stack start/stop
- FTP server transfer completion
- FTP server logon failure
- FTP client transfer completion
- TN3270 server session initiation and termination
- Telnet client connection initiation and termination

The SMFCONFIG statement is used to turn on SMF logging. It defines the type 118 and type 119 records to be collected (the default format is type 118).

The SMFPARMS statement can also be used to turn on SMF logging. However, you are encouraged to migrate to SMFCONFIG, which has the following advantages over the SMFPARMS statement:

- Using SMFCONFIG means that SMF records are written using standard subtypes. With SMFPARMS, you have to specify the subtypes to be used.
- SMFCONFIG allows you to record both type 118 and type 119 records. With SMFPARMS, only type 118 records can be collected.
- SMFCONFIG enables you to record a wider variety of information.
- By using SMFCONFIG, you gain support for dynamic reconfiguration, for all environments under which CS for z/OS IP is executing (SRB mode, reentrant, XMEM mode, etc.), and you can avoid duplicate SMF exit processes.

In the following example, type 118 FTP client records and type 119 TN3270 client records are collected:

```
SMFCONFIG TYPE118 FTPCLIENT
          TYPE119 TN3270CLIENT
```

The above example can also be coded this as below, since type 118 records are collected by default:

```
SMFCONFIG FTPCLIENT
          TYPE119 TN3270CLIENT
```

SMFCONFIG is coded in the PROFILE.TCPIP, but it has related entries in both Telnet and in FTP. (See *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 2 - Standard Applications*, SG24-7170.)

The only SMF exit supported in CS for z/OS IP is the FTP server SMF exit, FTPSMFEX. This exit is only called for type 118 records. If you need to access type 119 FTP SMF records, use the standard SMF exit facilities, IEFU83, IEFU84, and IEFU85.

For further information about TCP/IP SMF records, see *z/OS V1R2.0 CS: IP Configuration Guide*, SC31-8775. *z/OS V1R7.0 CS: IP Configuration Reference,* SC31-8776, contains the SMF record layouts and the standardized subtype numbers used by the Communications Server for z/OS IP.

ASSORTEDPARMS

If you use the ASSORTEDPARMS statement, consider migrating parameters from the ASSORTEDPARMS. All parameters on the ASSORTEDPARMS statement should be defined using the GLOBALCONFIG, IPCONFIG, TCPCONFIG, and UDPCONFIG statements. Support for the ASSORTEDPARMS statement will be removed in a future release of CS for z/OS IP. If the ASSORTEDPARMS statement is specified, the following warning message is issued:

```
EZZ0717I ASSORTEDPARMS STATEMENT ON LINE lineno WILL BE RETIRED IN A FUTURE RELEASE
```

ASSORTEDPARMS and the GLOBALCONFIG, IPCONFIG, TCPCONFIG, and UDPCONFIG statements are mutually exclusive and should not be used together.

TCPCONFIG FINWAIT2TIME

The TCPCONFIG FINWAIT2TIME parameter allows you to specify the number of seconds a TCP connection should remain in the FINWAIT2 state. When this time limit is reached, the

system waits a further 75 seconds before dropping the connection. The default is 600 seconds, but you can specify a value as low as 60 seconds, which will reduce the time a connection remains in the FINWAIT2 status, and thereby free up resources for future connections.

TCPCONFIG TCPTIMESTAMP

The TCP time stamp option is exchanged during connection setup. This option is enabled (by default) via the TCPCONFIG TCPTIMESTAMP parameter. Enabling TCP time stamp allows TCP/IP to better estimate the Route Trip Response Time (RTT), which helps avoid unnecessary retransmissions and helps protect against the wrapping of sequence numbers.

DEVICE AND LINK

Device and Link statements now support new features to support VLAN IDs and OFFLOAD processing to the OSA-Express adapter.

OFFLOAD

When sending or receiving packets over OSA-Express in QDIO mode with checksum offload support, TCP/IP offloads most IPv4 (outbound and inbound) checksum processing (IP header, TCP, and UDP checksums) to the OSA. The TCP/IP stack still performs checksum processing in the cases where checksum cannot be offloaded. Refer to the *z/OS Communications Server: IP Configuration Guide, Version 1 Release 7*, SC31-8775-07; and *z/OS Communications Server: IP Configuration Reference, Version 1 Release 7*, SC31-8776-08, for details.

When sending packets over OSA-Express in QDIO mode with TCP segmentation offload support, TCP/IP offloads most IPv4 outbound TCP segmentation processing to the OSA. The TCP/IP stack still performs TCP segmentation processing in the cases where segmentation cannot be offloaded.

Tip: Applications that use large TCP send buffers will obtain the most benefit from TCP segmentation offload. The size of the TCP receive buffer on the other side of the TCP connection also affects the negotiated buffer size. You can control the size of these buffers using the TCPSENDBFRSIZE and TCPRCVBUFRSIZE parameters on the TCPCONFIG statement to set the default TCP send/receive buffer size for all applications. However, an application can use the SO_SNDBUF socket option to override the default TCP send buffer sizes (example FTP).

The segmentation offload feature decreases host CPU utilization and increases data transfer efficiency for IPv4 packets. The Communications Server for z/OS V1R7 provides the Offloads feature for IPv4 segmentation processing to OSA-Express2 in QDIO mode. This enhances the data transfer efficiency of IPv4 packets while decreasing host CPU utilization

The OFFLOAD feature is currently supported by OSA-Express in QDIO mode. Example 6-14 displays the features of OSA-Express that are enabled.

Example 6-14 OFFLOAD enabled

```
MVS TCP/IP NETSTAT CS V1R7       TCPIP Name: TCPIPE          00:48:39
DevName: LOOPBACK          DevType: LOOPBACK
  DevStatus: Ready
  LnkName: LOOPBACK          LnkType: LOOPBACK   LnkStatus: Ready 1
NetNum: n/a  QueSize: n/a
    ActMtu: 65535
  BSD Routing Parameters:
    MTU Size: n/a               Metric: 00
```

```
    DestAddr: 0.0.0.0           SubnetMask: 0.0.0.0
  Multicast Specific:
    Multicast Capability: No
  Link Statistics:
    BytesIn                          = 602814
    Inbound Packets                  = 9190
    Inbound Packets In Error         = 0
    Inbound Packets Discarded        = 0
    Inbound Packets With No Protocol = 0
    BytesOut                         = 602814
    Outbound Packets                 = 9190
    Outbound Packets In Error        = 0
    Outbound Packets Discarded       = 0
IntfName: LOOPBACK6        IntfType: LOOPBACK6 IntfStatus: Ready 2
NetNum: n/a  QueSize: n/a
    ActMtu: 65535
  Multicast Specific:
    Multicast Capability: No
  Interface Statistics:
    BytesIn                          = 0
    Inbound Packets                  = 0
    Inbound Packets In Error         = 0
    Inbound Packets Discarded        = 0
    Inbound Packets With No Protocol = 0
    BytesOut                         = 0
    Outbound Packets                 = 0
    Outbound Packets In Error        = 0
    Outbound Packets Discarded       = 0
DevName: OSA2080           DevType: MPCIPA
  DevStatus: Ready   3
  LnkName: OSA2080LNK        LnkType: IPAQENET   LnkStatus: Ready
    NetNum: n/a  QueSize: n/a  Speed: 0000000100
    IpBroadcastCapability: No
    CfgRouter: Non                 ActRouter: Non
    ArpOffload: Yes     4           ArpOffloadInfo: Yes 4
ActMtu: 1492
    VLANid: 10       5              VLANpriority: Disabled
    DynVLANRegCfg: No              DynVLANRegCap: Yes
    ReadStorage: GLOBAL (4096K)    InbPerf: Balanced
    ChecksumOffload: Yes   4         SegmentationOffload: Yes 4
SecClass: 255
  BSD Routing Parameters:
    MTU Size: 1500             Metric: 10
    DestAddr: 0.0.0.0          SubnetMask: 255.255.255.0
  Multicast Specific:
    Multicast Capability: Yes
    Group           RefCnt
    -----           ------
    224.0.0.5       0000000001
    224.0.0.1       0000000001
  Link Statistics:
    BytesIn                          = 7312941
    Inbound Packets                  = 57814
    Inbound Packets In Error         = 24429
    Inbound Packets Discarded        = 0
    Inbound Packets With No Protocol = 0
    BytesOut                         = 1050260
    Outbound Packets                 = 9237
    Outbound Packets In Error        = 6
```

```
    Outbound Packets Discarded        = 0
```

4 indicates the enabled features of ARP, Segmentation, and Checksum Offload.

5 indicates the VLAN ID value.

Support is provided for virtual local area network standard IEEE 802.1q (VLAN). Implementing VLAN allows a physical LAN to be logically subdivided into separate logical LANs. It is configured and implemented in the z/OS environment through the LINK definitions in the PROFILE data set for OSA-Express in QDIO mode. VLANs support takeover capabilities (ARP takeover) in a *flat network* (no routing protocol) when connected appropriately and is explained in its implementation in Chapter 4, "Connectivity" on page 101. Here in this a link definition example of OSA2080LNK attached to virtual LAN 10:

```
DEVICE OSA2080  MPCIPA
 LINK   OSA2080LNK  IPAQENET      OSA2080 VLANID 10
```

TCP/IP built-in security functions

TCP/IP for MVS has some built-in security functions that may be activated and used to control specific areas:

- The Simple Mail Transfer Protocol (SMTP) provides a secure mail gateway option that allows an installation to create a database of registered network job entry (NJE) users who are allowed to send mail through SMTP to a TCP/IP network recipient.
- The FTP server gives you the opportunity to code security exits, in which you may extend the control over the functions performed by the FTP server. Using these exits you may control:
  - The use of the FTP server based on IP addresses and port numbers
  - The use of the FTP server based on user IDs
  - The use of individual FTP subcommands
  - The submission of batch jobs via the FTP server
- SNMP with Communications Server for z/OS IP has an SNMP agent that supports community-based security such as SNMPv1 and SNMPv2C, and user-based security such as SNMPv3. If you are concerned about sending SNMP data in a less secure environment, you may consider implementing SNMPv3, whose messages have data integrity and data origin authentication.

  RACF considerations for SNMP in CS for z/OS IP are covered in detail in *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 4 - Security*, SG24-7172.

Both the IMS sockets and CICS sockets support provides a user exit that you can use to validate each IMS or CICS transaction received by the Listener function. How you code this exit and what data you require to be present in the transaction initiation request is up to you to decide.

C

Examples used in our environment

This appendix provides the examples used in the configuration of our environment.

It provides the following:

- Resolver examples
- Routing examples

Resolver

The following are the complete examples used in our environment and discussed in Chapter 2, “The Resolver” on page 17.

Example: C-1 TCPIPA.TCPPARMS (IPNODES)

```
;  Entries in the hosts file have the following format:
;
;  Address HostName
;
;  Address HostName1 HostName2 HostName3 ..... HostName35
;
;  Address: is an IP address, it can be IPV4 or IPV6 address.
;           Note: IPv4-mapped IPv6 address is not allowed.
;  HostName: the length of the hostname is up to 128 characters,
;            and each IP address can have up to 35 hostnames.
;
   10.12.6.7    OURDNS
   10.10.1.230 WTSC30A tcpipa30
   10.10.1.221 WTSC32A tcpipa32
   10.10.1.241 WTSC31A tcpipa31
   10.10.2.1   router1
   10.10.3.2   router2
   1::2 TESTIPV6ADDRESS1
   1:2:3:4:5:6:7:8 TESTIPV6ADDRESS2
```

Example: C-2 TCPIPA.TCPPARMS (GLOBAL)

```
;DOMAINORIGIN  ITSO.IBM.COM
SEARCH  ITSO.IBM.COM IBM.COM
DATASETPREFIX TCPIP
MESSAGECASE MIXED
;SINTERADDR  10.12.6.7
NSPORTADDR 53
RESOLVEVIA UDP
RESOLVERTIMEOUT 10
RESOLVERUDPRETRIES 1
LOOKUP LOCAL
```

Example: C-3 TCPIPA.TCPPARMS (DEFAULT)

```
TCPIPJOBNAME TCPIP
HOSTNAME WTSC30
```

Example: C-4 SYS1.PROCLIB (RESOLV30)

```
//RESOLV32 PROC PARMS='CTRACE(CTIRES00)'
//*
//*
//*   Function: Start Resolver
//*
//EZBREINI EXEC PGM=EZBREINI,REGION=0M,TIME=1440,PARM=&PARMS
//*
```

```
//*   When the Resolver is started by UNIX System Services it is
//*     started with SUB=MSTR.
//*   This means that JES services are not available to the Resolver
//*     address space. Therefore, no DD cards with SYSOUT can be used.
//*     See the MVS JCL Reference manual for SUB=MSTR considerations in
//*     section "Running a Started Task Under the Master Subsystem".
//*   This Resolver start procedure will need to reside in a data
//*     set that is specified by the MSTJCLxx PARMLIB member's
//*     IEFPDSI DD card specification. If not, the procedure will
//*     not be found and the Resolver will not start.
//*     See the MVS Initialization and Tuning Reference manual for
//*     MSTJCL considerations in section "Understanding the Master
//*     Scheduler Job Control Language"
//*
//*   SETUP contains Resolver setup parameters.
//*     See the section on "Understanding Resolvers" in the
//*     IP Configuration Guide for more information. A sample of
//*     Resolver setup parameters is included in member RESSETUP
//*     of the SEZAINST data set.
//*
//SETUP   DD   DSN=TCPIPA.TCPPARMS(NEWSETUP),DISP=SHR,FREE=CLOSE
//*SETUP   DD   DSN=TCPIP.SETUP.RESOLVER,DISP=SHR,FREE=CLOSE
//*SETUP   DD   PATH='/etc/setup.resolver',PATHOPTS=(ORDONLY)
```

Example: C-5 TCPIPA.TCPPARMS (NEWSETUP)

```
;EFAULTTCPIPDATA('TCPIPA.TCPPARMS(DEFAULT)')
;
GLOBALTCPIPDATA('TCPIPA.TCPPARMS(GLOBAL)')
;
;GLOBALIPNODES('TCPIPA.TCPPARMS(IPNODES)')
;
;
DEFAULTIPNODES('TCPIPD.TCPPARMS(IPNODES)')
;
; NOCOMMONSEARCH
;
COMMONSEARCH
```

Routing

These are the complete examples used in our enviornment and discussed in Chapter 5, “Routing” on page 139. They include the TCPIPA stacks on SC30, SC31, and SC32.

Example: C-6 TCPIPA.OMPROUTE.STDENV30

```
RESOLVER_CONFIG=//'TCPIPA.TCPPARMS(DATAa30)'
OMPROUTE_FILE=//'TCPIPA.TCPPARMS(OMPA30S)'
OMPROUTE_DEBUG_FILE=/tmp/syslog/debuga30
OMPROUTE_DEBUG_FILE_CONTROL=10000,5
```

Example: C-7 TCPIPA.OMPROUTE.STDENV31

```
RESOLVER_CONFIG=//'TCPIPA.TCPPARMS(DATAA31)'
```

```
OMPROUTE_FILE=//'TCPIPA.TCPPARMS(OMPA31S)'
OMPROUTE_DEBUG_FILE=/tmp/syslog/debug31
OMPROUTE_DEBUG_FILE_CONTROL=10000,5
```

Example: C-8 TCPIPA.OMPROUTE.STDENV32

```
RESOLVER_CONFIG=//'TCPIPA.TCPPARMS(DATAA32)'
OMPROUTE_FILE=//'TCPIPA.TCPPARMS(OMPA32S)'
OMPROUTE_DEBUG_FILE=/tmp/syslog/debug32
OMPROUTE_DEBUG_FILE_CONTROL=10000,5
```

Example: C-9 TCPIPA.TCPARMS (DATAA30)

```
TCPIPJOBNAME TCPIPA
HOSTNAME SC30A
DOMAINORIGIN  ITSO.IBM.COM
DATASETPREFIX TCPIPA
MESSAGECASE MIXED
;NSINTERADDR  10.12.6.7
NSPORTADDR 53
RESOLVEVIA UDP
RESOLVERTIMEOUT 10
RESOLVERUDPRETRIES 1
```

Example: C-10 TCPIPA.DATAA31

```
TCPIPJOBNAME TCPIPA
HOSTNAME WTSC31A
DOMAINORIGIN  ITSO.IBM.COM
DATASETPREFIX TCPIPA
MESSAGECASE MIXED
; NSINTERADDR  10.12.6.7
NSPORTADDR 53
RESOLVEVIA UDP
RESOLVERTIMEOUT 10
RESOLVERUDPRETRIES 1
LOOKUP LOCAL
```

Example: C-11 TCPIPA.DATAA32

```
; OPTIONS DEBUG
TCPIPJOBNAME TCPIPA
HOSTNAME WTSC32A
DOMAINORIGIN  ITSO.IBM.COM
DATASETPREFIX TCPIPA
MESSAGECASE MIXED
NSINTERADDR  10.12.6.7
NSPORTADDR 53
RESOLVEVIA UDP
RESOLVERTIMEOUT 10
RESOLVERUDPRETRIES 1
```

Example: C-12 TCPIP.TCPPARMS (OPMA30S)

```
Area Area_Number=0.0.0.2
     Stub_Area=YES
     Authentication_type=None
     Import_Summaries=No;
;
OSPF RouterID=10.10.1.230;
;
Routesa_Config Enabled=No;
;
; Static vipa
OSPF_Interface IP_address=10.10.1.230
          Subnet_mask=255.255.255.252
          Name=STAVIPA1LNK
          Attaches_To_Area=0.0.0.2
          Advertise_VIPA_Routes=HOST_ONLY
          Cost0=10
          MTU=65535;
;
; OSA Qdio VLAN10
OSPF_Interface IP_address=10.10.2.*
           Subnet_mask=255.255.255.0
           Router_Priority=0
           Attaches_To_Area=0.0.0.2
           Cost0=10
           MTU=1492;
;
; OSA Qdio VLAN11
OSPF_Interface IP_address=10.10.3.*
           Subnet_mask=255.255.255.0
           Router_Priority=0
           Attaches_To_Area=0.0.0.2
           Cost0=10
           MTU=1492;
;
; hipersockets
OSPF_Interface IP_address=10.10.4.*
           Subnet_mask=255.255.255.0
           Attaches_To_Area=0.0.0.2
           Cost0=100
           MTU=8192;
; hipersockets
OSPF_Interface IP_address=10.10.5.*
           Subnet_mask=255.255.255.0
           Attaches_To_Area=0.0.0.2
           Cost0=100
           MTU=8192;
;
; xcf
OSPF_Interface IP_address=10.20.*.*
           Subnet_mask=255.255.255.0
           Attaches_To_Area=0.0.0.2
           Cost0=5
           MTU=8192;
;
```

```
AS_Boundary_routing
  Import_Static_Routes=yes
  Import_Direct_Routes=yes
  Learn_Default_Route=yes;
```

Example: C-13 TCPIPA.TCPPARMS (OMPA31S)

```
Area Area_Number=0.0.0.2
     Stub_Area=YES
     Authentication_type=None
     Import_Summaries=No;
;
OSPF RouterID=10.10.1.241;
;
Routesa_Config Enabled=No;
;
; Static vipa
OSPF_Interface IP_address=10.10.1.241
          Subnet_mask=255.255.255.252
          Name=STAVIPA1LNK
          Attaches_To_Area=0.0.0.2
          Advertise_VIPA_Routes=HOST_ONLY
          Cost0=10
          MTU=65535;
;
; OSA Qdio VLAN10
OSPF_Interface IP_address=10.10.2.*
           Subnet_mask=255.255.255.0
           Router_Priority=0
           Attaches_To_Area=0.0.0.2
           Cost0=10
           MTU=1492;
;
; OSA Qdio VLAN11
OSPF_Interface IP_address=10.10.3.*
           Subnet_mask=255.255.255.0
           Router_Priority=0
           Attaches_To_Area=0.0.0.2
           Cost0=10
           MTU=1492;
;
; hipersockets
OSPF_Interface IP_address=10.10.4.*
           Subnet_mask=255.255.255.0
           Attaches_To_Area=0.0.0.2
           Cost0=100
           MTU=8192;
; hipersockets
OSPF_Interface IP_address=10.10.5.*
           Subnet_mask=255.255.255.0
           Attaches_To_Area=0.0.0.2
           Cost0=100
           MTU=8192;
;
; xcf
OSPF_Interface IP_address=10.20.*.*
           Subnet_mask=255.255.255.0
           Attaches_To_Area=0.0.0.2
```

```
          Cost0=5
          MTU=8192;
;
AS_Boundary_routing
  Import_Direct_Routes=yes
  Import_Static_Routes=yes
  Learn_Default_Route=yes;
```

Example: C-14 TCPIPA.TCPPARMS (OMPA32S)

```
Area Area_Number=0.0.0.2
     Stub_Area=YES
     Authentication_type=None
     Import_Summaries=No;
;
OSPF RouterID=10.10.1.221;
;
Routesa_Config Enabled=No;
;
; Static vipa
OSPF_Interface IP_address=10.10.1.221
          Subnet_mask=255.255.255.252
          Name=STAVIPA1LNK
          Attaches_To_Area=0.0.0.2
          Advertise_VIPA_Routes=HOST_ONLY
          Cost0=10
          MTU=65535;
;
; OSA Qdio VLAN10
OSPF_Interface IP_address=10.10.2.*
           Subnet_mask=255.255.255.0
           Router_Priority=0
           Attaches_To_Area=0.0.0.2
           Cost0=10
           MTU=1492;
;
; OSA Qdio VLAN11
OSPF_Interface IP_address=10.10.3.*
           Subnet_mask=255.255.255.0
           Router_Priority=0
           Attaches_To_Area=0.0.0.2
           Cost0=10
           MTU=1492;
;
; hipersocket
OSPF_Interface IP_address=10.10.4.*
           Subnet_mask=255.255.255.0
           Attaches_To_Area=0.0.0.2
           Cost0=100
           MTU=8192;
; hipersocket
OSPF_Interface IP_address=10.10.5.*
           Subnet_mask=255.255.255.0
           Attaches_To_Area=0.0.0.2
           Cost0=100
           MTU=8192;
;
; xcf
```

```
OSPF_Interface IP_address=10.20.*.*
           Subnet_mask=255.255.255.0
           Attaches_To_Area=0.0.0.2
           Cost0=5
           MTU=8192;
;
AS_Boundary_routing
   Import_Direct_Routes=yes
   Import_Static_Routes=yes
   Learn_Default_Route=yes;
```

Example: C-15 SYS1.PROCLIB (OMPA)

```
//*
//OMP3OA PROC STDENV=STDENV&SYSCLONE
//OMP3OA EXEC PGM=OMPROUTE,REGION=4096K,TIME=NOLIMIT,
//        PARM=('POSIX(ON) ALL31(ON)',
//            'ENVAR("_BPXK_SETIBMOPT_TRANSPORT=TCPIPA"',
//            '"_CEE_ENVFILE=DD:STDENV")/-t2')
//*
//STDENV   DD DISP=SHR,DSN=TCPIPA.OMPROUTE.&STDENV
//*
//*        The stdout stream may be redirected to a z/OS UNIX filesystem file as
//*        shown below.
//*        The PATHOPTS OTRUNC option will clear the stdout file
//*        every time OMPROUTE is started.  If you want to retain
//*        previous stdout information, change it to OAPPEND.
//*
//*YSPRINT DD SYSOUT=*
//*YSPRINT DD PATH='/tmp/omproute/omproute.sysprint',
//*            PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//*            PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIWGRP)
//*
//*        The stderr stream may be redirected to a z/OS UNIX filesystem file as
//*        shown below.
//*        The PATHOPTS OTRUNC option will clear the stderr file
//*        every time OMPROUTE is started.  If you want to retain
//*        previous stderr information, change it to OAPPEND.
//*
//SYSOUT   DD SYSOUT=*
//*YSOUT   DD PATH='/tmp/omproute/omproute.stderr',
//*           PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//*           PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIWGRP)
//*TDOUT   DD PATH='/tmp/omproute/omproute.stdout',
//*            PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//*            PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIWGRP)
//*TDERR   DD PATH='/tmp/omproute/omproute.stderr',
//*            PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//*            PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIWGRP)
//*
//CEEDUMP  DD SYSOUT=*,DCB=(RECFM=FB,LRECL=132,BLKSIZE=132)
```

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 251. Note that some of the documents referenced here may be available in softcopy only.

- *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 2 - Standard Applications*, SG24-7170
- *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 3 - High Availability, Scalability, and Performance*, SG24-7171
- *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 4 - Security,* SG24-7172
- *z/OS Communications Server: SNA Network Implementation Guide*, *Migrating Subarea Networks to an IP Infrastructure Using Enterprise Extender,* SG24-5957
- *TCP/IP Tutorial and Technical Overview*, GG24-3376
- *The Basics of IP Network Design*, SG24-2580
- *OSA-Express Implementation Guide*, SG24-5948
- *zSeries HiperSockets*, SG24-6816
- *SNA in a Parallel Sysplex Environment,* SG24-2113
- *z/OS Infoprint Server Implementation*, SG24- 6234
- *z/OS Security Services Update,* SG24-6448-00
- *Deploying a Public Key Infrastructure*, SG24-5512

Other publications

These publications are also relevant as further information sources:

- *z/OS V1R7.0 XL C/C++ Run-Time Library Reference*, SA22-7821
- *z/OS V1R7.0 Communications Server: IP System Administrator’s Commands*, SC31-8781
- *z/OS V1R7.0 MVS IPCS Commands*, SA22-7594
- *z/OS V1R7.0 MVS System Commands*, SA22-7627
- *z/OS V1R7.0 Communications Server: SNA Operation*, SC31-8779
- *z/OS V1R7.0 TSO/E Command Reference*, SA22-7782
- *z/OS V1R7.0 UNIX System Services Command Reference*, SA22-7802
- *z/OS V1R2.0 Communications Server: CSM Guide*, SC31-8808
- *z/OS V1R7.0 Communications Server: New Function Summary*, GC31-8771
- *z/OS V1R7.0 Communications Server: Quick Reference*, SX75-0124

- *z/OS V1R7.0 Communications Server: IP and SNA Codes*, SC31-8791
- *z/OS V1R7.0 Communications Server: IP System Administrator's Commands*, SC31-8781
- *z/OS V1R7.0 MVS IPCS Commands*, SA22-7594
- *z/OS V1R7.0 Communications Server: IP Diagnosis Guide*, GC31-8782
- *z/OS V1R7.0 Communications Server: IP Configuration Guide*, SC31-8775
- *z/OS V1R7.0 Communications Server: IP Messages Volume 1 (EZA)*, SC31-8783
- *z/OS V1R7.0 Communications Server: IP Messages Volume 2 (EZB, EZD)*, SC31-8784
- *z/OS V1R7.0 Communications Server: IP Messages Volume 3 (EZY)*, SC31-8785
- *z/OS V1R7.0 Communications Server: IP Messages Volume 4 (EZZ, SNM)*, SC31-8786
- *z/OS V1R7.0 Communications Server: IP Programmer's Guide and Reference*, SC31-8787
- *z/OS V1R7.0 Communications Server: IP Configuration Reference*, SC31-8776
- *z/OS V1R7.0 Communications Server: IP Sockets Application Programming Interface Guide and Reference*, SC31-8788
- *z/OS V1R7.0 Communications Server: IP User's Guide and Commands*, SC31-8780
- *z/OS V1R7.0 Communications Server: IP User's Guide and Commands*, SC31-8780
- *z/OS V1R7.0 Communications Server: IPv6 Network and Application Design Guide*, SC31-8885
- *z/OS V1R7.0 Migration*, GA22-7499
- *z/OS V1R7.0 MVS System Commands*, SA22-7627
- *OSA-Express Customer's Guide and Reference*, SA22-7935
- *z/OS V1R7.0 Communications Server: SNA Operation*, SC31-8779
- *z/OS V1R7.0 TSO/E Command Reference*, SA22-7782
- *z/OS V1R7.0 UNIX System Services Programming: Assembler Callable Services Reference*, SA22-7803
- *z/OS V1R7.0 UNIX System Services Command Reference*, SA22-7802
- *z/OS V1R7.0 UNIX System Services File System Interface Reference*, SA22-7808
- *z/OS V1R7.0 UNIX System Services Messages and Codes*, SA22-7807
- *z/OS V1R7.0 UNIX System Services Parallel Environment: Operation and Use*, SA22-7810
- *z/OS V1R7.0 UNIX System Services Programming Tools*, SA22-7805
- *z/OS V1R7.0 UNIX System Services Planning*, GA22-7800
- *z/OS V1R7.0 UNIX System Services User's Guide*, SA22-7801
- *z/OS V1R7.0 UNIX System Services User's Guide*, SA22-7801

Online resources

These Web sites and URLs are also relevant as further information sources:

- Mainframe networking

  http://www.ibm.com/servers/eserver/zseries/networking/

- z/OS Communications Server product overview

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

- z/OS Communications Server publications

  http://www-03.ibm.com/servers/eserver/zseries/zos/bkserv/r7pdf/commserv.html

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Z

Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 1 - Base Functions, Connectivity, and Routing

Redbooks

Understand important CS for z/OS TCP/IP base function capabilities

See CS for z/OS base function implementation examples

Learn useful verification techniques

This new and improved Communications Server (CS) for z/OS TCP/IP Implementation series provides easy-to-understand step-by-step how-to guidance on enabling the most commonly used and important functions of CS for z/OS TCP/IP.

In this IBM Redbook we begin by providing an introduction to CS for z/OS TCP/IP. We then discuss the System Resolver showing the implementation of global and local settings for single and multi-stack environments. We then present implementation scenarios for TCP/IP base functions, connectivity, and routing. Finally, we discuss the IP version 6 support available with z/OS V1R7.0 Communications Server.

For more specific information about CS for z/OS standard applications, high availability, and security, please reference the other volumes in the series. These are:

- *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 2 - Standard Applications*, SG24-7170
- *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 3 - High Availability, Scalability, and Performance*, SG24-7171
- *Communications Server for z/OS V1R7 TCP/IP Implementation, Volume 4 - Security*, SG24-7172

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7169-00 ISBN 0738496197