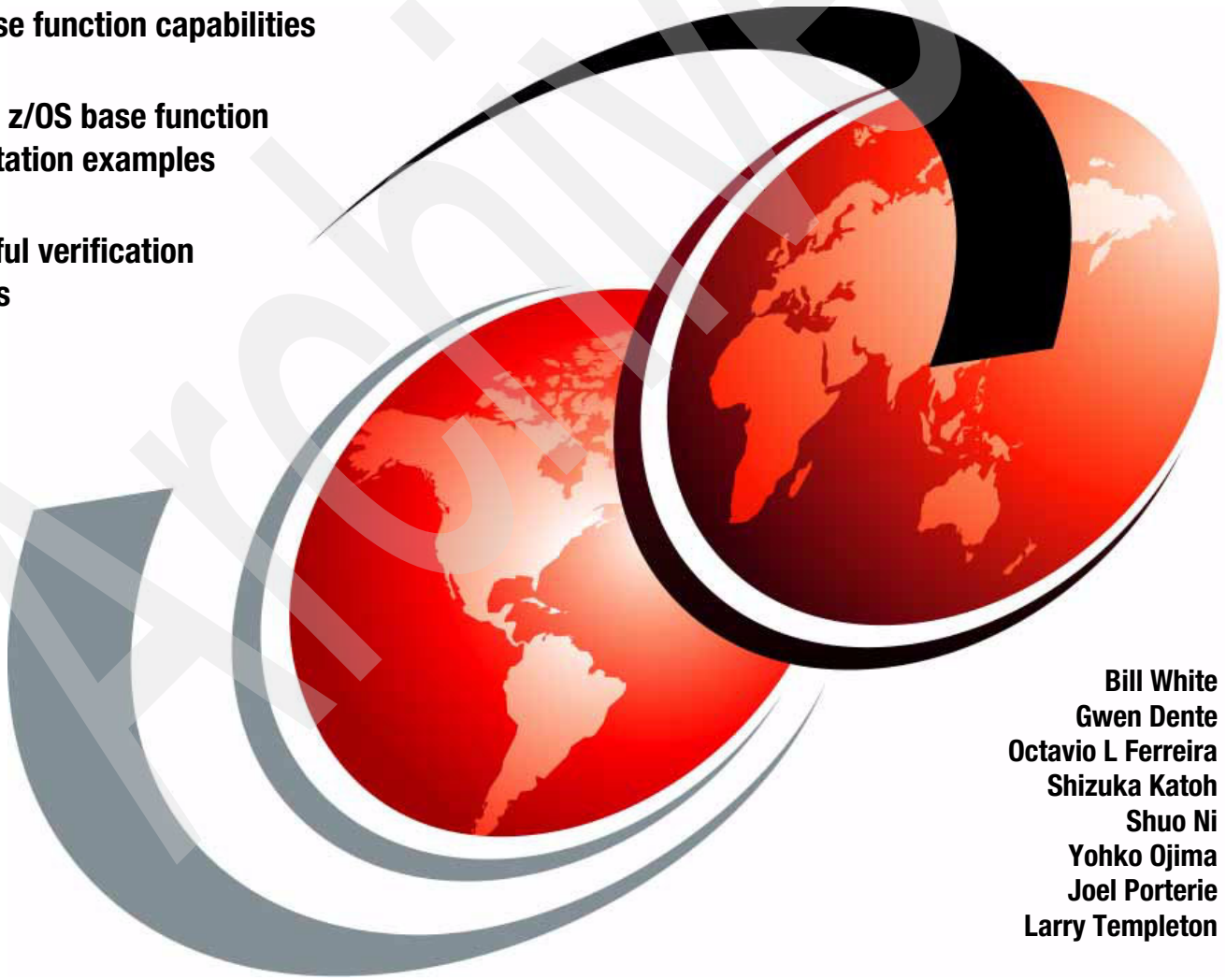
IBM

Communications Server for z/OS V1R9 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



Bill White
Gwen Dente
Octavio L Ferreira
Shizuka Katoh
Shuo Ni
Yohko Ojima
Joel Porterie
Larry Templeton

Redbooks

ibm.com/redbooks

International Technical Support Organization

CS for z/OS V1R9 TCP/IP Implementation Volume 1: Base Functions, Connectivity, and Routing

March 2008

SG24-7532-00

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

First Edition (March 2008)

This edition applies to Version 1, Release 9 of Communications Server for z/OS.

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 illustrate 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.

Trademarks

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

Redbooks (logo) ®
z/OS®
z/VM®
zSeries®
z9™
AIX®
CICS®
DFSMS™
DFSMS/MVS™
ESCON®
FICON®
HiperSockets™
IBM®
IMS™
Language Environment®
Lotus®
MVS™
NetView®
OMEGAMON®
Parallel Sysplex®
Redbooks®
RACF®
REXX™
System p™
System z™
System z9®
System/360™
Tivoli®
VTAM®
WebSphere®

The following terms are trademarks of other companies:

SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

Catalyst, the AMD Arrow logo, and combinations thereof, are trademarks of Advanced Micro Devices, Inc.

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

Convergence, 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

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. The *Communications Server for z/OS® TCP/IP Implementation* series provides easy-to-understand, step-by-step guidance about enabling the most commonly used and important functions of CS for z/OS TCP/IP.

In this IBM Redbooks® publication, *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 1: Base Functions, Connectivity, and Routing*, SG24-7532, we provide 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. Next, we present implementation scenarios for TCP/IP Base functions, Connectivity, Routing, Virtual MAC support, and sysplex subplexing.

For more specific information about CS for z/OS standard applications, high availability, and security, refer to the other volumes in the series:

- *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 2: Standard Applications*, SG24-7533
- *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 3: High Availability, Scalability, and Performance*, SG24-7534
- *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 4: Security and Policy-Based Networking*, SG24-7535

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 z™, 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. It provides, 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.

For comprehensive descriptions of the individual parameters for setting up and using the functions described in this book, along with step-by-step checklists and supporting examples, refer to the following publications:

- *z/OS Communications Server: IP Configuration Guide*, SC31-8775
- *z/OS Communications Server: IP Configuration Reference*, SC31-8776
- *z/OS Communications Server: IP User's Guide and Commands*, SC31-8780

This book does not duplicate the information in those publications. Instead, it complements them with practical implementation scenarios that can be useful in your environment. To determine at what level a specific function was introduced, refer to *z/OS Communications Server: New Function Summary*, GC31-8771. For complete details, we encourage you to review the documents referred to in the additional resources section at the end of each chapter.

The team that wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Bill White is a Project Leader and Senior Networking Specialist at the International Technical Support Organization, Poughkeepsie Center.

Gwen Dente has spent 27 years with IBM, focusing on mainframe networking and security support for customers and the IBM technical community. She is a Consulting IT Specialist with IBM Advanced Technical Support at the Washington Systems Center in Gaithersburg, Maryland, USA. Gwen presents frequently at SHARE and other IBM technical conferences, and has shared authorship of multiple IBM Redbooks publications.

Octavio L Ferreira is a Senior IT Specialist in IBM Brazil. He has 28 years of experience in IBM software support. His areas of expertise include z/OS Communications Server, SNA and TCP/IP, and Communications Server on all platforms. For the last 10 years, Octavio 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. He has also co-authored other IBM Redbooks publications.

Shizuka Katoh is an Advisory IT Specialist working for Advanced Technical Support in IBM Japan. She has more than 10 years of experience the field of Mainframe Networking. Her areas of expertise include z/OS Communications Server, TCP/IP (IPv4 and IPv6), VTAM®, and OSA. Shizuka specializes in High Availability Data Center networking solutions with Parallel Sysplex® environments.

Shuo Ni is an Advisory IT Specialist in IBM Global Services, China. Her areas of expertise include z/OS, TCP/IP, VTAM (Subarea and APPN), OSA-Express, HiperSockets™, WLM, and CICS®. For the last six years, Shuo has provided onsite service support to mainframe clients in southern China for z/OS Communication Server (including Enterprise Extender and CICS TCP/IP Sockets), as well as other mainframe subsystems.

Yohko Ojima is an IT Specialist in IBM Japan and has five years of experiences in the enterprise networking area. She specializes in TCP/IP and SNA networks with Communications Server products, routers, and switches. Yohko's responsibilities include providing technical support and reviewing the networking design for many customers, including Communications Server for z/OS (TCP/IP and VTAM), Communications Server for distributed servers, OSA, and Communication Controller for Linux® on System z.

Joel Porterie is a Senior IT Specialist who has been with IBM France for 30 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. Joel has taught OSA-Express and FICON® problem determination classes, and has provided onsite assistance in these areas in numerous countries. He has also co-authored many other IBM Redbooks publications.

Larry Templeton is a Network Architect with IBM Global Services, Network Outsourcing. He has 38 years of experience with IBM Mainframe and Networking systems, consulting with clients around the world. His current responsibilities include architecting secure mainframe IP connectivity solutions, designing inter-company Enterprise Extender configurations, and assisting clients with high-availability data center implementations. Larry has also co-authored other IBM Redbooks publications.

Thanks to the following people from the International Technical Support Organization, Poughkeepsie Center for their contributions to this project: David Bennin, Ella Buslovich, Rich Conway, and Bob Haimowitz.

As is always the case with any complex technical effort, 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 this book from the Communications Server for z/OS development team, especially Doris Bunn, Alfred Christensen, Jeff Haggar, Jean Kristufek, Alan Packett, Marc Price, and Ray Ward.

Finally, thanks to the authors of the previous *Communications Server (CS) for z/OS TCP/IP Implementation* series for creating the groundwork for this series: Rama Ayyar, Valirio Braga, Gilson Cesar de Oliveira, Octavio Ferreira, Adi Horowitz, Michael Jensen, Sherwin Lake, Bob Louden, Garth Madella, Yukihiko Miyamoto, Roland Peschke, Joel Porterie, Marc Price, Larry Templeton, Rudi van Niekerk, and Thomas Wienert.

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 have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer 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 e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

1

Introduction to z/OS Communications Server for IP

z/OS Communications Server (CS for z/OS) 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, and it provides a multitude of technologies. Collectively, those technologies 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 z/OS Communications Server 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 can be obtained from the publications listed in 1.4, “For additional information” on page 17.

This chapter discusses the following topics.

Section	Topic
1.1, “Overview” on page 2	Basic concepts of CS for z/OS IP
1.2, “Featured functions” on page 3	Key characteristics of CS for z/OS IP and why they may be important in your environment
1.3, “CS for z/OS IP implementation” on page 4	Functional overview of how CS for z/OS IP is implemented

1.1 Overview

z/OS Communications Server 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*.

CS for z/OS IP has 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 Manager 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 Manager (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.

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.

1.2 Featured functions

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 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.

The following list includes many of the technologies that have been implemented in the z/OS environment to complement TCP/IP.

- 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 CS for z/OS IP implementation

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 Manager (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 z/OS 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.

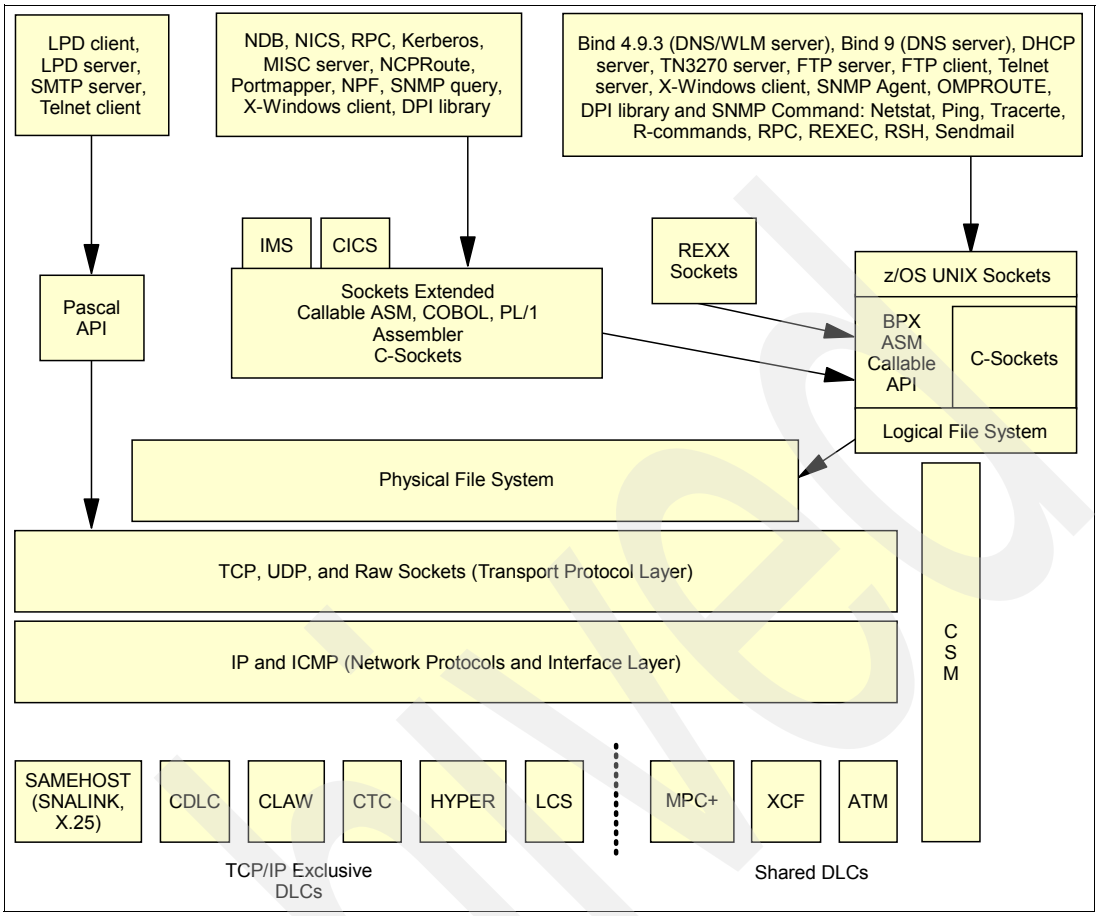


Figure 1-1 Functional overview

1.3.2 Operating environment

Because the z/OS UNIX environment is supported in the MVS environment, there is no need to discuss the creation of an MVS environment here. However, there are customization requirements on the UNIX Systems Services side of the environment that are needed 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 is the storage and input/output designs. The operating environment design features a tightly integrated storage and I/O model, known as Common Storage Manager (CSM).

Common Storage Manager

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 and 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. The figure 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 described here.

LAN Channel Station (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 z channels. In the past, this communication was provided by IUCV. Currently, SNALINK LU0, SNALINK LU6.2, and 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 those described here.

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 z server to the adapter.

    Segmentation offload is a feature of OSA-Express2. This interface is called IP Assist (IPA). Offloading reduces System z 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 z 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 only one routing application, called OMPROUTE. 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 141, for a detailed discussion about 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

We describe these items in more detail in the following sections

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 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 Communications Server.

The following TCP/IP socket APIs are included in the z/OS 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 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 *z/OS XL C/C++ Compiler and Run-Time Migration Guide for the Application Programmer,* GC09-4913, for complete documentation of the z/OS UNIX C sockets APIs.

Further guidance can be found in *z/OS UNIX System Services Programming Tools*, SA22-7805.

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*, SC31-8788, 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 V1R9 TCP/IP Implementation Volume 2: Standard Applications*, SG24-7533, and *z/OS Communications Server: IP Configuration Guide*, SC31-8775.

z/OS Communications Server also provides a number of specialized applications, including:

- Policy Agent for implementing networking and security policies in a z/OS environment
- Centralized or Distributed Policy Services
- Network Security Services (NSS)

These applications are discussed in detail in the IBM Redbooks publication *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 4: Security and Policy-Based Networking*, SG24-7535 and *z/OS Communications Server: IP Configuration Guide*, SC31-8775.

1.3.7 UNIX Systems Services

UNIX System Services is the z/OS Communications Server implementation of UNIX as defined by X/Open in 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, which is the default shell, and the tcsh shell (Ishell), which is 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. For this reason we present an overview of UNIX System Services next, to provide you with 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 UNIX System Services Planning, SA22-7800, provides a useful 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 similar to TSO/E, with a programming language of shell commands like those in 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 *z/OS CS: IP Migration,* GC31-8773, and *z/OS 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 *z/OS UNIX System Services Planning*, SA22-7800.

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 UNIX System Services Planning*, SA22-7800, 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 that users use to present themselves 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 next. 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, because 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 V1R9 TCP/IP Implementation Volume 4: Security and Policy-Based Networking*, SG24-7535.

Accessing the z/OS UNIX shells

You can access z/OS UNIX shells in the following ways:

- 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 can be used, 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 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.

Note: 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 on page 14). 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.

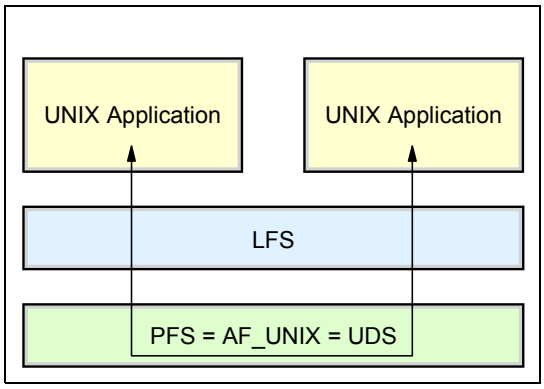


Figure 1-2 AF_UNIX sockets

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 47, or *z/OS 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.

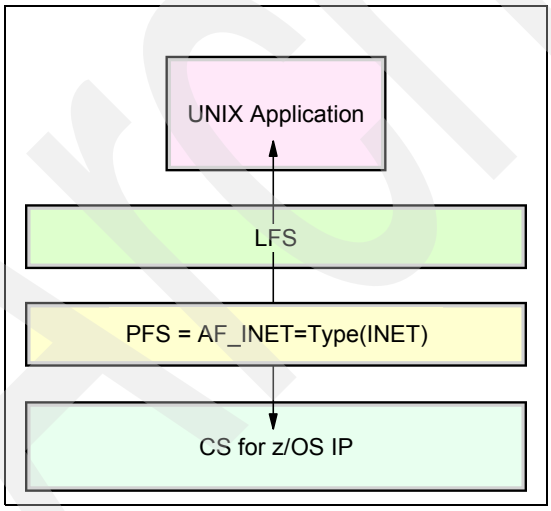


Figure 1-3 AF_INET sockets

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, it may seem strange to have a choice of using INET or CINET, because you are used to the TCP/IP protocol stack being an integral part of the UNIX operating system. However, 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 do 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 47.

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

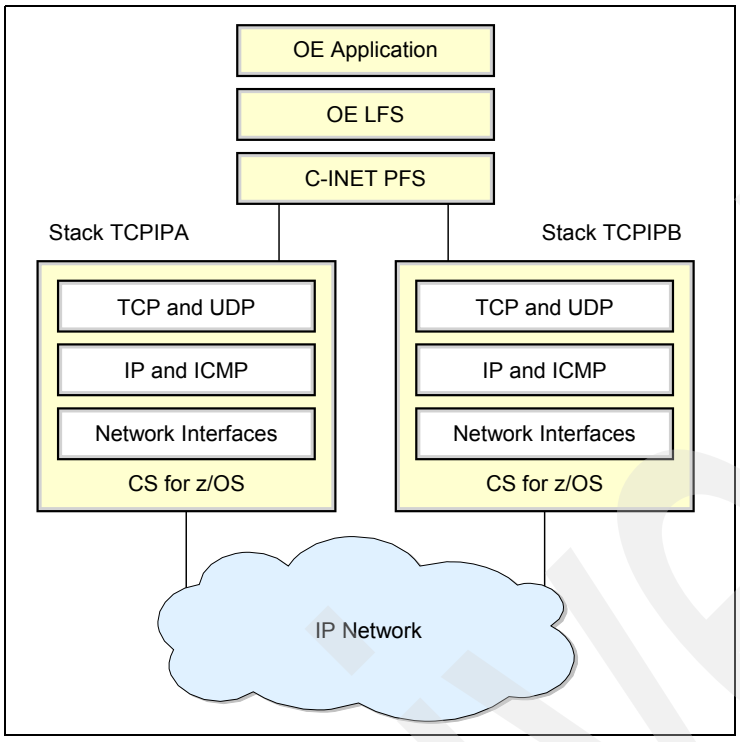


Figure 1-4 Multiple INET transport providers: CINET PFS

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.

1.4 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 Communications Server: IP Configuration Guide*, SC31-8775

  This document explains the major concepts of, and provides implementation guidance for, z/OS Communications Server functions.

- *z/OS Communications Server: IP Configuration Reference*, SC31-8776

  This document details the parameters or statements that can be used to implement z/OS Communications Server functions.

- *z/OS Communications Server: IP Programmer's Guide and Reference*, SC31-8787

  This document provides the guidelines for programming the IP applications on the z/OS.

- *z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference*, SC31-8788

  This document provides detailed information about the socket API for programming the IP applications on the z/OS.

For migration, the following publications are also helpful:

- *z/OS Communications Server: New Function Summary*, SC31-8771

  This document includes function summary topics to describe all the functional enhancements for the IP and SNA components of Communications Server, including task tables that identify the actions necessary to exploit new function. Use this document as a reference to using all the enhancements of z/OS Communications Server.

- *z/OS Planning for Installation*, GA22-7504

  This document helps you prepare to install z/OS by providing the information you need to write an installation plan.

- *z/OS Migration*, GA22-7499

  This document describes how to migrate (convert) from release to release. Use this document as a reference for keeping all z/OS applications working as they did in previous releases.

- *z/OS Introduction and Release Guide*, GA22-7502

  This document provides an overview of z/OS and lists the enhancements in each release. Use this document to determine whether to obtain a new release, and to decide which new functions to implement.

- *z/OS Summary of Message and Interface Changes*, SA22-7505

  This document describes the changes to interfaces for individual elements and features of z/OS. Use this document as a reference to the new and changed commands, macros, panels, exit routines, data areas, messages, and other interfaces of individual elements and features of z/OS.

2

The Resolver

TCP/IP protocols rely upon IP addressing in order to reach other hosts in a network to communicate. For ease of use, instead of using the IP addresses represented by numbers, they are sometimes mapped to easy-to-remember names. Typically, the names are assigned to the IP address of the servers that many users may access, such as Web servers, FTP servers, and TN3270 servers.

The Resolver function allows applications to use names instead of IP addresses to connect to other partners. The mapping of IP addresses and names is managed by name servers or local definitions. The Resolver queries those name servers, or searches local definitions, in order to convert the name to an IP address or the IP address to a name. Using the Resolver relieves users of having to remember the decimal or hexadecimal IP addresses.

The Resolver is important for enabling TCP/IP stacks or TCP/IP applications to establish connections to other hosts.

This chapter discusses the following topics.

Section	Topic
2.1, “Basic concepts of the Resolver” on page 20	Basic concepts of the Resolver
2.2, “The Resolver address space” on page 21	Key characteristics of the Resolver address space
2.3, “Implementing the Resolver” on page 32	The configuration and verification tasks of Resolver implementation
2.4, “Problem determination” on page 39	Problem determination techniques

2.1 Basic concepts of the Resolver

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

In most systems, in order for an 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(nnnnn)`** and **`gethostbyaddress(aaa.aaa.aaa.aaa)`**.

Figure 2-1 illustrates the information request and response flows. 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. After the relationship between the host name and IP address is established, the Resolver returns the response to the application.

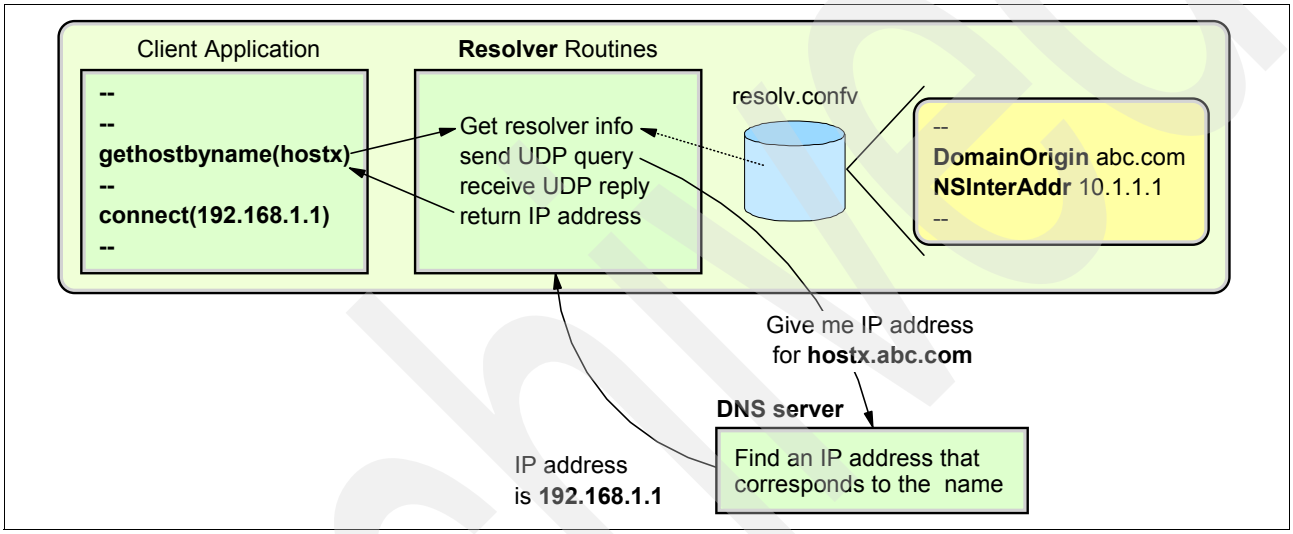


Figure 2-1 How the Resolver works

As mentioned, the Resolver function allows applications to use names instead of IP addresses to connect to other partners. Although using an IP address may seem to be an easy way to establish such a connection, for applications that need to connect to numerous partners, or for applications that are accessed by thousands of clients, using names is a much easier and more reliable form of establishing access.

Another important reason to use names instead of IP addressing is that a user or an application is not affected by the IP address changes to the underlying network.

Table 2-1 compares the benefits and drawbacks of the use of hard-coded IP addresses and the two name resolution methods: the local hosts file and the name server (DNS).

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 if 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.

2.2 The Resolver address space

In z/OS systems, the Resolver works as a procedure. The Resolver must be started before TCP/IP stacks or any TCP/IP applications issue the resolver calls. It can be started in one of the following ways:

- Default z/OS UNIX RESOLVER

  If no customized Resolver address space is configured, the z/OS USS starts the default Resolver. The default Resolver is named RESOLVER. To use the default RESOLVER address space, specify the RESOLVER_PROC(DEFAULT) statement or do not specify any RESOLVER_PROC statements in BPXPRMxx.

- Customized Resolver address space

  The customized Resolver address space can specify additional options to control the use of the Resolver configuration file. To create the Customized Resolver address space, create a Resolver started procedure and a SETUP data set to specify the additional options. The customized Resolver address space can be started automatically with the RESOLVER_PROC(*procname*) statement in BPXPRMxx.

Although the Resolver address space can be started manually, we recommend that you start the Resolver address space automatically during initialization of the UNIX System Services with the RESOLVER_PROC() statement within BPXPRMxx.

After the Resolver address space is activated, the global TCPIP.DATA statements cannot be overridden unless the MODIFY command is issued.

2.2.1 The Resolver SETUP data set

The Resolver SETUP data set is used by the customized Resolver address space. The default z/OS UNIX Resolver does not read this file. The SETUP data set can include the following statements:

- GLOBALTCPIPDATA
- DEFAULTTCPIPDATA
- GLOBALIPNODES
- DEFAULTIPNODES
- COMMONSEARCH or NOCOMMONSEARCH

The use of each statement is discussed in later sections.

2.2.2 The Resolver configuration file

The Resolver configuration file is often called TCPIP.DATA. In this file you can define how the Resolver should query the name-to-address or address-to-name resolution to the name servers or search the local hosts file.

The configuration file can be an MVS data set or a z/OS UNIX Hierarchical File System (HFS) file.

Note: The publication *z/OS 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.

TCPIP.DATA configuration statements

The following basic statements should be defined in the TCPIP.DATA file.

- TCPIPJOBNAME (equivalent to TCPIPUSERID)

  The name of the procedure used to start the TCP/IP address space. The default is TCPIP.

- DOMAIN (equivalent to DOMAINORIGIN)

  The domain origin that is appended to the host name to form the fully qualified domain name of a host.

- HOSTNAME

  The TCP host name of the z/OS Communications Server server.

- LOOKUP

  The order in which the DNS or local host files are to be searched for name resolution. By default, DNS is looked up first and if the resolution is unsuccessful, the local host files are searched next.

- NSINTERADDR (equivalent to NAMESERVER)

  The IP address of a name server the Resolver should query to.

- DATASETPREFIX

  The high-level qualifier for the dynamic allocation of data sets. DATASETPREFIX is referred to as the *hlq* of the TCP/IP stacks.

TCPIP.DATA search order

On z/OS, the configuration file is located based on the search order. You must be mindful of this search order, to ensure that the Resolver works in the way you expect.

The TCP/IP applications execute a set of commands in the Sockets API Library to initiate a request to the Resolver in z/OS. The Sockets API Library uses one of the following socket environments:

- Native MVS environment
- z/OS UNIX environment

Table 2-2 on page 23 lists some of the APIs, z/OS applications, and user commands that use the active MVS environment and the z/OS UNIX environment.

Table 2-2 Socket APIs, applications, and commands in Native MVS or z/OS UNIX environment

	Native MVS environment	z/OS UNIX environment
Socket APIs	TCP/IP C Sockets TCP/IP Pascal Sockets TCP/IP REXX Sockets TCP/IP Sockets Extended IMS Sockets CICS Sockets	Language Environment® C Sockets UNIX System Services
z/OS Applications	TN3270E Telnet server SMTP CICS Listener LPD Miscellaneous server PORTMAP RSHD	FTP OMPROUTE SNMP z/OS UNIX OPORTMAP z/OS UNIX OREXECD z/OS UNIX ORSHD
User commands	TSO FTP (batch) TSO NETSTAT TSO NSLOOKUP TSO PING TSO TRACERTE TSO DIG TSO LPR TSO REXEC TSO RPCINFO TSO RSH	TSO FTP (command) netstat nslookup ping traceroute ftp host hostname dnsdomainname dig rexec rpcinfo sendmail snmp

Each socket environment uses a different search order of the Resolver configuration file, as shown in Figure 2-2.

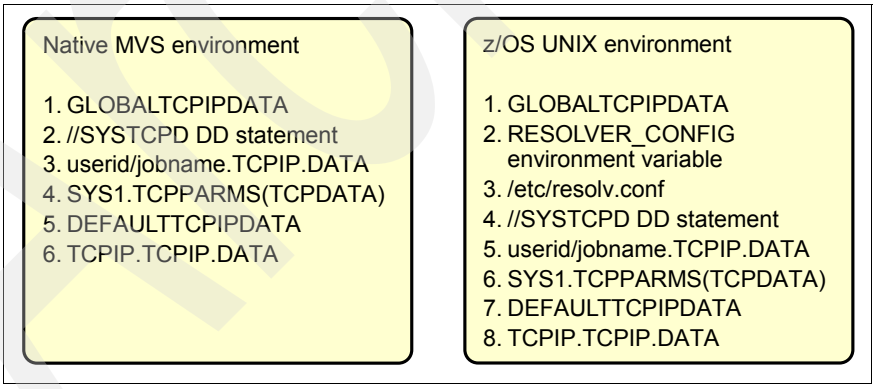


Figure 2-2 The Resolver configuration file search order for each socket environment

Note: USS Callable sockets use the z/OS UNIX environment search order, but cannot use the RESOLVER_CONFIG environment variable.

This provides the flexibility to control the Resolver lookup differently, depending on which socket API the application uses. However, because of the difference in search orders, it could sometimes cause an unexpected result in the address resolution.

For example, if you set up /etc/resolv.conf as your Resolver configuration file, the FTP server application that uses the z/OS UNIX search order can resolve the name-to-address or address-to-name successfully. However, the TN3270 server, which uses the native MVS search order, would fail because /etc/resolv.conf is not included in its search list.

Using GLOBALTCPIPDATA

In order to deal with the complexity of the different search orders in the environments, the GLOBALTCPIPDATA statement was introduced. Using the GLOBALTCPIPDATA statement, you can use the same Resolver configuration file throughout the z/OS system, because it is the first choice in all socket search orders. This consolidation allows for consistent name resolution processing across all TCP/IP applications.

To specify the GLOBALTCPIPDATA statement, you need to create a Resolver started procedure and its SETUP data set, instead of using the z/OS USS default RESOLVER address space. The use of the Resolver address space and GLOBALTCPIPDATA statement simplifies the Resolver configuration on z/OS.

The TCPIP.DATA file specified by the GLOBALTCPIPDATA statement is often called the global TCPIP.DATA file. If you define GLOBALTCPIPDATA, the following statements can only be included in the global TCPIP.DATA file:

- DomainOrigin/Domain or Search
- NSInterAddr/NameServer
- NSPortAddr
- ResolveVia
- ResolverTimeOut
- ResolverUDPRetries
- SortList

Other TCPIP.DATA statements can be optionally included in the global TCPIP.DATAfile, and the definition in the global TCPIP.DATA always has precedence. If TCPIPJobname is specified in both the global TCPIP.DATA file and the local (non-global) TCPIP.DATA file, then the one in the global TCPIP.DATA file is used.

If other TCPIP.DATA statements, such as HostName and TCPIPJobname, cannot be found in the global TCPIP.DATA file, then the Resolver continues its search according to the search order of the each socket environment. The search stops when the file is found.

If statements such as HostName and TCPIPJobname cannot be found in that file either, the defaults are applied. Note that it does not continue searching in the list. In other words, a maximum of two files can be used (global TCPIP.DATA file and one TCPIP.DATA file in the search order list).

Using GLOBALTCPIPDATA, the administrators can specify which statements should be applied throughout the z/OS image, and decide which statements can be customized by each socket environment by omitting those statements in the global TCPIP.DATA file.

Note: In the Common INET (CINET) multi-stack environment, you should omit the TCPIPJobname statement from the global TCPIP.DATA file so that each TCP/IP stack, or the applications that have affinity to a stack, can specify a local TCP.DATA with its own TCPIPJobname statement.

When using GLOBALTCPIPDATA in the CINET environment, the name server specified by NSInterAddr or NameServer in the global TCPIP.DATA file must be accessible from all TCP/IP stacks that issue Resolver calls.

Figure 2-3 depicts the relationship between global TCPIP.DATA and local TCPIP.DATA.

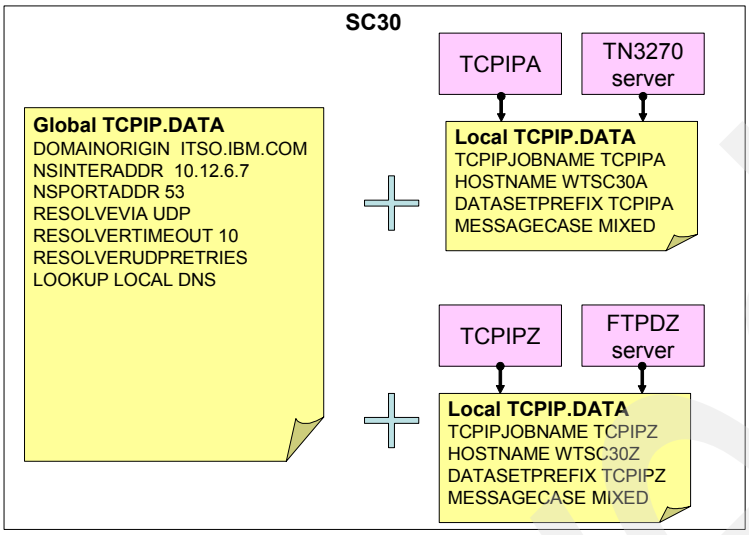


Figure 2-3 Using global TCPIP.DATA and Local TCPIP.DATA

Using DEFAULTTCPIPDATA

DEFAULTTCPIPDATA can be specified in the Resolver SETUP data set to define the last choice of the TCPIP.DATA in the search order. The file specified by DEFAULTTCPIPDATA is used when the application does not specify the local (non-global) TCPIP.DATA.

2.2.3 Local hosts file

The local hosts file lists the mapping of the IP addresses and the names just like the name servers, but held locally on the server. The LOOKUP statement in the TCPIP.DATA configuration file defines whether the Resolver address space performs the name resolution only in the local files, or using the defined name server, or both, in any specified order.

Using COMMONSEARCH

When the local hosts file is searched, the search order for the native MVS environment and the z/OS UNIX environment are different. The difference in the search orders adds complexity to configuration tasks and can lead unexpected results of the name resolution.

The simpler approach is to utilize the COMMONSEARCH statement in the Resolver SETUP data set. By specifying COMMONSEARCH, native MVS and z/OS UNIX environments use the same search order as shown in Figure 2-4 on page 26 (except the RESOLVER_IPNODES environment variable, which is only supported by the z/OS UNIX environment). In both environments, the first choice is the file specified by GLOBALIPNODES statement, which is defined in the Resolver SETUP data set.

The local hosts files looked up in this search order are typically called ETC.IPNODES files. When the COMMONSEARCH is specified in the Resolver SETUP data set, it uses the same search order for both IPv4 and IPv6 queries. You can list both IPv4 and IPv6 addresses in the ETC.IPNODES file.

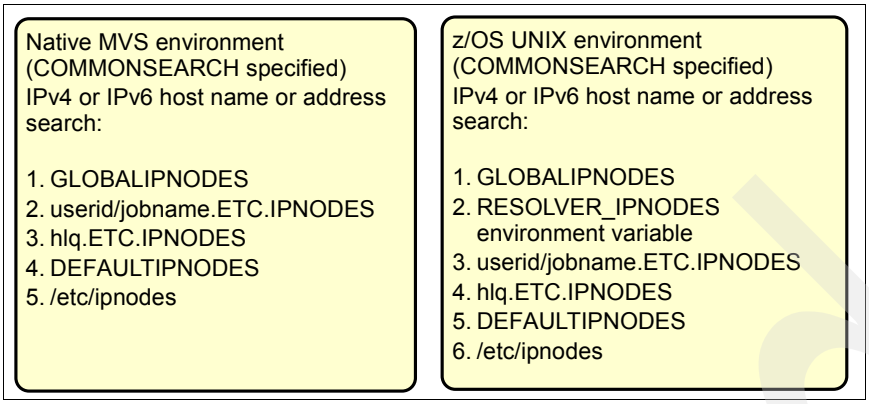


Figure 2-4 Local hosts file search order with COMMONSEARCH specified

To determine which environment is used for a particular socket's APIs, applications, or commands, refer to Table 2-2 on page 23.

If COMMONSEARCH is not specified in the Resolver SETUP data set, then the default is NOCOMMONSEARCH and the default search order shown in Figure 2-5 is used.

Using GLOBALIPNODES

The GLOBALIPNODES statement specifies the global local host file that is to be used in the entire z/OS image, regardless of which environment (native MVS or z/OS UNIX) that the applications or sockets API use. To put the GLOBALIPNODES statement into effect for the name resolution of IPv4 addresses, also specify COMMONSEARCH in the Resolver SETUP data set.

Using DEFAULTIPNODES

The DEFAULTIPNODES statement specifies the last candidate of the local host file search. To put the DEFAULTIPNODES statement into effect for the name resolution of IPv4 addresses, also specify COMMONSEARCH in the Resolver SETUP data set.

Default local hosts file search order

If NOCOMMONSEARCH (the default) is specified in the Resolver SETUP data set or default z/OS UNIX Resolver is used, the default local hosts file search order shown in Figure 2-5 is used. The default local hosts file search order only applies to the query of IPv4 addresses. The query for IPv6 addresses always uses the search order listed in Figure 2-4.

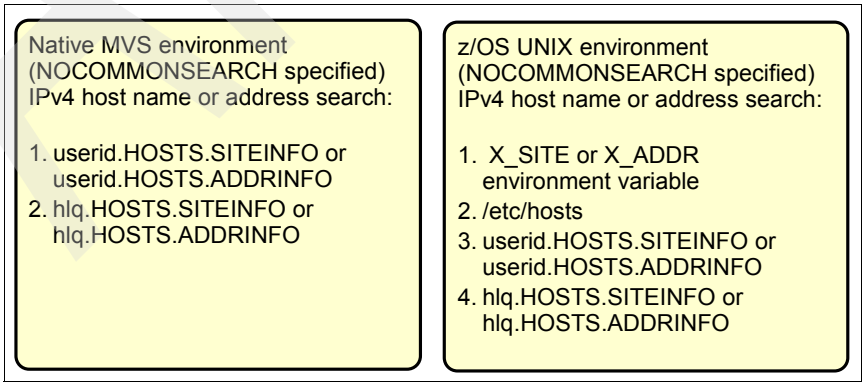


Figure 2-5 Local hosts file search order with NOCOMMONSEARCH specified (default)

2.2.4 Affinity servers and generic servers

In the multiple stack environment, a TCP/IP application may have an affinity to a specific TCP/IP stack. When designing a multiple stack system, it is important to check each application that will be used, and how it will be implemented in the environment.

Affinity server

An *affinity server* is an application that has affinity to a specific TCP/IP stack; it provides service to the clients that are connected through the TCP/IP stack to the applications.

In this case, you need to code a TCP/IPJobname statement that represents the application in order to direct traffic to a specific stack. So, when designing the global definitions in the Resolver address space, do not code a TCPIPJobname statement in GLOBALTCPIPDATA. Instead, allow it to be coded in the local TCPIP.DATA.

- A native TCP/IP sockets program will always use one stack only, and by default, it will be 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 2-6.

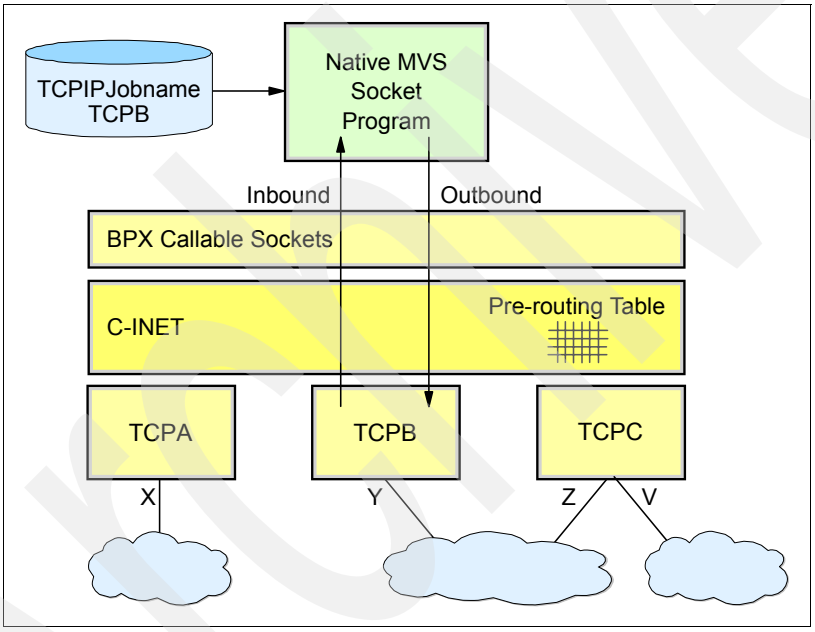


Figure 2-6 Native TCP/IP applications in a multiple stack environment

- 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 2-7 on page 28.

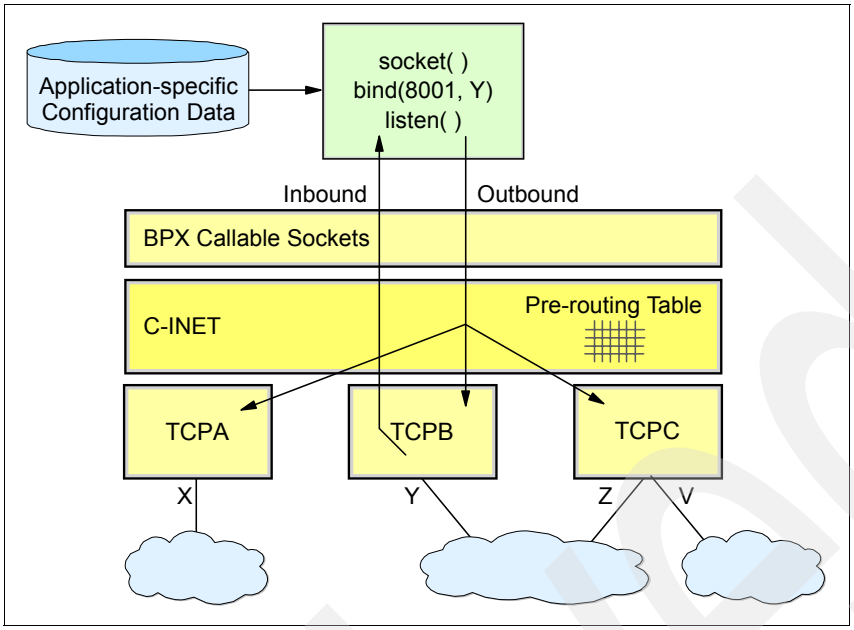


Figure 2-7 USS callable APIs or LE C/C++ sockets APIs using specific bind in a multi-stack environment

Generic server

A *generic server* is a server without an affinity to a specific stack, and it provides service to any clients that are connected to any TCP/IP stacks on the system.

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.

- 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 29.

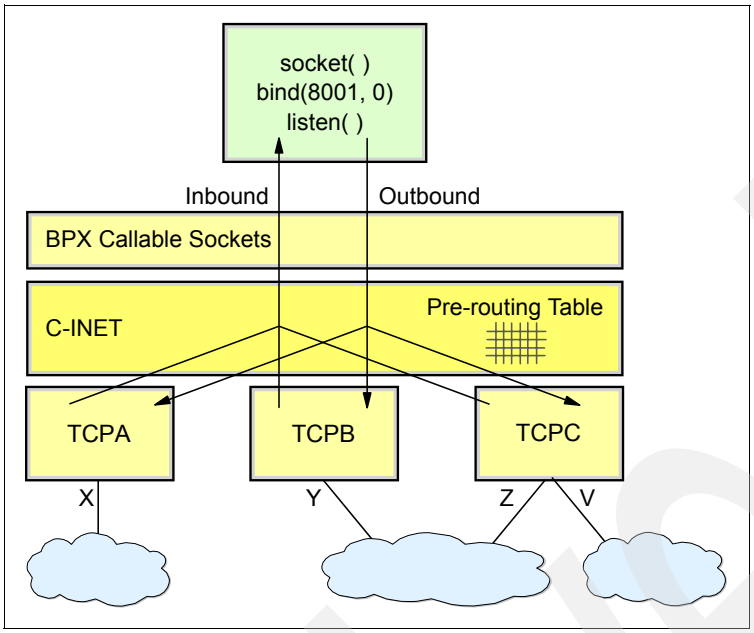


Figure 2-8 USS callable APIs or LE C/C++ sockets APIs using generic bind in a multi-stack environment

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 stack has it reserved to another address space, the bind() call fails.

2.2.5 Resolving IPv6 address

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.
- An algorithm is defined to describe how a Resolver needs to sort a list of IP addresses returned for a multihomed host.
- 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).

IPv6 Resolver statements

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 IPv6 search order is same as the COMMONSEARCH search order, as shown in Figure 2-4 on page 26. If you do not want to use the COMMONSEARCH search order for existing IPv4 local hosts files, you may need to maintain two different local host files (for example, IPv4 addresses in HOSTS.LOCAL, and IPv6 and IPv4 addresses in ETC.IPNODES).

Name and address resolution functions

The APIs such as getaddrinfo, getnameinfo, and freeaddrinfo allow applications to resolve host names to IP addresses and vice versa for IPv6. 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, just like IPv4 address resolution. These specifications determine whether the APIs will query a name server first and 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 local host file 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.

2.2.6 Considerations

To implement the Resolver address space, it is important to first determine whether your environment requires 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.

The statements defined in the global TCPIP.DATA cannot be overridden by the local TCPIP.DATA file of the each TCP/IP stack. The local TCPIP.DATA file can only specify the statement if it is not already defined in the global TCPIP.DATA file.

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 43.

The Resolver in a single stack environment

We recommend that you create a global TCPIP.DATA file for a single stack environment. The TCPIPJobname statement can be coded in a global TCPIP.DATA file or in the local (non-global) TCPIP.DATA file, because there is only one stack on the system. If some applications have requirements to specify their own TCPIP.DATA statements, then omit them from the global TCPIP.DATA so the applications can point to the local TCPIP.DATA file to be used.

The Resolver in a multiple stack environment

When implementing for a multiple stack environment, each TCP/IP stack should use a local TCPIP.DATA file specifying stack-specific statements, such as TCPIPJobname and HostName. Optionally, you can merge some statements that can be applied to all TCP/IP stacks and all TCP/IP applications to a global TCPIP.DATA file. You need to determine which statements should be defined in the global TCPIP.DATA and used in the entire z/OS image. This will depend on how much you want to allow each stack or application to define its own definitions.

In the multiple stack environment, we recommend that you create a global TCPIP.DATA if all the statements needed in the global TCPIP.DATA (see “Using GLOBALTCPIPDATA” on page 24) can be applied to all the stacks as shown in Figure 2-3 on page 25. If not, do not use the global TCPIP.DATA and only use local TCPIP.DATA for each stack. Figure 2-9 on page 32 depicts the multiple stack environment without the use of a global TCPIP.DATA.

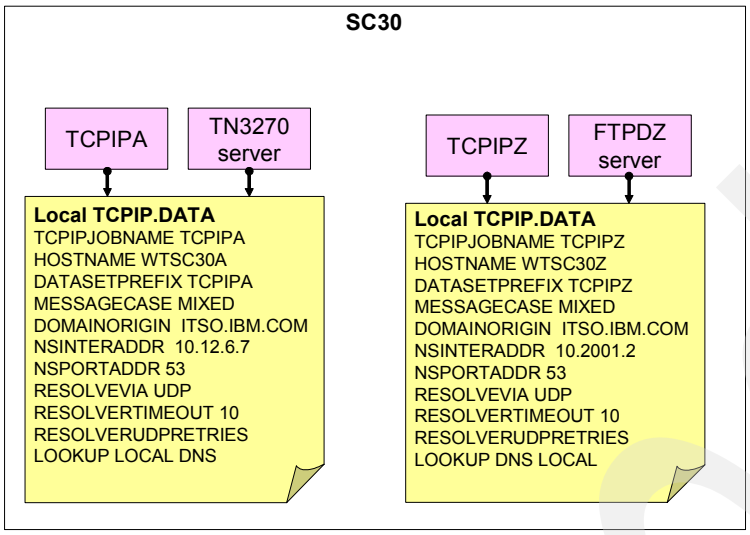


Figure 2-9 The multiple stack environment without global TCPIP.DATA

Recommendation: Although there are specialized cases where multiple stacks per LPAR can provide value, generally we 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 Implementing the Resolver

In this scenario we use the customized Resolver address space and specify GLOBALTCPIPDATA, DEFAULTTCPIPDATA, and GLOBALIPNODES in the Resolver SETUP data set. We define a global TCPIP.DATA file and define a common set of parameters for entire z/OS image. We leave some statements omitted from the global TCPIP.DATA file so the applications or TCP/IP stack can use their own local TCPIP.DATA file for the statements undefined in the global TCPIP.DATA file.

Figure 2-10 depicts the environment we use for this implementation.

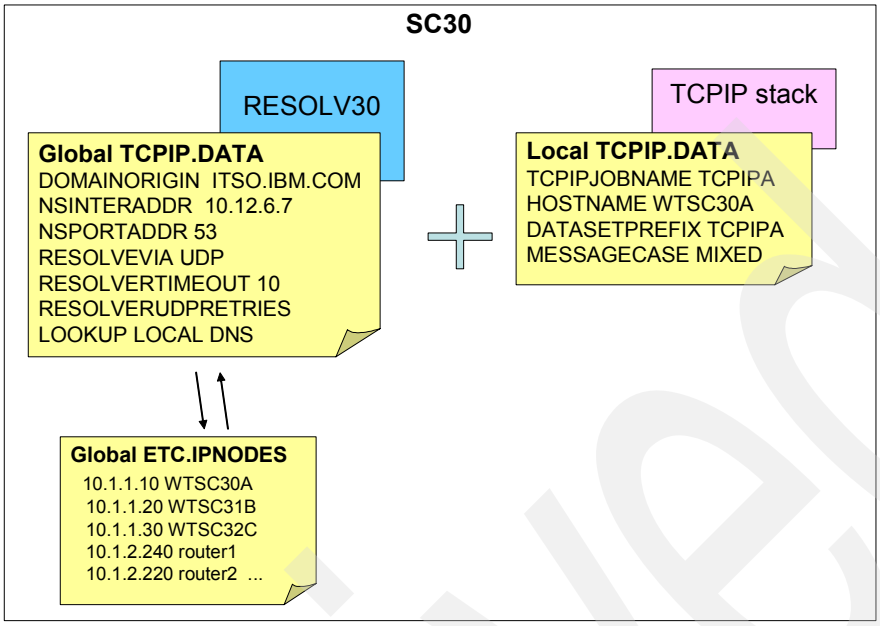


Figure 2-10 Our Resolver environment on SC30

2.3.1 Implementation tasks

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

1. Set up the Resolver started procedure.
2. Customize BPXPRMxx.
3. Configure the Resolver SETUP data set.
4. Create the global TCPIP.DATA file.
5. Create the default TCPIP.DATA file.
6. Create the global ETC.IPNODES data set.
7. Create a TCPIP.DATA file for TCPIPA stack.
8. Create the TCPIPA stack started procedure.

Set up the Resolver started procedure

We created the Resolver procedure to be started during the UNIX System Services initialization.

To create the procedure, we copied the sample procedure hlq.SEZAINST(EZBREPRC) and customized it to our environment, as shown in Example 2-1. 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.

Example 2-1 The Resolver started procedure

```
/*****************************************
/* SYS1.PROCLIB(RESOLV30)
/*****************************************
//RESOLV30 PROC PARMS='CTRACE(CTIRES00)' 1
//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=TCPIPA.TCPPARMS(RESOLV&SYSCLONE),DISP=SHR,FREE=CLOSE 2
```

- 1 The name of the Resolver procedure is RESOLV30.
- 2 Specifies the Resolver SETUP data set. The &SYSCLONE MVS system symbol value on this system is 30.

Customize BPXPRMxx

We customized the RESOLVER_PROC statement in BPXPRMxx, to specify the procedure name we used. This will cause the Resolver to start automatically the next time z/OS USS initializes. Example 2-2 shows the partial contents of BPXPRMxx.

Example 2-2 Specifying the Resolver procedure to be started

```
/*****************************************
/* SYS1.PARMLIB(BPXPRM00)
/*****************************************
/* 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(RESOLV30) 1
```

- 1 Specifies the name of the Resolver procedure we created in the previous step.

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. Otherwise, the procedure will not be found and the Resolver will not start. SYS1.PROCLIB is usually one of the libraries specified there.

Configure the Resolver SETUP data set

We configured the Resolver SETUP data set which is specified with the SETUP DD definition in the Resolver started procedure. This data set defines the location of the global and default

TCPIP.DATA files containing the parameters we wanted to be defined in the z/OS environment.

In our test environment, we copied the SETUP sample data set hlq.SEZAINST(RESSETUP) and changed its contents to meet our requirements, as shown in Example 2-3.

Example 2-3 Resolver address space SETUP data set

```
; ****************************************
; TCPIPA.TCPPARMS(RESOLV30)
; ****************************************
GLOBALTCPIPDATA('TCPIPA.TCPPARMS(GLOBAL)')    1
DEFAULTTCPIPDATA('TCPIPA.TCPPARMS(DEFAULT)') 2
GLOBALIPNODES('TCPIPA.TCPPARMS(IPNODES)')     3
COMMONSEARCH                                  4
```

- 1 Specifies the first choice of the TCPIP.DATA file.
- 2 Specifies the last choice of the TCPIP.DATA file.
- 3 Specifies the first choice of the local hosts file.
- 4 The COMMONSEARCH search order is used. This statement is needed to have GLOBALIPNODES to be applied.

Important: Plan carefully to create these global parameters. The definitions in the Resolver SETUP data set is applied to all TCP/IP stacks or applications.

Create the global TCPIP.DATA file

In this step, we provided the global statements that all TCP/IP stacks and applications used in our z/OS environment. To define these statements we copied the sample TCPIP.DATA file provided in hlq.SEZAINST(TCPDATA) and customized the statements, as shown in Example 2-4.

Example 2-4 Global TCPIP.DATA file

```
; ****************************************
; TCPIPA.TCPPARMS(GLOBAL)
; ****************************************
DOMAINORIGIN  ITSO.IBM.COM     1
NSINTERADDR  10.12.6.7         2
NSPORTADDR 53
RESOLVEVIA UDP
RESOLVERTIMEOUT 10
RESOLVERUDPRETRIES 1
LOOKUP LOCAL DNS               3
```

- 1 Specifies the list of domain names appended to the host name when the search is performed.
- 2 Specifies the IP address of the DNS server.
- 3 The local hosts file is looked up first. If it fails, the DNS server is searched.

Important: If GLOBALTCPIPDATA is specified:

- Any statements contained in the global TCPIP.DATA file will take precedence over any statements in local TCPIP.DATA file found by way of the appropriate environment's (Native z/OS or z/OS UNIX) search order.
- The TCPIP.DATA statements in Example 2-4 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.

Create the default TCPIP.DATA file

We created a default TCPIP.DATA file as shown in Example 2-5 to be the last choice of the local TCPIP.DATA search order. It is used when the application does not specify the local TCPIP.DATA explicitly.

Example 2-5 Default TCPIP.DATA file

```
; ****************************************
; TCPIPA.TCPPARMS(DEFAULT)
; ****************************************
TCPIPJOBNAME TCPIP 1
HOSTNAME WTSC30    2
```

- 1 Specifies the default TCP/IP procedure name.
- 2 Specifies the default host name.

Important: 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, it can potentially cause errors in a multi-stack environment.

Create the global ETC.IPNODES data set

We created the global ETC.IPNODES data set, which is referred as GLOBALIPNODES in the Resolver SETUP data set. It contains name-to-address mappings. This data set is used for the local search to resolve a name into an IP address or vice versa.

We chose to use the COMMONSEARCH, because it allowed us to have a common local search environment with IPv4 or IPv6 hosts. Example 2-6 shows the contents of the GLOBALIPNODES data set. When using COMMONSEARCH, only the IPNODES data set is used.

Example 2-6 GLOBALIPNODES data set

```
; ****************************************
; TCPIPA.TCPPARMS(IPNODES)
; ****************************************
  10.12.6.7   OURDNS                1
  10.1.1.10   WTSC30A               1
  10.1.1.20   WTSC31B               1
  10.1.1.30   WTSC32C               1
  10.1.2.240  router1               1
  10.1.2.220 router2                1
  1::2 TESTIPV6ADDRESS1             2
  1:2:3:4:5:6:7:8 TESTIPV6ADDRESS2  2
```

- 1 The mapping of a name and a IPv4 address is listed.
- 2 The mapping of a name and a IPv6 address is listed.

Create a TCPIP.DATA file for TCPIPA stack

We created a local TCPIP.DATA file for the TCPIPA stack with file name DATAA30.

Example 2-7 TCPIP.DATA file DATAA30

```
; ****************************************
; TCPIPA.TCPPARMS(DATAA30)
; ****************************************
TCPIPJOBNAME TCPIPA               1
HOSTNAME WTSC30A                  2
```

```
DATASETPREFIX TCPIPA
MESSAGECASE MIXED
```

- 1 Specifies the procedure name of TCPIPA stack.
- 2 Specifies the host name of the TCPIPA stack.

Create the TCPIPA stack started procedure

We created the TCPIPA stack procedure (RESOLVER_CONFIG) and pointed to TCPIPA.TCPPARMS(DATAA30), 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 2-8.

Example 2-8 TCPIPA procedure

```
/*****************************************
/* SYS1.PROCLIB(TCPIPA)
/*****************************************
//TCPIPA    PROC PARMS='CTRACE(CTIEZB00),IDS=00', 1
//             PROFILE=PROFA&SYSCLONE.,TCPDATA=DATAA&SYSCLONE
//TCPIPA    EXEC PGM=EZBTCPIP,REGION=0M,TIME=1440,
//             PARM=('&PARMS',
//         'ENVAR("RESOLVER_CONFIG=//''TCPIPA.TCPPARMS(&TCPDATA)''")') 2
//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.) 3
```

- 1 The TCP/IP procedure name is TCPIPA.
- 2 The local TCPIP.DATA is specified.
- 3 The TCP/IP profile is specified (the TCP/IP configuration file example is not shown in this chapter).

2.3.2 Activation and verification

To verify the Resolver address space was working as we expected, we followed these steps:

1. Stop the default z/OS UNIX Resolver.
2. Start the Resolver address space.
3. Display the Resolver address space configuration.
4. Use the **ping** command to verify the name resolution.

To implement our Resolver address space, we first halted the running Resolver using the STOP command, as shown in Example 2-9.

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

Stop the default z/OS UNIX Resolver

In our current environment, the default z/OS UNIX Resolver was running. We stopped this default Resolver to run the customized Resolver.

Example 2-9 Stopping the Resolver address space

```
P 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
```

Start the Resolver address space

We started the customized Resolver address space using the procedure we created in the previous step, as shown in Example 2-10.

Example 2-10 Starting a configured Resolver address space

```
S RESOLV30,SUB=MSTR
IRR812I PROFILE ** (G) IN THE STARTED CLASS WAS USED 366
        TO START RESOLV30 WITH JOBNAME RESOLV30.
EZZ9298I DEFAULTTCPIPDATA - TCPIPA.TCPPARMS(DEFAULT) 1
EZZ9298I GLOBALTCPIPDATA - TCPIPA.TCPPARMS(GLOBAL)   2
EZZ9298I DEFAULTIPNODES - None
EZZ9298I GLOBALIPNODES - TCPIPA.TCPPARMS(IPNODES)    3
EZZ9304I COMMONSEARCH                                4
EZZ9291I RESOLVER INITIALIZATION COMPLETE
$HASP395 BPXAS    ENDED
```

- 1 The correct DEFAULTTCPIPDATA file is applied.
- 2 The correct GLOBALTCPIPDATA file is applied.
- 3 The correct GLOBALIPNODES file is applied.

Note: If you want to start the default z/OS UNIX Resolver, use the following command instead.

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

If you want to reload the SETUP dataset content changes, use the MODIFY command to refresh the Resolver. To show how this is done, we have created a new SETUP data set named NEWSETUP, without GLOBALTCPIPDATA or DEFAULTTCPIPDATA definitions, and refreshed the Resolver to reflect the changes; see Example 2-11.

Example 2-11 Modifying the Resolver address space

```
F RESOLV30,REFRESH,SETUP=TCPIPA.TCPPARMS(NEWSETUP)
EZZ9298I DEFAULTTCPIPDATA - None
EZZ9298I GLOBALTCPIPDATA - None
EZZ9298I DEFAULTIPNODES - TCPIPA.TCPPARMS(IPNODES)
EZZ9298I GLOBALIPNODES - None
EZZ9304I COMMONSEARCH
EZZ9293I REFRESH COMMAND PROCESSED
```

Display the Resolver address space configuration

To verify the correct configuration file is applied to the Resolver address space, use the MODIFY command with the display option, as shown in Example 2-12.

Example 2-12 Modify Resolver with display option

```
F RESOLV30,DISPLAY
```

```
EZZ9298I DEFAULTTCPIPDATA - TCPIPA.TCPPARMS(DEFAULT) 1
EZZ9298I GLOBALTCPIPDATA - TCPIPA.TCPPARMS(GLOBAL)   2
EZZ9298I DEFAULTIPNODES - None
EZZ9298I GLOBALIPNODES - TCPIPA.TCPPARMS(IPNODES)    3
EZZ9304I COMMONSEARCH
EZZ9293I DISPLAY COMMAND PROCESSED
```

- 1 The correct DEFAULTTCPIPDATA file is applied.
- 2 The correct GLOBALTCPIPDATA file is applied.
- 3 The correct GLOBALIPNODES file is applied.

Use the ping command to verify the name resolution

Verify that the Resolver is able to perform the expected name-to-address resolution by using the **ping** command, as shown in Example 2-14. As you can see, the name *router1* has resolved to address 10.1.2.240.

Example 2-13 UNIX ping command display

```
CSO3 @ SC3O:/u/csO3>ping router1
CS V1R9: Pinging host router1 (10.1.2.240)
Ping #1 response took 0.000 seconds.
```

The TSO PING command was also successful, as shown in Example 2-14.

Example 2-14 TSO PING command display

```
TSO PING ROUTER1
CS V1R9: Pinging host router1 (10.1.2.240)
 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 file. For an explanation of how this is done and what the contents of this trace will be, refer to 2.4, "Problem determination" on page 39.

2.4 Problem determination

To diagnose Resolver problems, you can use two kinds of trace tools:

- Trace Resolver

  This provides information that can be helpful in debugging problems that an application program could have with using Resolver facilities (for example, GetHostByName or GetHostByAddr).

- Component Trace RESOLVER (SYSTCPRE)

  This is useful 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. For more information about Resolver diagnosis, refer to *z/OS Communications Server: IP Diagnosis Guide*, GC31-8782.

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 2-3).

Table 2-3 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 that 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 LOCAL DNS statement 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:

- Check whether the correct Resolver data sets is in use. If an unexpected TCPIP.DATA file is used, check the search orders of the data set.
- Check whether 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, so ask the DNS administrator for help.

Trace Resolver can be activated in the following ways, in its precedence order:

1. The RESOLVER_TRACE environment variable (z/OS UNIX environment only).
2. SYSTCPT DD allocation.
3. TRACE RESOLVER or OPTIONS DEBUG statements (you must allocate STDOUT or SYSPRINT to generate trace data).
4. The resDebug bit set to on in the _res structure option field (you must allocate STDOUT or SYSPRINT to generate trace data).

Next, we illustrate using Trace Resolver in a z/OS UNIX environment, and in a TSO environment.

Using Trace Resolver in z/OS UNIX environment

Example 2-15 shows how to enable and disable the Trace Resolver in z/OS UNIX environment.

Example 2-15 Using Trace Resolver in a z/OS UNIX environment

```
CS06 @ SC30:/u/cs06>export RESOLVER_TRACE=/u/cs06/trace1.txt 1
CS06 @ SC30:/u/cs06>ping router1 2
CS V1R9: Pinging host router1 (10.1.2.240)
Ping #1 response took 0.000 seconds.
CS06 @ SC30:/u/cs06>set -A RESOLVER_TRACE 3
CS06 @ SC30:/u/cs06>obrowse /u/cs06/trace1.txt
```

- 1 To enable the Trace Resolver, set the RESOLVER_TRACE environment variable. This command directs the output to the /u/cs06/trace1.txt HFS file. You can also direct the output to STDOUT by specifying RESOLVER_TRACE=STDOUT. If you want to direct it to a new MVS data set, specify RESOLVER_TRACE="//'SOME.MVS.DATASET'".
- 2 After enabling a Trace Resolver, perform a z/OS UNIX shell command that invokes a resolver call.
- 3 This command disables the Trace Resolver.

Using Trace Resolver in a TSO environment with SYSTCPT DD

Example 2-16 shows how to enable and disable the Trace Resolver in a TSO environment environment.

Example 2-16 Using Trace Resolver in a TSO environment

```
alloc dd(systcpt) da(*) 1
ping router1            2
free dd(systcpt)        3
```

- 1 To enable the Trace Resolver, allocate a SYSTCPT dataset. If you specify da(*), the Trace Resolver output to a TSO terminal. If you want to direct the output to a specific data set, specify da('SOME.DATASET.NAME').
- 2 After enabling the Trace Resolver, perform a TSO command that invokes a Resolver call.
- 3 To disable theTrace Resolver, free a SYSTCPT dataset.

> **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.

Using Trace Resolver for applications with TCPIP.DATA statements

Allocate STDOUT or SYSPRINT (as a DD statement in the procedure) as an output data set, and define the statement TRACE RESOLVER or OPTIONS DEBUG in the first line of the TCPIP.DATA file that is being used by the application, as shown in Example 2-17. Start the application that invokes a Resolver call.

Example 2-17 Using the OPTIONS DEBUG to get a trace of the resolver

```
OPTIONS DEBUG 1
TCPIPJOBNAME TCPIPA
HOSTNAME WTSC30A
DOMAINORIGIN  ITSO.IBM.COM
DATASETPREFIX TCPIPA
MESSAGECASE MIXED
```

```
NSINTERADDR  10.12.6.7
NSPORTADDR 53
```

- 1 Specify OPTIONS DEBUG or TRACE RESOLVER to enable Trace Resolver.

Display the output of the Trace Resolver

Example 2-18 shows the output of the Trace Resolver in the z/OS UNIX environment (which was taken from Example 2-15 on page 41).

Note that the Trace Resolver taken in the TSO environment (Example 2-16 on page 41) is almost identical.

Example 2-18 Trace Resolver output - z/OS UNIX shell environment

```
Resolver Trace Initialization Complete -> 2007/09/28 11:31:14.088323

res_init Resolver values:
 Global Tcp/Ip Dataset  = TCPIPA.TCPPARMS(GLOBAL) 1
 Default Tcp/Ip Dataset = TCPIPA.TCPPARMS(DEFAULT)
 Local Tcp/Ip Dataset   = /etc/resolv.conf 2
 Translation Table      = Default
 UserId/JobName         = CS06
 Caller API             = LE C Sockets
 Caller Mode            = EBCDIC
 (L) DataSetPrefix = TCP
 (L) HostName      = wtsc30
 (L) TcpIpJobName  = TCPIP
 (G) Search        = ITSO.IBM.COM
                     IBM.COM
 (G) NameServer    = 10.12.6.7
 (G) NsPortAddr    = 53            (G) ResolverTimeout    = 10
 (G) ResolveVia    = UDP           (G) ResolverUdpRetries = 1
 (*) Options NDots = 1
 (*) SockNoTestStor
 (*) AlwaysWto     = NO            (L) MessageCase        = MIXED
 (G) LookUp        = LOCAL DNS 3
res_init Succeeded
res_init Started: 2007/09/28 11:31:14.127535
res_init Ended: 2007/09/28 11:31:14.127540
********************************************************************************
GetAddrInfo Started: 2007/09/28 11:31:14.127575
GetAddrinfo Invoked with following inputs:
   Host Name:  router1 4
   No Service operand specified
   Hints parameter supplied with settings:
       ai_family = 0, ai_flags = 0x00000062
       ai_protocol = 0, ai_socktype = 0
GetAddrInfo Opening Socket for IOCTLs
 BPX1SOC:  RetVal = 0, RC = 0, Reason = 0x00000000
GetAddrInfo Opened Socket 0x00000004
GetAddrInfo Only IPv4 Interfaces Exist
GetAddrInfo Searching Local Tables for IPv4 Address
Global IpNodes Dataset  = TCPIPA.TCPPARMS(IPNODES) 5
Default IpNodes Dataset = None
Search order            = CommonSearch
 SITETABLE from globalipnodes TCPIPA.TCPPARMS(IPNODES)
 - Lookup for router1.ITSO.IBM.COM
 - Lookup for router1.IBM.COM
 - Lookup for router1 6
 ADDRTABLE from globalipnodes TCPIPA.TCPPARMS(IPNODES)
```

```
 - Lookup for 10.1.2.240 7
GetAddrInfo Returning Zero as Port Number
GetAddrInfo Built 1 Addrinfos
GetAddrInfo Closing IOCTL Socket 0x00000004
 BPX1CLO:  RetVal = 0, RC = 0, Reason = 0x00000000
GetAddrInfo Succeeded:  IP Address(es) found:
  IP Address(1) is 10.1.2.240
GetAddrInfo Ended: 2007/09/28 11:31:14.199817
*********************************************************************
FreeAddrInfo Started: 2007/09/28 11:31:14.199862
FreeAddrInfo Called to free addrinfo structures
FreeAddrInfo Succeeded, Freed 1 Addrinfos
FreeAddrInfo Ended: 2007/09/28 11:31:14.199874
*********************************************************************
```

- 1 Informs the global TCPIP.DATA in use.
- 2 Informs the local TCPIP.DATA in use.
- 3 The local hosts file is looked up first, followed by the DNS server if it fails.
- 4 *router1* host name is looked up.
- 5 Informs the global ETC.IPNODE in use.
- 6 *router1* entry is looked up in the global ETC.IPNODES file.
- 7 *router1* is resolved to 10.1.2.240.

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 128K. The maximum size is 128M. The default size is 16M. Trace records can optionally be written to an external writer.

The Resolver CTRACE can be started any time needed by using the TRACE CT command, or it can be activated during Resolver procedure initialization.

Using CTRACE for RESOLVER

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 2-19.

Example 2-19 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 RESOLV30,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 2-20 on page 44.

Example 2-20 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 2-21.

Example 2-21 TRACE CT command flow

```
  TRACE CT,ON,COMP=SYSTCPRE,SUB=(RESOLV30)
 *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 2-22.

Example 2-22 Saving the trace contents

```
  TRACE CT,ON,COMP=SYSTCPRE,SUB=(RESOLV30)
 *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 2-23.

Example 2-23 Stopping CTRACE

```
TRACE CT,OFF,COMP=SYSTCPRE,SUB=(RESOLV30)
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
```

Display the CTRACE result

The resulting display will show the Resolver process entries, as shown in Example 2-24.

Example 2-24 Resolver formatted trace entries

```
COMPONENT TRACE TALLY REPORT
SYSNAME(SC30)
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
```

2.4.1 For additional information

For more specific information regarding the Resolver address space, refer to *z/OS Communications Server: IP Configuration Guide*, SC31-8775 and *z/OS Communications Server: IP Configuration Reference*, SC31-8776.

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

3

Base functions

The term *base functions* in this case 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 topics.

Section	Topic
3.1, “The base functions” on page 48	Basic concepts of base functions
3.2, “Common design scenarios for base functions” on page 48	Key characteristics of base functions, and why they may be important in your environment
3.2, “Common design scenarios for base functions” on page 48	Commonly implemented base functions design scenarios, their dependencies, advantages, and considerations, as well as our recommendations
3.3, “z/OS UNIX System Services setup for TCP/IP” on page 52	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. Base functions 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 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.

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). The details of the standard applications can be found in *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 2: Standard Applications*, SG24-7533. 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.

Because 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.

3.2 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

Important: Although there are specialized cases where multiple stacks per LPAR can provide value, in general we recommend implementing only one TCP/IP stack per LPAR.

3.2.1 Single stack environment

A single stack environment refers to 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.

Dependencies

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

- Implement a *full-function* UNIX System Services system on z/OS. Detailed information about this topic is available in *z/OS UNIX System Services Planning*, GA22-7800, and in and *z/OS MVS Initialization and Tuning Reference*, SA22-7592. Also refer to *z/OS Program Directory*, GI10-0670-07, which is available at the following address:

  `http://publibz.boulder.ibm.com/epubs/pdf/i1006707.pdf`

- 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. Detailed information is available in *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 4: Security and Policy-Based Networking*, SG24-7535; *z/OS Security Server RACF Security Administrator's Guide*, SA22-7683; *z/OS Security Server RACF System Programmer's Guide*, SA22-7681; and *z/OS Security Server RACF Command Language Reference*, SA22-7687.

- 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. Detailed information is available in *z/OS MVS Initialization and Tuning Reference*, SA22-7592; *z/OS 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 is required to support the interfaces used by TCP/IP.

Advantages

The advantages of a single stack are:

- Fewer CPU cycles are spent processing TCP/IP traffic, because 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.
- Multiple TCP/IP stacks add a significant level of complexity to TCP/IP system administration tasks.

Considerations - single stack environment

When creating a TCP/IP stack, you 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.

3.2.2 Multiple stack environment

A multiple stack environment consists of more than one stack running concurrently in a single LPAR. These stacks 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 advantages for running a multiple stack environment, because it provides you with the flexibility to partition your networking environment. Here are advantages to consider:

- You might want 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 might 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 - multiple stack environment

The considerations for a multiple stack environment are primarily the same as they are for a single stack environment. We therefore indicate here 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 “The Resolver in a multiple stack environment” on page 31, 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 19, for further details.

TSO clients

TSO client functions can be directed against any number of TCP/IP stacks. Keep in mind, though, that the client must be able to find the TCPIP.DATA data set appropriate for the stack of interest. You can 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, however, 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). Therefore, 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, because 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 (for example, UNIX System Services DNS, OSNMP, and ONETSTAT) is related to the internal functioning of the stack.

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

- Ephemeral ports

  In addition to 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(). Use the PORTRANGE statement in the PROFILE.TCPIP to reserve a group of ports, and 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.

3.2.3 Recommendation

In general, we recommend implementing only one TCP/IP stack per LPAR, for the following reasons:

- 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.

3.2.4 Recommendations for MTU

The MTU is largest packet size that can be sent using this route. If the packet is larger than this size, the packet will have to be fragmented if fragmentation is permitted. If fragmentation is not permitted, the packet is dropped and an ICMP error is returned to the originator of the packet. If a route is inactive, the configured MTU value that was defined using the MTU parameter in the ROUTE statement (or the default MTU value for the specified interface type) is displayed. If a route is active, then the actual MTU value is displayed.

For more information about MTU sizes for OSA-Express and HiperSockets, refer to *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 3: High Availability, Scalability, and Performance*, SG24-7534.

3.3 z/OS UNIX System Services setup for TCP/IP

There are several areas that require your attention and action in order to implement a TCP/IP stack successfully. In Chapter 1, “Introduction to z/OS Communications Server for IP” on page 1, we review the UNIX concepts in the z/OS environment. We make specific references to the BPXPRMxx member in SYS1.PARMLIB. However, it is important to first understand the security considerations for the UNIX environment.

3.3.1 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 it 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 Communications Server for z/OS 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 V1R9 TCP/IP Implementation Volume 4: Security and Policy-Based Networking*, SG24-7535.

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 V1R9 TCP/IP Implementation Volume 4: Security and Policy-Based Networking*, SG24-7535.

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 or drop connections or to stop the TN3270 server.

Establish RACF security environment

The notes that follow are merely an overview of the steps in the process. 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 and add an OMVS identity by altering the group to include a GID (ALTGROUP). 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 V1R9 TCP/IP Implementation Volume 4: Security and Policy-Based Networking*, SG24-7535.

3.3.2 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 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. This 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.

3.3.3 Changes to SYS1.PARMLIB members

As we noted, the z/OS environment consists of the traditional MVS and UNIX Systems Services (USS) environment. Because 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 subsystems. 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 include:

- IEASYS00
- BPXPRMxx
- Integrated Sockets PFS definitions

IEASYS00

Because 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 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. *z/OS UNIX System Services Planning*, GA22-7800, and *z/OS MVS Initialization and Tuning Guide*, SA22-7591, explain the details and significance of each parameter in the BPXPRMxx member.

z/OS UNIX System Services Planning, GA22-7800; *z/OS UNIX System Services User's Guide*, SA22-7802; and *z/OS Program Directory*, GI10-0670, detail the structure, design, installation, and implementation of the z/OS UNIX environment.

z/OS Program Directory, GI10-0670, is available at the following address:

`http://publibz.boulder.ibm.com/epubs/pdf/i1006707.pdf`

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 3-1illustrates how support for IPv4 and IPv6 (dual mode) is defined for a single stack environment.

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

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)

/* IPv4 support
 NETWORK DOMAINNAME(AF_INET)      1
         DOMAINNUMBER(2)
         MAXSOCKETS(25000)
         TYPE(INET)               2
         INADDRANYPORT(10000)
         INADDRANYCOUNT(2000)

FILESYSTYPE TYPE(INET)            2
            ENTRYPOINT(EZBPFINI) 3

/* IPv6 support
 NETWORK DOMAINNAME(AF_INET6)     4
        DOMAINNUMBER(19)
        TYPE(INET)
```

INET specifies a single stack with TCPIP (by default) as the stack name.

- 1 AF_INET specifies IPv4 support for the physical file type for socket address used by this stack (TCPIP).
- 2 Specify TYPE(INET) for a single stack environment. If you specify INET, you cannot start multiple TCP/IP stacks.
- 3 EZBPFINI identifies a TCP/IP stack (this is the only valid value).
- 4 AF_INET6 specifies IPv6 support for the physical file type for socket address used by this stack (TCPIP).

Example 3-2 shows BPXPRMxx definitions for a multiple stack environment.

Example 3-2 BPXPRMxx definitions for a multiple stack supporting dual mode

```
FILESYSTYPE TYPE(UDS) ENTRYPOINT(BPXTUINT)
NETWORK DOMAINNAME(AF_UNIX)
        DOMAINNUMBER(1)
        MAXSOCKETS(10000)
        TYPE(UDS)
FILESYSTYPE TYPE(CINET)
        ENTRYPOINT(BPXTCINT)
NETWORK DOMAINNAME(AF_INET)   1
        DOMAINNUMBER(2)
        MAXSOCKETS(10000)
        TYPE(CINET)           2
        INADDRANYPORT(10000)
        INADDRANYCOUNT(2000)

NETWORK DOMAINNAME(AF_INET6)  3
        DOMAINNUMBER(19)
        MAXSOCKETS(10000)
        TYPE(CINET)

SUBFILESYSTYPE NAME(TCPIPA)   4
        TYPE(CINET)           2
        ENTRYPOINT(EZBPFINI) 5
```

```
        DEFAULT

SUBFILESYSTYPE NAME(TCPIPB)   4
        TYPE(CINET)           2
        ENTRYPOINT(EZBPFINI) 5
.....
```

- 1 AF_INET specifies IPv4 support for the physical file type for socket address used by this stack (TCPIP).
- 2 Specify TYPE(CINET) for a single stack environment. If you specify INET, you cannot start multiple TCP/IP stacks.
- 3 AF_INET6 specifies IPv6 support for the physical file type for socket address used by this stack (TCPIP).
- 4 Specify the name of TCP/IP stack you want to configure.
- 5 EZBPFINI identifies a TCP/IP stack (this is the only valid value).

Additional SYS1.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
```

  Before you make this update, make sure that the hlq.SEZALOAD definition has been added to LNKLSTxx and the library itself has been APF authorized. 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 following entries 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. After VMCF and TNF have initialized successfully, you can start your TCP/IP system address spaces.

7. IKJTSOxx

   You 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 ACF/VTAM and CS for z/OS IP usage:

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

10. CTIEZBxx

    Copy CTIEZB00 to SYSx.PARMLIB from hlq.SEZAINST for use with CTRACE.

    This member can be customized to include a different size 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.

    For more information about the use of component tracing (CTRACE), refer to *z/OS CS: IP Diagnosis*, GC31-8782. Also refer to *z/OS CS: IP Migration*, GC31-8773. In this IBM Redbook, see Chapter 8, “Diagnosis” on page 215.

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 3-3.

Example 3-3 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

    Use these to add product and feature information in a z/OS environment.

3.3.4 Changes to SYS1.PROCLIB members

TCP/IP JCL procedures

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).

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.

3.3.5 Additional z/OS customization for UNIX System Services

Updating the MVS system libraries must be done with great care. Follow the instructions in *z/OS 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.

3.3.6 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 V1R9 TCP/IP Implementation Volume 2: Standard Applications*, SG24-7533.

3.3.7 TCP/IP client functions

The client functions of Communications Server for z/OS 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.

3.3.8 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 V1R9 TCP/IP Implementation Volume 2: Standard*

Applications, SG24-7533, and *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 4: Security and Policy-Based Networking*, SG24-7535.

Common errors implementing UNIX System Services

In this section, we discuss 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 zero (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 granting 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 can enter su from the OMVS shell, or you can 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 switch to an OMVS shell the user 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 failed
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 the user's 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(CS RESIDENT     ) 703
  /u/CS01 CL(DIRSRCH) FID(01E2D7D3C5E7F34E2B0F000000000003)
  INSUFFICIENT AUTHORITY TO LOOKUP
  ACCESSINTENT(--X)  ACCESS ALLOWED(OTHER ---)
ICH408I USER(CS01 ) GROUP(WTCRES ) NAME(CS RESIDENT     ) 704
```

Figure 3-3 Error with UNIX permission bit settings

In this case, 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, and so on) 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 the /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 (depending on whether you want the current directory searched first or last):

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

The instructions for setting up this user profile are contained in *z/OS UNIX System Services User's Guide,* SA22-7801, and *z/OS 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; see Figure 3-4.

```
          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.

3.3.9 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 Communications Server for z/OS 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 (which is *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 ACF/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 ACF/VTAM, but also CS for z/OS IP?
    - IVTPRMxx: FIXED MAX(120M)
    - IVTPRMxx: ECSA MAX(120M)
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 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 Communications Server: IP Configuration Reference*, SC31-8776*?*

3.4 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 these has a different interface and requires special knowledge to configure.

z/OS Communications Server IP continues to be enhanced with new features, enhancements, and defaults. So if you are migrating from a previous release, consult with the migration guide for your particular release from which you are migrating. For further details, refer to *z/OS Communications Server: New Function Summary*, GC31-8771.

3.4.1 TCP/IP configuration data set names

This topic is described in *z/OS 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 as to 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 z/OS UNIX functions do.

- 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 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 these 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. The various data sets used by TCP/IP functions and their resolution method are described in *z/OS CS: IP Configuration Guide,* SC31-8775.

3.4.2 PROFILE.TCPIP

Before you start your TCP/IP stack, you must configure the operational and address space characteristics. These definitions are defined in the configuration data set which is often called PROFILE.TCPIP. The PROFILE.TCPIP data set is read by the TCP/IP address space during initialization.

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

- Operating characteristics
- Port number definitions
- Network interface definitions
- Network routing definitions

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

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 141, respectively.

PROFILE.TCPIP statements

In this section we show some essential statements for configuring TCP/IP stack.

The syntax for the parameters in the PROFILE can be found in *z/OS Communications Server: IP Configuration Reference*, SC31-8776. Additional profile statements and descriptions are available in “PROFILE.TCPIP statements” on page 290.

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.

Appendix D, “Our implementation environment” on page 315, describes the environment we used to create this book.

DEVICE and LINK

Use DEVICE and LINK statements to define the physical or virtual interfaces, such as OSA, HiperSockets, and VIPA. z/OS Communications Server can define multiple interfaces. You need to define a pair of DEVICE and LINK statements for each interface you want to configure for a TCP/IP stack.

Each device type has a different set of parameters that you can define. For details on each device type and its definition, refer to Chapter 4, “Connectivity” on page 101.

The following is an example of DEVICE and LINK statements for defining one OSA in QDIO mode.

```
DEVICE OSA20A0    MPCIPA
LINK   OSA20A0L   IPAQENET    OSA20A0
```

The following is an example of DEVICE and LINK statements for defining one VIPA.

```
DEVICE VIPA1      VIRTUAL 0
LINK   VIPA1L     VIRTUAL 0   VIPA1
```

HOME

The HOME statement is used for assigning an IP address for each interface you defined with DEVICE and LINK statements. The following is an example of a HOME statement.

```
HOME
    10.1.1.10     VIPA1L
    10.1.2.12     OSA20A0L
```

The TCP/IP stack uses an IP address of 127.0.0.1 for IPv4 and ::1 for IPv6 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
```

You can display the HOME IP address defined in a particular TCP/IP stack with a D TCPIP,*procname*,Netstat HOME command. The z/OS UNIX shell command **onetstat -h** can also be used for similar. There is an additional field, called the Flag field, that indicates which interface is the primary interface. The primary interface is the first entry in the HOME list in the PROFILE.TCPIP definitions unless the PRIMARYINTERFACE parameter is specified.

The PRIMARYINTERFACE statement can be used to specify which link is to be designated as the default local host address for the GETHOSTID() function.

Example 3-4 Netstat home display

```
D TCPIP,TCPIPA,N,HOME
RESPONSE=SC30
 EZD0101I NETSTAT CS V1R9 TCPIPA
 HOME ADDRESS LIST:
 LINKNAME:   VIPA3L
   ADDRESS:  10.1.30.10
     FLAGS:
 LINKNAME:   VIPA1L
   ADDRESS:  10.1.1.10
     FLAGS:  PRIMARY
 LINKNAME:   VIPA2L
   ADDRESS:  10.1.2.10
     FLAGS:
 LINKNAME:   OSA2080L
   ADDRESS:  10.1.2.11
     FLAGS:
 LINKNAME:   OSA20C0L
   ADDRESS:  10.1.3.11
     FLAGS:
 LINKNAME:   OSA20E0L
   ADDRESS:  10.1.3.12
```

```
    FLAGS:
LINKNAME:   OSA20A0L
  ADDRESS:  10.1.2.12
    FLAGS:
LINKNAME:   IUTIQDF4L
  ADDRESS:  10.1.4.11
    FLAGS:
LINKNAME:   IUTIQDF5L
  ADDRESS:  10.1.5.11
    FLAGS:
LINKNAME:   IUTIQDF6L
  ADDRESS:  10.1.6.11
    FLAGS:
LINKNAME:   EZASAMEMVS
  ADDRESS:  10.1.7.11
    FLAGS:
LINKNAME:   IQDIOLNK0A01070B
  ADDRESS:  10.1.7.11
    FLAGS:
LINKNAME:   VIPL0A01080A
  ADDRESS:  10.1.8.10
    FLAGS:
LINKNAME:   VIPL0A010814
  ADDRESS:  10.1.8.20
    FLAGS:  INTERNAL
LINKNAME:   LOOPBACK
  ADDRESS:  127.0.0.1
    FLAGS:
INTFNAME:   LOOPBACK6
  ADDRESS:  ::1
    TYPE:   LOOPBACK
    FLAGS:
16 OF 16 RECORDS DISPLAYED
END OF THE REPORT
```

BEGINROUTES

Use this statement to define static routes for TCP/IP routing table. This statement is optional when you use OMPROUTE dynamic routing daemon. However, if you do not configure OMPROUTE dynamic routing daemon, BEGINROUTES is necessary for a TCP/IP stack to communicate with other hosts. For details on static and dynamic routing, refer to Chapter 5, “Routing” on page 141.

VIPADYNAMIC

This statement is not always necessary. Use this statement to define dynamic VIPA or the functions related to dynamic VIPA, such as sysplex distributor and dynamic VIPA takeover. Refer to *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 3: High Availability, Scalability, and Performance*, SG24-7534, for details about high availability and load balancing functions using dynamic VIPA.

AUTOLOG

The procedures specified in AUTOLOG statement are initialized at TCP/IP startup, so you do not have to start the TCP/IP applications manually after the TCP/IP startup. 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 * NOAUTOLOG ;OMVS
    21   TCP  FTPDA1 ;Contol Port

AUTOLOG 1
    FTPDA JOBNAME FTPDA1 ; FTP Server
ENDAUTOLOG
```

START

Specify a device name on a START statement to initialize the interface at the TCP/IP stack startup. The following is an example of START statement for an OSA and a HiperSockets device. VIPA does not need to be started because it is virtual and always active.

If you do not specify a device name on a START statement, you can initialize the device with the TCPIP,*procname*,START,*devicename* command after the TCP/IP stack startup.

```
START OSA20A0
START IUTIQDF4
```

IPCONFIG

IPv4 features are defined within IPCONFIG. There is a separate configuration section for IPv6 parameters. Refer to “PROFILE.TCPIP statements” on page 290 for commonly used IPCONFIG statements.

TCPCONFIG

TCP features are defined within TCPCONFIG. Refer to “PROFILE.TCPIP statements” on page 290 for commonly used TCPCONFIG statements.

UDPCONFIG

UDP features are defined within UDPCONFIG. Refer to “PROFILE.TCPIP statements” on page 290 for commonly used UDPCONFIG statements.

GLOBALCONFIG

GLOBALCONFIG statement defines the parameters that are affective to the entire TCP/IP stack. Refer to “PROFILE.TCPIP statements” on page 290 for commonly used GLOBALCONFIG statements.

INTERFACE

The HOME statement in combination with the DEVICE and LINK statements is 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 *z/OS Communications Server: IP Configuration Guide,* SC31-8775, and *z/OS Communications Server: IP Configuration Reference,* SC31-8776, for further details.

IPCONFIG6

All IPv6 features are defined within IPCONFIG6.

Locating PROFILE.TCPIP

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

1. //PROFILE DD
   ```
   //PROFILE DD DSN=TCPIPA.TCPPARMS(PROFA30)
   ```
2. jobname.nodename.TCPIP
3. hlq.nodename.TCPIP
4. jobname.PROFILE.TCPIP
5. hlq.PROFILE.TCPIP

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

3.4.3 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 3-5.

Example 3-5 TRLE VTAM major node definition for device OSA2080

```
OSA2080  VBUILD TYPE=TRL
OSA2080P TRLE  LNCTL=MPC,                                      *
               READ=2080,                                      *
               WRITE=2081,                                     *
               DATAPATH=(2082-2088),                           *
               PORTNAME=OSA2080,                               *
               MPCLEVEL=QDIO
```

Because 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 3-5) must also be the same as the device name defined in PROFILE.TCPIP data set on the DEVICE and LINK statements.

3.4.4 TCPIP.DATA

The Resolver configuration file is often called 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.

The TCPIP.DATA configuration data set is read during initialization of *all* TCP/IP server and client functions. TCPIP.DATA contains the configuration for the Resolver address space. We define the way name-to-address or address-to-name resolution is performed by the Resolver.

TCPIP.DATA is also used by the TCP/IP applications to specify the TCP/IP stack it establishes an affinity with. The associated TCP/IP stack name is specified with TCPIPJOBNAME statement. Other stack-specific statements are HOSTNAME, which is the host name of the TCP/IP stack, and DATASETPREFIX, which is the data set prefix (hlq) to be used for searching a configuration data set.

The syntax for the parameters in the TCPIP.DATA file can be found in *z/OS Communications Server: IP Configuration Guide*, SC31-8775. A sample TCPIP.DATA configuration file is provided in *hlq*.SEZAINST(TCPDATA). You can define the TCPIP.DATA parameters in an MVS data set or z/OS UNIX file system file.

For further information about TCPIP.DATA file and the Resolver address space, refer to Chapter 2, “The Resolver” on page 19.

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 3-6 TCPIP.DATA with HOMETEST

```
EZAO619I Running IBM MVS TCP/IP CS V1R9 TCP/IP Configuration Tester

Resolver Trace Initialization Complete -> 2007/10/03 14:07:33.438588
res_init Resolver values:
  Global Tcp/Ip Dataset  = SYS1.TCPPARMS(GLBLDATA)
  Default Tcp/Ip Dataset = TCPIPA.TCPPARMS(DEFAULT)
  Local Tcp/Ip Dataset   = //DD:SYSTCPD
                         ==> TCPIPA.TCPPARMS(DATAA30)
  Translation Table      = TCPIPA.STANDARD.TCPXLBIN
  UserId/JobName         = CS03
  Caller API             = TCP/IP Pascal Sockets
  Caller Mode            = EBCDIC
  (L) DataSetPrefix = TCPIPA
  (L) HostName      = WTSC30A
  (L) TcpIpJobName  = TCPIPA
  (G) Search        = ITSO.IBM.COM
                      IBM.COM
  (G) NameServer    = 9.12.6.7
  (G) NsPortAddr    = 53           (G) ResolverTimeout    = 10
  (G) ResolveVia    = UDP          (G) ResolverUdpRetries = 1
  (L) Options NDots = 1
  (L) Trace Resolver               (*) SockNoTestStor
  (*) AlwaysWto     = NO           (L) MessageCase        = MIXED
  (G) LookUp        = LOCAL DNS
 res_init Succeeded
res_init Started: 2007/10/03 14:07:33.469946
res_init Ended: 2007/10/03 14:07:33.469952
GetHostByName Started: 2007/10/03 14:08:14.952365
GetHostByName Resolving Name:  WTSC30A
GetHostByName Trying Local Tables
Global IpNodes Dataset  = TCPIPA.TCPPARMS(IPNODES)
Default IpNodes Dataset = None
Search order            = CommonSearch
 SITETABLE from globalipnodes TCPIPA.TCPPARMS(IPNODES)
 - Lookup for WTSC30A.ITSO.IBM.COM
 - Lookup for WTSC30A.IBM.COM
 - Lookup for WTSC30A
```

```
 ADDRTABLE from globalipnodes TCPIPA.TCPPARMS(IPNODES)
 - Lookup for 10.1.1.10
GetHostByName Succeeded:  IP Address(es) found:
  IP Address(1) is 10.1.1.10
GetHostByName Ended: 2007/10/03 14:08:14.986069
************************************************************************
EZA0611I The following IP addresses correspond to TCP Host Name: WTSC30A
EZA0612I 10.1.1.10

EZA0614I The following IP addresses are the HOME IP addresses defined in PROFIL

.TCPIP:
EZA0615I 10.1.30.10
EZA0615I 10.1.1.10
EZA0615I 10.1.2.10
EZA0615I 10.1.2.11
EZA0615I 10.1.3.11
EZA0615I 10.1.3.12
EZA0615I 10.1.2.12
EZA0615I 10.1.4.11
EZA0615I 10.1.5.11
EZA0615I 10.1.6.11
EZA0615I 10.1.7.11
EZA0615I 10.1.7.11
EZA0615I 10.1.8.10
EZA0615I 10.1.8.20
EZA0615I 127.0.0.1

EZA0618I All IP addresses for WTSC30A are in the HOME list!

EZA0622I Hometest was successful - all Tests Passed!
```

3.4.5 Configuring the local hosts file

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 or NAMESERVER statement in the TCPIP.DATA configuration file). A domain name server can be used in conjunction with the local hosts file. 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.

Refer to Chapter 2, “The Resolver” on page 19 for further explanation and details.

3.5 Implementing the TCP/IP stack

In this scenario we create a TCP/IP stack by the name of TCPIPA on the SC30 system (LPAR A23). We define four OSAs and three HiperSockets interfaces and static routing. Figure 3-5 illustrates the OSA2080 and OSA20A0 pair connecting to the same VLAN via two different OSA-Express features. The same applies to the OSA20C0 and OSA20E0 pair. We also defined a dynamic XCF connection, which in our environment can use either a Coupling Facility link or HiperSockets (CHPID F7).

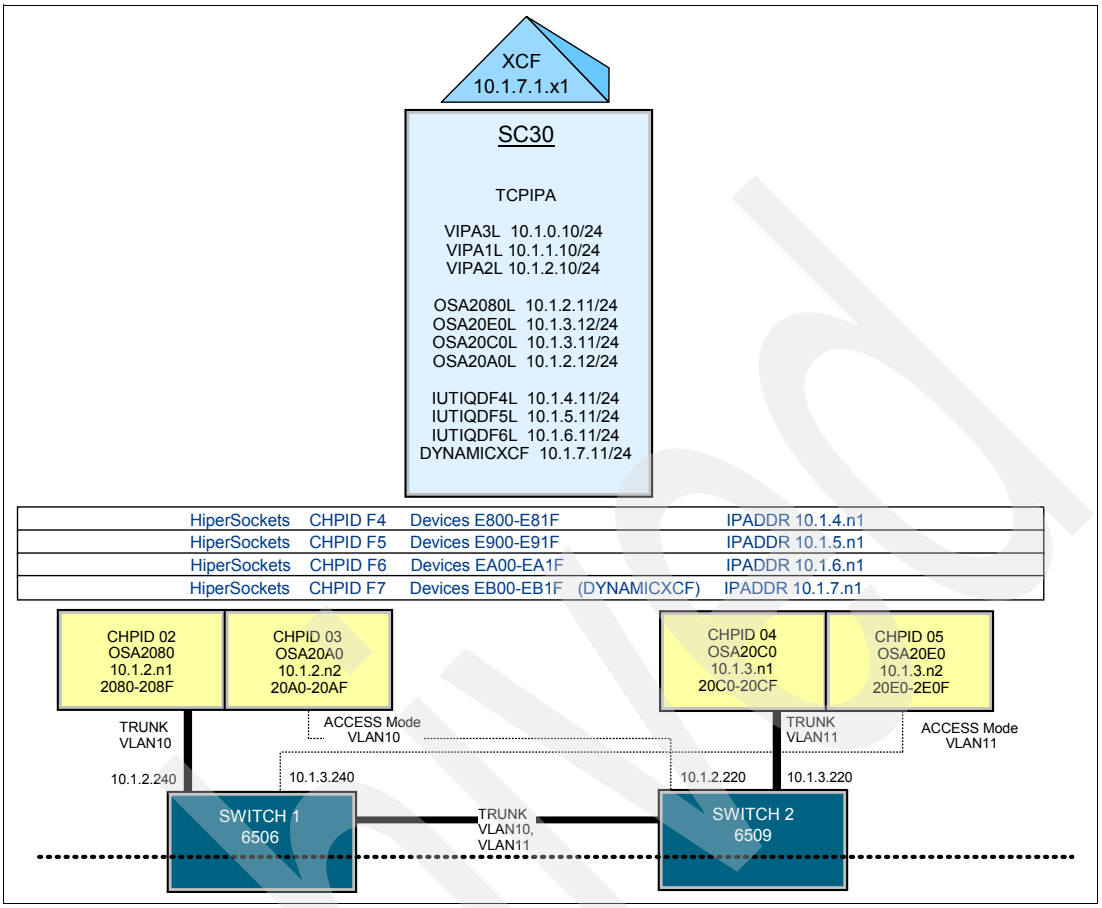


Figure 3-5 Network diagram

3.5.1 Implementation tasks

Perform following steps to implement the TCP/IP stack to support base functions:

1. Create a TCPIP.DATA file.
2. Create a PROFILE.TCPIP file.
3. Check BPXPRMxx.
4. Create a TCP/IP cataloged procedure.
5. RACF definitions.
6. Create a VTAM TRL major node for MP CIPA OSA.

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 TCPIPA.TCPPARMS(DATAA30).

Create TCPIP.DATA file

We defined a global TCPIP.DATA and a local TCPIP.DATA for TCPIPA, as shown in Example 3-7 and Example 3-8.

Example 3-7 Global TCPIP.DATA file

```
; ****************************************
; TCPIPA.TCPPARMS(GLOBAL)
; ****************************************
DOMAINORIGIN  ITSO.IBM.COM     1
```

```
NSINTERADDR  10.12.6.7          2
NSPORTADDR 53
RESOLVEVIA UDP
RESOLVERTIMEOUT 10
RESOLVERUDPRETRIES 1
LOOKUP LOCAL DNS                3
```

We created a local TCPIP.DATA file for the TCPIPA stack.

Example 3-8 local TCPIP.DATA file

```
; ****************************************
; TCPIPA.TCPPARMS(DATAA30)
; ****************************************
TCPIPJOBNAME TCPIPA                1
HOSTNAME WTSC30A                   2
DATASETPREFIX TCPIPA
MESSAGECASE MIXED
```

- 1 Specifies the procedure name of TCPIPA stack.
- 2 Specifies the host name of the TCPIPA stack.

Update the Domain Name Server

If you are using a domain name server, ensure that it is updated with your new host name and address.

Update the local hosts file

If you are not using a domain name server, edit your global ETC.IPNODES file or the local ETC.IPNODES file and add your new host name and address.

Create the PROFILE.TCPIP file

We created a TCP/IP profile and included the following statements.

DEVICE and LINK statement

We configured two OSA-Express features, each card having two ports. We configured four interfaces (ports) with DEVICE and LINK statements. 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.

HOME statement

We assigned a IP address for each interface that was configured with a DEVICE and LINK statement pair.

BEGINROUTES statement

We defined static routes with BEGINROUTES statement to route a traffic to other hosts on a network using the OSA-Express or HiperSockets interfaces.

PORT statement

We reserved TCP ports for some applications with PORT statement

START statement

We defined a START statement to initialize the interfaces at the TCP/IP stack startup.

DYNAMICXCF statement

We defined a DYNAMICXCF statement to dynamically define the device to join the sysplex.

Example 3-9 shows a sample PROFILE.TCPIP file.

Example 3-9 PROFILE.TCPIP file

```
; ****************************************
; TCPIPA.TCPPARMS(PROFA3OS)
; ****************************************
ARPAGE 20
;
GLOBALCONFIG NOTCPIPSTATISTICS
;
IPCONFIG DATAGRAMFWD SYSPLEXROUTING
;
DYNAMICXCF 10.1.7.11 255.255.255.0 1
;
SOMAXCONN 10
;
TCPCONFIG TCPSENDBFRSIZE 64K TCPRCVBUFRSIZE 64K TCPMAXRCVBUFRSIZE 256K
TCPCONFIG SENDGARBAGE FALSE UNRESTRICTLOWPORTS
;
UDPCONFIG UNRESTRICTLOWPORTS
;
;OSA DEFINITIONS
DEVICE OSA2080    MPCIPA
LINK   OSA2080L   IPAQENET    OSA2080 VLANID 10
DEVICE OSA20C0    MPCIPA
LINK   OSA20C0L   IPAQENET    OSA20C0 VLANID 11
DEVICE OSA20E0    MPCIPA
LINK   OSA20E0L   IPAQENET    OSA20E0
DEVICE OSA20A0    MPCIPA
LINK   OSA20A0L   IPAQENET    OSA20A0
;
;HIPERSOCKETS DEFINITIONS
DEVICE IUTIQDF4   MPCIPA
LINK   IUTIQDF4L  IPAQIDIO    IUTIQDF4
DEVICE IUTIQDF5   MPCIPA
LINK   IUTIQDF5L  IPAQIDIO    IUTIQDF5
DEVICE IUTIQDF6   MPCIPA
LINK   IUTIQDF6L  IPAQIDIO    IUTIQDF6
;
;STATIC VIPA DEFINITIONS
DEVICE VIPA1      VIRTUAL 0
LINK   VIPA1L     VIRTUAL 0   VIPA1
DEVICE VIPA2      VIRTUAL 0
LINK   VIPA2L     VIRTUAL 0   VIPA2
DEVICE VIPA3      VIRTUAL 0
LINK   VIPA3L     VIRTUAL 0   VIPA3
;
HOME
   10.1.&SYSCLONE..10  VIPA3L
   10.1.1.10     VIPA1L
   10.1.2.10     VIPA2L
   10.1.2.11     OSA2080L
   10.1.3.11     OSA20C0L
   10.1.3.12     OSA20E0L
   10.1.2.12     OSA20A0L
   10.1.4.11     IUTIQDF4L
```

```
   10.1.5.11     IUTIQDF5L
   10.1.6.11     IUTIQDF6L
BEGINRoutes
; Direct Routes - Routes that are directly connected to my interfaces
;     Destination     Subnet Mask   First Hop Link Name     Packet Size
 ROUTE 10.1.2.0/24                    =        OSA2080L      MTU 1492
 ROUTE 10.1.2.0/24                    =        OSA20A0L      MTU 1492
 ROUTE 10.1.3.0/24                    =        OSA20C0L      MTU 1492
 ROUTE 10.1.3.0/24                    =        OSA20E0L      MTU 1492
 ROUTE 10.1.4.0/24                    =        IUTIQDF4L     MTU 8192
 ROUTE 10.1.5.0/24                    =        IUTIQDF5L     MTU 8192
 ROUTE 10.1.6.0/24                    =        IUTIQDF6L     MTU 8192
; Indirect Routes - Routes that are indirectly connected to my interfaces
;     Destination    Subnet Mask   First Hop  Link Name     Packet Size ;
 ROUTE 10.1.1.20/32                 10.1.4.21  IUTIQDF4L     MTU 8192
 ROUTE 10.1.2.20/32                 10.1.4.21  IUTIQDF4L     MTU 8192
 ROUTE 10.1.31.10/32                10.1.4.21  IUTIQDF4L     MTU 8192
 ROUTE 10.1.100.0/24                10.1.2.240 OSA2080L      MTU 1492
 ROUTE 10.1.100.0/24                10.1.2.240 OSA20A0L      MTU 1492
 ROUTE 10.1.100.0/24                10.1.3.240 OSA20C0L      MTU 1492
 ROUTE 10.1.100.0/24                10.1.3.240 OSA20E0L      MTU 1492

; Default Routes
; Destination                        First Hop    Link Name   Packet Size
 ROUTE DEFAULT                       10.1.2.240   OSA2080L    MTU 1492
 ROUTE DEFAULT                       10.1.2.220   OSA2080L    MTU 1492
 ROUTE DEFAULT                       10.1.2.240   OSA20A0L    MTU 1492
 ROUTE DEFAULT                       10.1.2.220   OSA20A0L    MTU 1492
 ROUTE DEFAULT                       10.1.3.240   OSA20C0L    MTU 1492
 ROUTE DEFAULT                       10.1.3.220   OSA20C0L    MTU 1492
 ROUTE DEFAULT                       10.1.3.240   OSA20E0L    MTU 1492
 ROUTE DEFAULT                       10.1.3.220   OSA20E0L    MTU 1492
 ENDRoutes
;
 PORT
    20 TCP OMVS      NOAUTOLOG ; FTP Server  1
    23 TCP TN3270A             ; Telnet Server
    514 UDP OMVS               ; UNIX SyslogD Server   3
    21 TCP FTPDA1  BIND 10.1.1.10  ; control port
    25 TCP SMTP                ; SMTP Server
;
START OSA2080
START OSA20C0
START OSA20E0
START OSA20A0
START IUTIQDF4
START IUTIQDF5
START IUTIQDF6
```

Check BPXPRMxx

Refer to SYS1.PARMLIB(BPXPRMxx) and make sure you have your TCP/IP stack name defined in it. If you do not have the stack name in BPXPRMxx, refer to 3.3.3, “Changes to SYS1.PARMLIB members” on page 56.

Create TCP/IP cataloged procedure

We created a cataloged procedure for TCPIPA stack, as shown in Example 3-10.

Example 3-10 Address space JCL procedure (SC30)

```
//TCPIPA    PROC PARMS='CTRACE(CTIEZB00),IDS=00',
//            PROFILE=PROFA&SYSCLONE,TCPDATA=DATAA&SYSCLONE          1
//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.)
//SYSTCPD  DD DSN=TCPIPA.TCPPARMS(&TCPDATA.),DISP=SHR 2
```

- 1 Illustrates the use of SYSTEM SYMBOLS.
- 2 SYSTCPD DD statement pointing to TCPIPA.TCPPARMS(DATAA30).

RACF definitions

Add RACF definitions to assign started task user IDs to new address spaces. This will be done by the RACF administrator. Example 3-11 shows the RACF command we used.

Example 3-11 Defining TCPIP.* procedure to started task*

```
ADDGROUP TCPGRP OMVS(UID(100))
ADDUSER TCPIP DFLTGRP(TCPGRP) OMVS(UID(0) HOME(/) PROGRAM(/bin/sh)) NOPASSWORD
SETROPTS GENERIC(STARTED)
SETROPTS CLASSACT(STARTED) RACLIST(STARTED)
DEFINE STARTED TCPIP*.* STDATA(USER(TCPIP) GROUP(TCPGRP))
SETROPTS RACLIST(STARTED) REFRESH
```

Create a VTAM TRL major node for MPCIPA OSA

We defined our TRLEs in VTAM. Remember to include it in the VTAM startup list in ATCCONxx. Example 3-12 and Example 3-13 on page 78 are sample TRL major nodes for one OSA device; we then created TRLEs for all OSA devices.

Example 3-12 displays the TRLE VTAM major node definition for device OSA2080.

Example 3-12 TRLE VTAM major node definition for device OSA2080

```
OSA2080  VBUILD TYPE=TRL
OSA2080P TRLE  LNCTL=MPC,                                          *
               READ=2080,                                          *
               WRITE=2081,                                         *
               DATAPATH=(2082-2088),                               *
               PORTNAME=OSA2080,                                   *
               MPCLEVEL=QDIO
```

Example 3-13 on page 78 displays the TRLE VTAM major node definition for device OSA20A0.

Example 3-13 TRLE VTAM major node definition for device OSA20A0

```
OSA20A0  VBUILD TYPE=TRL
OSA20A0P TRLE  LNCTL=MPC,
               READ=20A0,
               WRITE=20A1,
               DATAPATH=(20A2-20A7),
```

```
                PORTNAME=OSA20A0,
                MPCLEVEL=QDIO
```

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.

3.6 Activating the TCP/IP stack

If you IPL your z/OS system with PARMLIB definitions similar to our environment, you should get messages similar to those shown in Example 3-14. 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*.

Example 3-14 IPL and start TCPIPA

```
/* IPL and start of TCPIPA
IEE252I MEMBER BPXPRM9A FOUND IN SYS1.PARMLIB 1
CEE3739I LANGUAGE ENVIRONMENT INITIALIZATION COMPLETE 2
IEE252I MEMBER BPXPRM9A FOUND IN SYS1.PARMLIB
IEE252I MEMBER CTIEZB00 FOUND IN SYS1.IBM.PARMLIB
IEE252I MEMBER CTIIDS00 FOUND IN SYS1.IBM.PARMLIB
IEE252I MEMBER CTINTA00 FOUND IN SYS1.PARMLIB
/*
EZZ4202I Z/OS UNIX - TCP/IP CONNECTION ESTABLISHED FOR TCPIPA
BPXF206I ROUTING INFORMATION FOR TRANSPORT DRIVER TCPIPA HAS BEEN 3
INITIALIZED OR UPDATED.
/*
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE OSA2080
IEF196I IEF237I EA02 ALLOCATED TO TPEA02
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE OSA20C0
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE OSA20E0
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE IUTIQDF5
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE IUTIQDF6
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE IUTIQDF4
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE OSA20A0
EZB6473I TCP/IP STACK FUNCTIONS INITIALIZATION COMPLETE. 5
EZAIN11I ALL TCPIP SERVICES FOR PROC TCPIPA ARE AVAILABLE.
/*
EZBH006E GLOBALCONFIG SYSPLEXMONITOR RECOVERY was not specified when
IPCONFIG DYNAMICXCF or IPCONFIG6 DYNAMICXCF was configured.
/*
EZD1176I TCPIPA HAS SUCCESSFULLY JOINED THE TCP/IP SYSPLEX GROUP 4
EZBTCPCS
EZD1214I INITIAL DYNAMIC VIPA PROCESSING HAS COMPLETED FOR TCPIPA
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE IUTIQDIO
/*
```

- 1 The first important message indicates whether the right UNIX customization data set is used. In our environment it is BPXPRM9A. This contains the *root* system upon which all other file systems are mounted, and it 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 because the zFSs are SMS-managed. Note that the file systems are mounted subsequently starting with the root.

- 2 This 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 by 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.

- 3 The following two messages indicate the successful initialization of the UNIX Systems Services environment and TCP/IP services according to the BPXPRMxx definitions.
- 4 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.

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

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 that the system is running a functional SMS environment, as shown in Example 3-15.

Example 3-15 Output of D SMS command

```
IGD002I 15:49:30 DISPLAY SMS 638
SCDS = SYS1.SMS.SCDS
ACDS = SYS1.SMS.ACDS
COMMDS = SYS1.SMS.COMMDS
DINTERVAL = 150
REVERIFY = NO
ACSDEFAULTS = NO
    SYSTEM     CONFIGURATION LEVEL    INTERVAL SECONDS
    SC30       2007/10/03 15:49:20           10
    SC31       2007/10/03 15:49:14           10
    SC32       2007/10/03 15:49:10           10
    SC33       2007/10/03 15:49:11           10
```

You see in the output from the SMS display that we are supporting four different LPARs and the captured information pertains to SC30, SC31, SC32 and SC33.

Example 3-16 Displaying the OMVS system that is running

```
D OMVS,ASID=ALL
BPXO040I 15.56.17 DISPLAY OMVS 648
OMVS     000F ACTIVE             OMVS=(9A) 1
USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
OMVSKERN BPXOINIT 003F          1          0 MKI--- 10.41.26    44.144 2
  LATCHWAITPID=         0 CMD=BPXPINPR
  SERVER=Init Process                     AF=    0 MF=00000 TYPE=FILE
TCPIP    NFSCLNT  0023      65540          1 MF---- 10.41.28    38.569
```

```
  LATCHWAITPID=         0 CMD=GFSCINIT
TCPIP    TCPIP    0078      65543          1 1F---B 14.18.55  2160.583
  LATCHWAITPID=         0 CMD=EZACFALG
TCPIP    TCPIP    0078      65548          1 1F---B 14.18.55  2160.583 3
  LATCHWAITPID=         0 CMD=EZASASUB
OMVSKERN INETD1   0040      65549          1 1FI--- 10.41.40      .424
  LATCHWAITPID=         0 CMD=/usr/sbin/inetd /etc/inetd.conf
TCPIP    TN3270   0048   16842768          1 1R---- 10.41.44   986.517
  LATCHWAITPID=         0 CMD=EZBTTSSL
TCPIP    TN3270   0048      65553          1 1R---- 10.41.44   986.517
  LATCHWAITPID=         0 CMD=EZBTZMST
TCPIP    TN3270   0048      65555          1 1R---- 10.41.44   986.517
  LATCHWAITPID=         0 CMD=EZBTMCTL
TCPIP    TN3270   0048      65557          1 1R---- 10.41.44   986.517
  LATCHWAITPID=         0 CMD=EZBTTMST
TCPIP    TN3270   0048      65558          1 1R---- 10.41.44   986.517
  LATCHWAITPID=         0 CMD=EZBTTMST
TCPIP    TN3270   0048      65559          1 MR---- 10.41.44   986.517
  LATCHWAITPID=         0 CMD=EZBTTMST
TCPIP    PORTMAP  007B   33619998          1 1FI--- 14.19.06      .012
  LATCHWAITPID=         0 CMD=PORTMAP
TCPIP    FTPOE1   007E   16842784          1 1FI--- 14.19.06      .022
  LATCHWAITPID=         0 CMD=FTPD
TCPIP    FTPMVS1  007D      65570          1 1FI--- 14.19.06      .049
  LATCHWAITPID=         0 CMD=FTPD
TCPIP    REXECD   007C      65571          1 1FI--- 14.19.06      .012
  LATCHWAITPID=         0 CMD=RSHD
IBMUSER  IOASRV   007A   67174436          1 1FI--- 14.30.33      .041
  LATCHWAITPID=         0 CMD=IOAXTSRV
TCPIP    FTPDA1   00AD      65582          1 1FI--- 15.22.38      .016
  LATCHWAITPID=         0 CMD=FTPD
TCPIP    TN3270A  00AE   33620045          1 1R---- 15.14.48    31.758
  LATCHWAITPID=         0 CMD=EZBTZMST
TCPIP    TN3270A  00AE   33620047          1 1F---- 15.14.48    31.758
  LATCHWAITPID=         0 CMD=EZBTSSUB
TCPIP    TN3270A  00AE   16842833          1 1R---- 15.14.48    31.758
  LATCHWAITPID=         0 CMD=EZBTTSSL
PAGENT   PAGENT   0094   50397293          1 HF---- 14.01.09    30.947
  LATCHWAITPID=         0 CMD=PAGENT
TCPIP    TN3270A  00AE      65679          1 1R---- 15.14.48    31.758
  LATCHWAITPID=         0 CMD=EZBTTMST
TCPIP    TN3270A  00AE      65682          1 1R---- 15.14.48    31.758
  LATCHWAITPID=         0 CMD=EZBTTMST
TCPIP    TN3270A  00AE      65694          1 1R---- 15.14.48    31.758
  LATCHWAITPID=         0 CMD=EZBTTMST
TCPIP    TCPIPA   00B5   33620199          1 MF---B 15.22.26     1.804
  LATCHWAITPID=         0 CMD=EZBTCPIP
TCPIP    TCPIPA   00B5   16843025          1 1F---B 15.22.28     1.804
  LATCHWAITPID=         0 CMD=EZACFALG
TCPIP    TCPIPA   00B5   16843032          1 1F---B 15.22.28     1.804
  LATCHWAITPID=         0 CMD=EZASASUB
TCPIP    TN3270A  00AE   16843047          1 1R---- 15.14.48    31.758
  LATCHWAITPID=         0 CMD=EZBTMCTL
TCPIP    TCPIP    0078   50397484          1 MF---B 14.18.53  2160.583
  LATCHWAITPID=         0 CMD=EZBTCPIP
```

In Example 3-16 on page 80:

- The OMVS member that is running is related to **1** BPXPRM97A.

- The initialization process is running as superuser 2 OMVSKERN, and the PID is 1 (the first process to start).
- 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).

Also notice the different TPC/IP stacks and tasks associated with them. There is TCPIPA and TCPIP (the default stack), both executing EZBTCPIP. There are also multiple tasks associated with the same RACF user ID, TCPIP. This offers the advantage of easier maintenance and system definitions, However, this also presents the disadvantage of having no distinguishing features among messages for individual tasks. Many users of TCP/IP and UNIX System Services would assign individual RACF user IDs to each OMVS user for easier problem determination.

For a thorough discussion on the use and implementation of RACF, refer to *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 4: Security and Policy-Based Networking*, SG24-7535.

Example 3-17 shows the display of available file systems after the initialization of the z/OS UNIX Systems Services environment. The display should list all of the files defined in the mount statement in the BPXPRMxx member, which in our scenario is BPXPRM9A.

Example 3-17 Output of D OMVS,F

```
D OMVS,F
OMVS     000F ACTIVE             OMVS=(9A)
TYPENAME   DEVICE ----------STATUS----------- MODE  MOUNTED    LATCHES
ZFS         90107 ACTIVE                      RDWR  09/25/2007  L=65
  NAME=RC30.ZFS                                     14.59.24    Q=0
  PATH=/u/rc30
  AGGREGATE NAME=RC30.ZFS
  OWNER=SC30     AUTOMOVE=Y CLIENT=N
ZFS         68861 ACTIVE                      RDWR  09/19/2007  L=63
  NAME=PACLIENT.HFS                                 11.29.50    Q=0
  PATH=/u/paclient
  AGGREGATE NAME=PACLIENT.HFS
  OWNER=SC30     AUTOMOVE=Y CLIENT=N
ZFS             3 ACTIVE                      RDWR  08/31/2007  L=15
  NAME=OMVS.ZOSR19.Z19RB1.ROOT                      10.41.25    Q=15
  PATH=/Z19RB1
  AGGREGATE NAME=OMVS.ZOSR19.Z19RB1.ROOT
  OWNER=SC30     AUTOMOVE=Y CLIENT=N
AUTOMNT        18 ACTIVE                      RDWR  08/31/2007  L=24
  NAME=*AMD/u                                       10.41.25    Q=24
  PATH=/u
  OWNER=SC30     AUTOMOVE=Y CLIENT=N
TFS           7492 ACTIVE                     RDWR  08/31/2007  L=40
  NAME=/DEV                                         10.41.26    Q=0
  PATH=/SC30/dev
  MOUNT PARM=-s 10
  OWNER=SC30     AUTOMOVE=U CLIENT=N
TFS           7491 ACTIVE                     RDWR  08/31/2007  L=39
  NAME=/SC30/TMP                                    10.41.26    Q=0
  PATH=/SC30/tmp
  MOUNT PARM=-s 500
  OWNER=SC30     AUTOMOVE=U CLIENT=N
```

```
TFS         52004 ACTIVE                      RDWR  09/14/2007  L=33
  NAME=/SC31/TMP                                    14.34.19    Q=33
  PATH=/SC31/tmp
  MOUNT PARM=-s 500
  OWNER=SC31     AUTOMOVE=U CLIENT=Y
HFS         52001 ACTIVE                      RDWR  09/14/2007  L=52
  NAME=WTSCPLX5.SC31.SYSTEM.ROOT                    14.34.19    Q=52
  PATH=/SC31
  OWNER=SC31     AUTOMOVE=U CLIENT=Y
HFS         52002 ACTIVE                      RDWR  09/14/2007  L=50
  NAME=OMVS.SC31.ETC                                14.34.19    Q=50
  PATH=/SC31/etc
  OWNER=SC31     AUTOMOVE=U CLIENT=Y
HFS          7490 ACTIVE                      RDWR  08/31/2007  L=38
  NAME=OMVS.SC30.VAR                                10.41.26    Q=0
  PATH=/SC30/var
  OWNER=SC30     AUTOMOVE=U CLIENT=N
HFS          7489 ACTIVE                      RDWR  08/31/2007  L=37
  NAME=OMVS.SC30.ETC                                10.41.26    Q=0
  PATH=/SC30/etc
  OWNER=SC30     AUTOMOVE=U CLIENT=N
HFS          7488 ACTIVE                      RDWR  08/31/2007  L=36
  NAME=WTSCPLX5.SC30.SYSTEM.ROOT                    10.41.25    Q=0
  PATH=/SC30
  OWNER=SC30     AUTOMOVE=U CLIENT=N
HFS         52003 ACTIVE                      RDWR  09/14/2007  L=34
  NAME=OMVS.SC31.VAR                                14.34.19    Q=34
  PATH=/SC31/var
  OWNER=SC31     AUTOMOVE=U CLIENT=Y
HFS            15 ACTIVE                      RDWR  08/31/2007  L=23
  NAME=OMVS.PP.HFS                                  10.41.25    Q=23
  PATH=/pp
  OWNER=SC30     AUTOMOVE=Y CLIENT=N
HFS             1 ACTIVE                      RDWR  08/31/2007  L=14
  NAME=WTSCPLX5.SYSPLEX.ROOT                        10.41.25    Q=14
  PATH=/
  OWNER=SC30     AUTOMOVE=Y CLIENT=N
HFS         18552 ACTIVE                      RDWR  09/04/2007  L=53
  NAME=CS03.HFS                                     16.47.37    Q=0
  PATH=/u/cs03
  OWNER=SC30     AUTOMOVE=Y CLIENT=N
.....
```

Example 3-18 shows some of the files defined in the active BPXPRM9A member for comparative purposes only. We can compare the names defined in the active BPXPRM9A member with the names that are actually active by using the **D OMVS,F** command.

Example 3-18 BPXPRM9A member

```
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.ZOSR19.&SYSR1..ROOT')
```

```
      MOUNTPOINT('/$VERSION')
      AUTOMOVE
      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

MOUNT FILESYSTEM('/DEV')
      MOUNTPOINT('/dev')
      TYPE(TFS)
      PARM('-s 10')
      UNMOUNT
```

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

Refer to *z/OS UNIX System Services Planning*, GA22-7800-08, and *z/OS UNIX System Services User's Guide*, SA22-7802-07, for detailed information about UNIX commands in the z/OS UNIX environment.

Example 3-19 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  -   Aug 31 ?        0:44 BPXPINPR
 BPXROOT  16842754          1  -   Aug 31 ?        0:17 IAZNJTCP
     NET     65539          1  -   Aug 31 ?       25:34 ISTMGCEH
 BPXROOT     65540          1  -   Aug 31 ?        0:38 GFSCINIT
 BPXROOT  16842757          1  -   Aug 31 ?        0:00 CEAPSRVR
 BPXROOT     65543          1  -   Sep 05 ?       36:05 EZACFALG
 BPXROOT     65544          1  -   Aug 31 ?        6h30 ERB3GMFC
 BPXROOT     65548          1  -   Sep 05 ?       36:05 EZASASUB
 BPXROOT     65549          1  -   Aug 31 ?        0:00 /usr/sbin/inetd /etc/i
netd.conf
 BPXROOT  16842768          1  -   Aug 31 ?       16:28 EZBTTSSL
 BPXROOT     65553          1  -   Aug 31 ?       16:28 EZBTZMST
BPXROOT     65555          1  -   Aug 31 ?       16:28 EZBTMCTL
 BPXROOT     65557          1  -   Aug 31 ?       16:28 EZBTTMST
 BPXROOT     65558          1  -   Aug 31 ?       16:28 EZBTTMST
 BPXROOT     65559          1  -   Aug 31 ?       16:28 EZBTTMST
 BPXROOT  16842777          1  -   Sep 21 ?        0:47 /usr/sbin/syslogd -f /
etc/syslog.conf
 BPXROOT  33619998          1  -   Sep 05 ?        0:00 PORTMAP
 BPXROOT     65571          1  -   Sep 05 ?        0:00 RSHD
 BPXROOT  67174436          1  -   Sep 05 ?        0:00 IOAXTSRV
    CS03  67174470          1  - 16:38:47 ?        0:39 OMVS
```

```
BPXROOT      65612   67174594  - 16:39:27 ttyp0000  0:00 ps -ef
BPXROOT   33620045          1  -   Oct 02 ?         0:32 EZBTZMST
BPXROOT   33620047          1  -   Oct 02 ?         0:32 EZBTSSUB
BPXROOT   16842833          1  -   Oct 02 ?         0:32 EZBTTSSL
BPXROOT      65629          1  -   Sep 13 ?         1:50 HZSTKSCH
BPXROOT   33620066          1  -   Sep 10 ?         7:16 IOBSNMP
BPXROOT   50397286          1  -   Sep 13 ?         3:02 KN3ACTCS
BPXROOT   50397293          1  -   Sep 20 ?         0:31 PAGENT
BPXROOT   50397310          1  -   Sep 07 ?         2:16 EZASNMPD
BPXROOT      65679          1  -   Oct 02 ?         0:32 EZBTTMST
BPXROOT      65682          1  -   Oct 02 ?         0:32 EZBTTMST
BPXROOT      65694          1  -   Oct 02 ?         0:32 EZBTTMST
BPXROOT   67174594   16842954  - 16:38:51 ttyp0000  0:00 sh
   CS03   16842954   67174470  - 16:38:47 ttyp0000  0:39 sh -L
LBAGENT   16842956          1  - 10:13:02 ?         0:01 EZBLBAGE
BPXROOT   33620199          1  - 15:22:26 ?         0:03 EZBTCPIP
BPXROOT      65798          1  -   Sep 13 ?         3:02 KN3ANMON
BPXROOT      65803          1  -   Sep 13 ?         3:02 KLV
BPXROOT   33620236          1  -   Sep 13 ?         3:02 KLV
BPXROOT      65807          1  -   Sep 13 ?        19:18 KLV
BPXROOT   16843025          1  - 15:22:29 ?         0:03 EZACFALG
BPXROOT   16843032          1  - 15:22:29 ?         0:03 EZASASUB
BPXROOT   16843035          1  -   Sep 13 ?         4:04 KLV
BPXROOT   16843047          1  -   Oct 02 ?         0:32 EZBTMCTL
BPXROOT   50397484          1  -   Sep 05 ?        36:05 EZBTCPIP
```

Notice that in Example 3-19 on page 84, the UNIX System Services after this initialization is running with user ID BPXROOT. The reason for this is because 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.

Here are some typical UNIX commands:

- The **mkdir/u/cso1** command creates the directory for the user mount point. The permission bits would be set as specified in the etc/profile or $home/.profile.
- The **ls -all** command 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 provides 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

Example 3-20 on page 85 shows the startup of our TCP/IP stack.

Example 3-20 CS for z/OS IP startup

```
S TCPIPA
$HASP373 TCPIPA   STARTED
```

```
IEE252I MEMBER CTIEZB00 FOUND IN SYS1.IBM.PARMLIB 1
IEE252I MEMBER CTIIDS00 FOUND IN SYS1.IBM.PARMLIB
IEE252I MEMBER CTINTA00 FOUND IN SYS1.PARMLIB
EZZ0300I OPENED PROFILE FILE DD:PROFILE
EZZ0309I PROFILE PROCESSING BEGINNING FOR 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                   5
EZZ0338I TCP PORTS 1 THRU 1023 ARE NOT RESERVED
EZZ0338I UDP PORTS 1 THRU 1023 ARE NOT RESERVED
EZZ0613I TCPIPSTATISTICS IS ENABLED 6
EZZ4202I Z/OS UNIX - TCP/IP CONNECTION ESTABLISHED FOR TCPIPA    7
BPXF206I ROUTING INFORMATION FOR TRANSPORT DRIVER TCPIPA HAS BEEN
INITIALIZED OR UPDATED.
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE OSA2080
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE OSA20C0
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE OSA20E0
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE IUTIQDF5
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE IUTIQDF6
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE IUTIQDF4
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE OSA20A0
EZB6473I TCP/IP STACK FUNCTIONS INITIALIZATION COMPLETE.  8
EZAIN11I ALL TCPIP SERVICES FOR PROC TCPIPA ARE AVAILABLE.8
EZD1176I TCPIPA HAS SUCCESSFULLY JOINED THE TCP/IP SYSPLEX GROUP
EZBTCPCS
EZZ8302I VIPA 10.1.8.10 TAKEN FROM TCPIPC ON SC32
EZD1214I INITIAL DYNAMIC VIPA PROCESSING HAS COMPLETED FOR TCPIPA
EZZ8303I VIPA 10.1.8.10 GIVEN TO TCPIPA ON SC30
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE IUTIQDIO 9
EZZ8781I PAGENT CONNECTED TO POLICY SERVER FOR TCPIPA : PRIMARY AT
10.1.1.40
EZZ8790I PAGENT REMOTE POLICY PROCESSING COMPLETE FOR TCPIPA : QOS
IEE252I MEMBER BPXPRM9A FOUND IN SYS1.PARMLIB
```

- 1 Shows how the member that defines CTRACE processing has been found: CTIEZB00. This is discussed in “Taking a component trace” on page 227.
- 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.
- 5 Dynamic XCFs are enabled (DYNAMICXCF parameter).
- 6 TCPIPSTATICTICS will be generated.
- 7 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.
- 8 The stack (TCP/IP) is successfully initialized and READY FOR WORK.
- 9 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.

Important: Because 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 3-21 shows the output from the NETSTAT CONFIG display.

Example 3-21 NETSTAT CONFIG display

```
D TCPIP,TCPIPA,NETSTAT,CONFIG
EZD0101I NETSTAT CS V1R9 TCPIPA 739
TCP CONFIGURATION TABLE:                                          1
DEFAULTRCVBUFSIZE:  00065536  DEFAULTSNDBUFSIZE: 00065536
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:    NO        SENDGARBAGE:       NO
TCPTIMESTAMP:       YES       FINWAIT2TIME:      600
TTLS:               NO
UDP CONFIGURATION TABLE:                                          2
DEFAULTRCVBUFSIZE: 00065535  DEFAULTSNDBUFSIZE: 00065535
CHECKSUM:          YES
RESTRICTLOWPORT:   NO        UDPQUEUELIMIT:     YES
IP CONFIGURATION TABLE:                                           3
FORWARDING: YES    TIMETOLIVE: 00064  RSMTIMEOUT:  00060
IPSECURITY: NO
ARPTIMEOUT: 01200  MAXRSMSIZE: 65535  FORMAT:      LONG
IGREDIRECT: YES    SYSPLXROUT: YES    DOUBLENOP:   NO
STOPCLAWER: NO     SOURCEVIPA: NO
MULTIPATH:  NO     PATHMTUDSC: NO     DEVRTRYDUR:  0000000090
DYNAMICXCF: YES
  IPADDR: 10.1.7.11        SUBNET: 255.255.255.0    METRIC: 01
SECCLASS: 255
IQDIOROUTE: NO
TCPSTACKSRCVIPA: NO
IPV6 CONFIGURATION TABLE:
FORWARDING:    YES  HOPLIMIT:   00255  IGREDIRECT:  NO
SOURCEVIPA:    NO   MULTIPATH:  NO     ICMPERRLIM:  00003
IGRTRHOPLIMIT: NO
IPSECURITY: NO
DYNAMICXCF: NO
TCPSTACKSRCVIPA: NO
SMF PARAMETERS:                                                   4
TYPE 118:
TCPINIT:      00   TCPTERM:    00   FTPCLIENT:    00
  TN3270CLIENT: 00   TCPIPSTATS: 00
```

```
TYPE 119:
  TCPINIT:      NO   TCPTERM:    NO   FTPCLIENT:    NO
  TCPIPSTATS:   NO   IFSTATS:    NO   PORTSTATS:    NO
  STACK:        NO   UDPTERM:    NO   TN3270CLIENT: NO
GLOBAL CONFIGURATION INFORMATION:                                  5
TCPIPSTATS: NO   ECSALIMIT: 0000000K  POOLLIMIT: 0000000K
MLSCHKTERM: NO   XCFGRPID:      IQDVLANID:  0
SEGOFFLOAD: NO   SYSPLEXWLMPOLL: 060
EXPLICITBINDPORTRANGE:  00000-00000
SYSPLEX MONITOR:
  TIMERSECS: 0060  RECOVERY: NO   DELAYJOIN: NO   AUTOREJOIN: NO
  MONINTF:   NO    DYNROUTE: NO
ZIIP:
  IPSECURITY:NO
NETWORK MONITOR CONFIGURATION INFORMATION:                         6
PKTTRCSRV: NO   TCPCNNSRV: NO   SMFSRV: YES
AUTOLOG CONFIGURATION INFORMATION: WAIT TIME: 0300
PROCNAME: FTPDA     JOBNAME: FTPDA1   DELAYSTART: NO
  PARMSTRING:
PROCNAME: OMPA      JOBNAME: OMPA     DELAYSTART: NO
  PARMSTRING:
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 the settings 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

You can display the status of devices by using the MVS display command, TSO NETSTAT, or UNIX **onetstat -d**; see Example 3-22.

Example 3-22 Results of device display

```
D TCPIP,TCPIPA,N,DEV
MVS TCP/IP NETSTAT CS V1R9       TCPIP Name: TCPIPA          17:10:41
.....
DevName: OSA2080           DevType: MPCIPA
  DevStatus: Ready   1
  LnkName: OSA2080L          LnkType: IPAQENET   LnkStatus: Ready 1
    NetNum: n/a  QueSize: n/a  Speed: 0000001000
IpBroadcastCapability: No
    CfgRouter: Non                   ActRouter: Non
    ArpOffload: Yes                  ArpOffloadInfo: Yes 2
    ActMtu: 8992
    VLANid: 10 3                VLANpriority: Disabled
    DynVLANRegCfg: No                DynVLANRegCap: Yes
    ReadStorage: GLOBAL (4096K)      InbPerf: Balanced
    ChecksumOffload: Yes
    SecClass: 255                    MonSysplex: No
  BSD Routing Parameters:
    MTU Size: 1492              Metric: 100
    DestAddr: 0.0.0.0           SubnetMask: 255.255.255.0
  Multicast Specific:
    Multicast Capability: Yes
```

```
    Group             RefCnt        SrcFltMd
    -----             ------        --------
    224.0.0.1         0000000001    Exclude
      SrcAddr: None
  Link Statistics:
    BytesIn                          = 1756
    Inbound Packets                  = 7
    Inbound Packets In Error         = 0
    Inbound Packets Discarded        = 0
    Inbound Packets With No Protocol = 0
    BytesOut                         = 2148
    Outbound Packets                 = 9
    Outbound Packets In Error        = 0
    Outbound Packets Discarded       = 0
.....
```

- 1 Indicates the overall status of the OSA device OSA2080: READY. If this status is not READY, verify that the VTAM Major node is active. You can do this by using the VTAM command.
- 2 The OFFLOAD feature is enabled.
- 3 The VLAN ID defined on the LINK statement in the PROFILE data set.

Example 3-23 Results of the TRLE display

```
D NET,TRL,TRLE=OSA2080P
IST097I DISPLAY ACCEPTED
IST075I NAME = OSA2080P, TYPE = TRLE 764
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 = 087A
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 = TCPIPA
```

- 1 The Major node is ACTIVE and ONLINE.
- 2 The READ and WRITE channels are ACTIVE and ONLINE.
- 3 The data channel is also ACTIVE.

Display storage usage

The z/OS Communications Server uses the Common 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. 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, as illustrated in Example 3-24.

Example 3-24 Results of storage display

```
D TCPIP,TCPIPE,STOR
EZZ8453I TCPIP STORAGE
EZZ8454I TCPIPA   STORAGE    CURRENT   MAXIMUM     LIMIT
EZZ8455I TCPIPA   ECSA         9026K    10066K   NOLIMIT
EZZ8455I TCPIPA   POOL         6388K     6430K   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 9*, SC31-8781. 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 9*, SC31-8781.

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 you run HOMETEST, the output shown in Example 3-25 is displayed.

Example 3-25 Hometest command output

```
EZAO619I Running IBM MVS TCP/IP CS V1R9 TCP/IP Configuration Tester

EZA9431I FTP.DATA file not found. Using hardcoded default values.

EZAO602I TCP Host Name is: WTSC30A

EZAO605I Using Name Server to Resolve WTSC30A
EZAO611I The following IP addresses correspond to TCP Host Name: WTSC30A
EZAO612I 10.1.1.10
EZAO614I The following IP addresses are the HOME IP addresses defined in PROFILE.TCPIP:
 EZAO615I 10.1.30.10
 EZAO615I 10.1.1.10
 EZAO615I 10.1.2.10
 EZAO615I 10.1.2.11
 EZAO615I 10.1.3.11
 EZAO615I 10.1.3.12
 EZAO615I 10.1.2.12
 EZAO615I 10.1.4.11
 EZAO615I 10.1.5.11
 EZAO615I 10.1.6.11
 EZAO615I 10.1.7.11
 EZAO615I 10.1.7.11
 EZAO615I 10.1.8.10
 EZAO615I 10.1.8.20
 EZAO615I 127.0.0.1

 EZAO618I All IP addresses for WTSC30A are in the HOME list!

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

3.7 Reconfiguring 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 about the VARY OBEYFILE command, see *z/OS CS: IP System Administrator's Commands*, SC31-8781. For more information about RACF definitions, see *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 2: Standard Applications*, SG24-7533.

3.7.1 Deleting a device and adding or 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, the addition of new devices, or the 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.

Deleting and adding back a device

If you want to delete a device, then the order of the steps you perform 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.

3.7.2 Modifying a device

In this example we wanted to change the address of OSA-Express device OSA2080 at address 2080 from 10.1.2.11 to 10.1.2.14. This process would involve deleting the current definition and redefining the device.

Example 3-26 shows all devices in the TCPIPA stack. Under each device a single link is defined, which is associated with an IP address.

Example 3-26 Displays of devices and home before deletion

```
D TCPIP,TCPIPA,N,HOME
```

```
MVS TCP/IP NETSTAT CS V1R9       TCPIP Name: TCPIPA          08:50:26
Home address list:
LinkName:   VIPA3L
  Address:  10.1.30.10
    Flags:
LinkName:   VIPA1L
  Address:  10.1.1.10
    Flags:  Primary
LinkName:   VIPA2L
  Address:  10.1.2.10
    Flags:
LinkName:   OSA2080L
  Address:  10.1.2.11
    Flags:
LinkName:   OSA20C0L
  Address:  10.1.3.11
    Flags:
LinkName:   OSA20E0L
  Address:  10.1.3.12
    Flags:
LinkName:   OSA20A0L
  Address:  10.1.2.12
    Flags:
LinkName:   IUTIQDF4L
  Address:  10.1.4.11
    Flags:
LinkName:   IUTIQDF5L
  Address:  10.1.5.11
    Flags:
LinkName:   IUTIQDF6L
  Address:  10.1.6.11
    Flags:
LinkName:   EZASAMEMVS
  Address:  10.1.7.11
    Flags:
LinkName:   IQDIOLNK0A01070B
  Address:  10.1.7.11
    Flags:
LinkName:   VIPLOA01080A
  Address:  10.1.8.10
    Flags:
LinkName:   VIPLOA010814
  Address:  10.1.8.20
    Flags:  Internal
LinkName:   LOOPBACK
  Address:  127.0.0.1
    Flags:
IntfName:   LOOPBACK6
  Address:  ::1
    Type:   Loopback
    Flags:
```

Notice the address of OSA2080L (10.1.2.11). We needed 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 execute STOP and DELETE in one step. Trying to do so will result in the error messages displayed in Example 3-27.

Example 3-27 Timing problems with device/link deletion

```
V TCPIP,TCPIPA,O,TCPIPA.TCPPARMS(DELA30)
EZZ0060I PROCESSING COMMAND: VARY TCPIP,TCPIPA,O,TCPIPA.TCPPARMS(DELA30)
EZZ0300I OPENED OBEYFILE FILE 'TCPIPA.TCPPARMS(DELA30)'
EZZ0309I PROFILE PROCESSING BEGINNING FOR 'TCPIPA.TCPPARMS(DELA30)'
EZZ0395I DELETE LINK OSA2080L ON LINE 16 FAILED BECAUSE LINK IS ACTIVE
EZZ0395I DELETE DEVICE OSA2080 ON LINE 17 FAILED BECAUSE DEVICE IS ACTIVE
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'TCPIPA.TCPPARMS(DELA30)'
EZZ0303I OBEYFILE FILE CONTAINS ERRORS
EZZ0331I NO HOME ADDRESS ASSIGNED TO LINK OSA2080L
EZZ0059I VARY OBEY COMMAND FAILED: SEE PREVIOUS MESSAGES
```

We also wanted to delete the LINK *before* the DEVICE (step 3 and step 4 in "Deleting and adding back a device" on page 92) 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 error displayed in Example 3-28 occurs.

Example 3-28 Another timing problem with device/link deletion

```
V TCPIP,TCPIPE,O,TCPIPE.TCPPARMS(DELE30)
EZZ0060I PROCESSING COMMAND: VARY TCPIP,TCPIPA,O,TCPIPA.TCPPARMS(DELA30)
EZZ0300I OPENED OBEYFILE FILE 'TCPIPA.TCPPARMS(DELA30)'
EZZ0309I PROFILE PROCESSING BEGINNING FOR 'TCPIPA.TCPPARMS(DELA30)'
EZZ0395I DELETE DEVICE OSA2080 ON LINE 16 FAILED BECAUSE DEVICE IS ACTIVE
EZZ0395I DELETE LINK OSA2080L ON LINE 17 FAILED BECAUSE LINK IS ACTIVE
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'TCPIPA.TCPPARMS(DELA30)'
EZZ0303I OBEYFILE FILE CONTAINS ERRORS
EZZ0331I NO HOME ADDRESS ASSIGNED TO LINK OSA2080L
EZZ0059I VARY OBEY COMMAND FAILED: SEE PREVIOUS MESSAGES
```

Next, we changed the characteristic of the device and link of OSA2080L. In the following steps we stopped the device, as shown in Example 3-29.

Example 3-29 Command to stop the device

```
V TCPIP,TCPIPA,STOP,OSA2080
EZZ0060I PROCESSING COMMAND: VARY TCPIP,TCPIPA,STOP,OSA2080
EZZ0053I COMMAND VARY STOP COMPLETED SUCCESSFULLY
EZZ0331I NO HOME ADDRESS ASSIGNED TO LINK OSA2080L
EZZ4329I LINK OSA20A0L HAS TAKEN OVER ARP RESPONSIBILITY FOR INACTIVE
LINK OSA2080L
EZZ4315I DEACTIVATION COMPLETE FOR DEVICE OSA2080
```

Then we deleted it from the stack, as shown in Example 3-30.

Example 3-30 Command to delete the device

```
V TCPIP,TCPIPA,O,TCPIPA.TCPPARMS(DELA30)
EZZ0060I PROCESSING COMMAND: VARY TCPIP,TCPIPA,O,TCPIPA.TCPPARMS(DELA30)
EZZ0300I OPENED OBEYFILE FILE 'TCPIPA.TCPPARMS(DELA30)'
EZZ0309I PROFILE PROCESSING BEGINNING FOR 'TCPIPA.TCPPARMS(DELA30)'
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'TCPIPA.TCPPARMS(DELA30)'
EZZ0053I COMMAND VARY OBEY COMPLETED SUCCESSFULLY
```

The results of the delete are shown in the display NETSTAT for the HOME addresses; see Example 3-31. Notice the missing OSA2080LNK link as compared with Example 3-26 on page 92.

Example 3-31 Omission of OSA2080LNK

```
D TCPIP,TCPIPE,N,HOME
EZD0101I NETSTAT CS V1R9 TCPIP
HOME ADDRESS LIST:
LINKNAME:   VIPA1L
  ADDRESS:  10.1.1.10
    FLAGS:  PRIMARY
LINKNAME:   VIPA2L
  ADDRESS:  10.1.2.10
    FLAGS:
LINKNAME:   OSA20C0L
  ADDRESS:  10.1.3.11
    FLAGS:
LINKNAME:   OSA20E0L
  ADDRESS:  10.1.3.12
    FLAGS:
LINKNAME:   OSA20A0L
  ADDRESS:  10.1.2.12
    FLAGS:
LINKNAME:   IUTIQDF4L
  ADDRESS:  10.1.4.11
    FLAGS:
LINKNAME:   IUTIQDF5L
  ADDRESS:  10.1.5.11
    FLAGS:
LINKNAME:   IUTIQDF6L
  ADDRESS:  10.1.6.11
FLAGS:
LINKNAME:   EZASAMEMVS
  ADDRESS:  10.1.7.11
    FLAGS:
LINKNAME:   IQDIOLNK0A01070B
  ADDRESS:  10.1.7.11
    FLAGS:
LINKNAME:   VIPLOA01080A
  ADDRESS:  10.1.8.10
    FLAGS:
LINKNAME:   VIPLOA010814
  ADDRESS:  10.1.8.20
    FLAGS:  INTERNAL
LINKNAME:   LOOPBACK
  ADDRESS:  127.0.0.1
    FLAGS:
INTFNAME:   LOOPBACK6
  ADDRESS:  ::1
    TYPE:   LOOPBACK
    FLAGS:
14 OF 14 RECORDS DISPLAYED
END OF THE REPORT
```

Example 3-32 displays the statements necessary to delete the IP address, the LINK, and the DEVICE.

Example 3-32 OBEYFILE member to delete the device OSA2080

```
HOME
   10.1.1.10     VIPA1L
   10.1.2.10     VIPA2L
;;;10.1.2.11     OSA2080L
```

```
   10.1.3.11     OSA20C0L
   10.1.3.12     OSA20E0L
   10.1.2.12     OSA20A0L
   10.1.4.11     IUTIQDF4L
   10.1.5.11     IUTIQDF5L
   10.1.6.11     IUTIQDF6L
;
 DELETE  LINK  OSA2080L
 DELETE DEVICE OSA2080
```

Then we added the device and link back with the changed address definition **3**, as Example 3-33 illustrates.

Example 3-33 OBEYFILE member to add the device (ADDA30)

```
DEVICE OSA2080  MPCIPA
 LINK   OSA2080L  IPAQENET   OSA2080 VLANID 10
;
HOME
   10.1.1.10     VIPA1L
   10.1.2.10     VIPA2L
   10.1.2.14     OSA2080L                    3
   10.1.3.11     OSA20C0L
   10.1.3.12     OSA20E0L
   10.1.2.12     OSA20A0L
   10.1.4.11     IUTIQDF4L
   10.1.5.11     IUTIQDF5L
   10.1.6.11     IUTIQDF6L
```

We issued the command shown in Example 3-34 to add the device and link associated with its own IP address.

Example 3-34 Adding the device and link

```
V TCPIP,TCPIPA,O,TCPIPA.TCPPARMS(ADDA30)
EZZ0060I PROCESSING COMMAND: VARY TCPIP,TCPIPA,O,TCPIPA.TCPPARMS(ADDA30)
EZZ0300I OPENED OBEYFILE FILE 'TCPIPA.TCPPARMS(ADDA30)'
EZZ0309I PROFILE PROCESSING BEGINNING FOR 'TCPIPA.TCPPARMS(ADDA30)'
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'TCPIPA.TCPPARMS(ADDA30)'
EZZ0053I COMMAND VARY OBEY COMPLETED SUCCESSFULLY
```

This is followed by a display to verify the addition to the stack, as shown in Example 3-35.

Example 3-35 Display with OSA2080 using a new address

```
D TCPIP,TCPIPE,N,HOME
EZD0101I NETSTAT CS V1R9 TCPIPA
HOME ADDRESS LIST:
LINKNAME:   VIPA1L
  ADDRESS:  10.1.1.10
    FLAGS:  PRIMARY
LINKNAME:   VIPA2L
  ADDRESS:  10.1.2.10
    FLAGS:
LINKNAME:   OSA2080L
  ADDRESS:  10.1.2.14
    FLAGS:
LINKNAME:   OSA20C0L
  ADDRESS:  10.1.3.11
    FLAGS:
```

```
LINKNAME:   OSA2OE0L
  ADDRESS:  10.1.3.12
    FLAGS:
LINKNAME:   OSA2OA0L
  ADDRESS:  10.1.2.12
    FLAGS:
LINKNAME:   IUTIQDF4L
  ADDRESS:  10.1.4.11
    FLAGS:
LINKNAME:   IUTIQDF5L
ADDRESS:  10.1.5.11
    FLAGS:
LINKNAME:   IUTIQDF6L
  ADDRESS:  10.1.6.11
    FLAGS:
LINKNAME:   EZASAMEMVS
  ADDRESS:  10.1.7.11
    FLAGS:
LINKNAME:   IQDIOLNK0A01070B
  ADDRESS:  10.1.7.11
    FLAGS:
LINKNAME:   VIPL0A01080A
  ADDRESS:  10.1.8.10
    FLAGS:
LINKNAME:   VIPL0A010814
  ADDRESS:  10.1.8.20
    FLAGS:  INTERNAL
LINKNAME:   LOOPBACK
  ADDRESS:  127.0.0.1
    FLAGS:
INTFNAME:   LOOPBACK6
  ADDRESS:  ::1
    TYPE:   LOOPBACK
    FLAGS:
15 OF 15 RECORDS DISPLAYED
END OF THE REPORT
```

3.8 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 *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 2: Standard Applications*, SG24-7533, for more information about the syslog daemon. Individual server documentation also provides information about diagnosis.

3.9 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:

Message type	Publication
Messages with prefix BPX	*z/OS MVS System Messages, Vol 3 (ASB-BPX)*, SA22-7633

Message type	Publication
Messages with prefix EZA	For Communications Server for z/OS IP, refer to *z/OS Communications Server: IP Messages Volume 1 (EZA)*, SC31-8783
Messages with prefix EZB	For Communications Server for z/OS IP, refer to *z/OS Communications Server: IP Messages Volume 2 (EZB, EZD)*, SC31-8784
Messages with prefix EZY	For Communications Server for z/OS IP, refer to *z/OS Communications Server: IP Messages Volume 3 (EZY)*, SC31-8785
Messages with prefix EZZ and SNM™	For Communications Server for z/OS IP, refer to *z/OS Communications Server: IP Messages Volume 4 (EZZ, SNM)*, SC31-8786
Messages with prefix FOMC, FOMM, FOMO, FSUC, and FSUM	*z/OS UNIX System Services Messages and Codes*, SA22-7807
Eight-digit SNA sense codes and DLC codes	*z/OS Communications Server: IP and SNA Codes*, SC31-8791
UNIX System Services return codes and reason codes	*z/OS UNIX System Services Messages and Codes*, SA22-7807

3.10 Health Checker for z/OS

IBM Health Checker is a component of z/OS that can be used to gather information about the 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 z/OS framework.

IBM Health Checker for z/OS can be used to check whether appropriate values are defined for the maximum amount of storage to be dedicated to fixed buffers and ECSA buffers in CSM.

In z/OS Communication Server, TCP/IP is used by Health Checker to perform the following checks:

- A check related to the TCPMAXRCVBUFRSIZE parameter on the TCPCONFIG statement

  This check determines if the value specified on the TCPMAXRCVBUFRSIZE is large enough to accommodate FTP use. FTP requires a TCPMAXRCVBUFRSIZE of at least 180 K.

- A check related to CTRACE

  This check determines if the TCP/IP Event Trace (SYSTCPIP) is running with more than the default options. If so, an exception message is issued, suggesting that additional trace options beyond the default be turned off.

- A check to determine whether the GLOBALCONFIG SYSPLEXMONITOR RECOVERY option is enabled when the IPCONFIG DYNAMICXCF or IPCONFIG6 DYNAMICXCF options have been configured

  The RECOVERY option enables a TCP/IP stack in a sysplex to perform internal checks to determine whether the stack is unhealthy and should therefore remove itself from the

sysplex and allow a healthy backup TCP/IP stack to take over the ownership of the DVIPA interfaces. This enables continued availability to applications.

Figure 3-6 shows a sample display of the resources Health Checker checks.

```
F HEALTHCK,DISPLAY,CHECKS
HZS0200I 10.25.57 CHECK SUMMARY
CHECK OWNER      CHECK NAME                              STATE   STATUS
IBMCS            CSTCP_SYSPLEXMON_RECOV_TCPCS1           AE      EXCEPTION-LOW
IBMCS            CSTCP_TCPMAXRCVBUFRSIZE_TCPCS1          AE      SUCCESSFUL
IBMCS            CSTCP_SYSTCPIP_CTRACE_TCPCS1            AE      EXCEPTION-LOW
IBMCS            CSVTAM_T1BUF_T2BUF_NOEE                 AE      SUCCESSFUL
IBMCS            CSVTAM_T1BUF_T2BUF_EE                   AD      ENV N/A
IBMCS            CSVTAM_VIT_OPT_ALL                      AE      EXCEPTION-LOW
IBMCS            CSVTAM_VIT_DSPSIZE                      AE      EXCEPTION-LOW
IBMCS            CSVTAM_VIT_OPT_PSSSMS                   AE      SUCCESSFUL
IBMCS            CSVTAM_VIT_SIZE                         AE      EXCEPTION-LOW
IBMCS            CSVTAM_CSM_STG_LIMIT                    AE      SUCCESSFUL
IBMUSS           USS_MAXSOCKETS_MAXFILEPROC              AD      UNEXP ERROR
IBMUSS           USS_AUTOMOUNT_DELAY                     AD      ENV N/A
IBMUSS           USS_FILESYS_CONFIG                      AE      EXCEPTION-MED
IBMIXGLOGR       IXGLOGR_ENTRYTHRESHOLD                  AE      SUCCESSFUL
```

Figure 3-6 Display of the resources checked by Health Checker

The letters in the state column can be:

- A (active)
- I (inactive)
- E (enabled)
- D (disabled)

The status field of the display shows the status of the check (that is, whether the check was successful or generated an exception message).

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,* SA22-7994. 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/

3.11 For additional information

When you install and customize the Communications Server for z/OS 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 Program Directory, Program Number 5694-A01*, GI10-0670
- *z/OS XL C/C++ Run-Time Library Reference*, SA22-7821
- *z/OS Migration*, GA22-7499
- *z/OS Communications Server: IP Configuration Guide*, SC31-8775

- *z/OS Communications Server: IP Configuration Reference*, SC31-8776
- *z/OS Communications Server: IP Messages Volume 1 (EZA)*, SC31-8783
- *z/OS Communications Server: IP Messages Volume 2 (EZB, EZD)*, SC31-8784
- *z/OS Communications Server: IP Messages Volume 3 (EZY)*, SC31-8785
- *z/OS Communications Server: IP Messages Volume 4 (EZZ, SNM)*, SC31-8786
- *z/OS Communications Server: IP and SNA Codes*, SC31-8791
- *z/OS UNIX System Services Planning*, GA22-7800
- *z/OS UNIX System Services User's Guide*, SA22-7801
- *z/OS UNIX System Services Messages and Codes*, SA22-7807
- *z/OS MVS System Messages, Vol 1 (ABA-AOM)*, SA22-7631
- *z/OS MVS System Messages, Vol 2 (ARC-ASA)*, SA22-7632
- *z/OS MVS System Messages, Vol 3 (ASB-BPX)*, SA22-7633
- *z/OS MVS System Messages, Vol 4 (CBD-DMO)*, SA22-7634
- *z/OS MVS System Messages, Vol 5 (EDG-GFS)*, SA22-7635
- *z/OS MVS System Messages, Vol 6 (GOS-IEA)*, SA22-7636
- *z/OS MVS System Messages, Vol 7 (IEB-IEE)*, SA22-7637
- *z/OS MVS System Messages, Vol 8 (IEF-IGD)*, SA22-7638
- *z/OS MVS System Messages, Vol 9 (IGF-IWM)*, SA22-7639
- *z/OS MVS System Messages, Vol 10 (IXC-IZP)*, SA22-7640

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 mainframes, this chapter discusses the most commonly used and the most dynamic types of mainframe connectivity.

Detailed topics regarding these interfaces are provided, including useful implementation information, design scenarios, and setup examples.

This chapter discusses the following topics.

Section	Topic
4.1, “What is connectivity” on page 102	Network connectivity options supported by z/OS Communications Server (TCP/IP) and IBM System z servers Key characteristics of VLAN implementation
4.3, “Connectivity for the z/OS environment” on page 112	Basic implementation information for z/OS and Communications Server when connecting to the immediate LAN environment
4.4, “OSA-Express connectivity” on page 114	Configuration examples, with dependencies, considerations, and our recommendations for an OSA-Express interface
4.5, “HiperSockets connectivity” on page 121	Configuration examples, with dependencies, considerations, and our recommendations for a HiperSockets interface
4.6, “Dynamic XCF connectivity” on page 126	Configuration examples, with dependencies, considerations, and our recommendations for a dynamic XCF interface
4.7, “Controlling and activating devices” on page 132	Commands to start and stop devices, as well as activate modified device definitions
4.8, “Problem determination” on page 134	How to determine why certain connectivity options are not working
4.9, “References” on page 140	Useful documentation based on connectivity type

4.1 What is connectivity

Connectivity is the pipeline through which data is exchanged between clients and servers via physical and logical communication interfaces and the network. IBM System z 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. Others support Local Area Network (LAN) connectivity. Figure 4-1 depicts the physical interfaces (and device types) provided by System z servers. The physical network interface is enabled through z/OS Communications Server (TCP/IP) definitions.

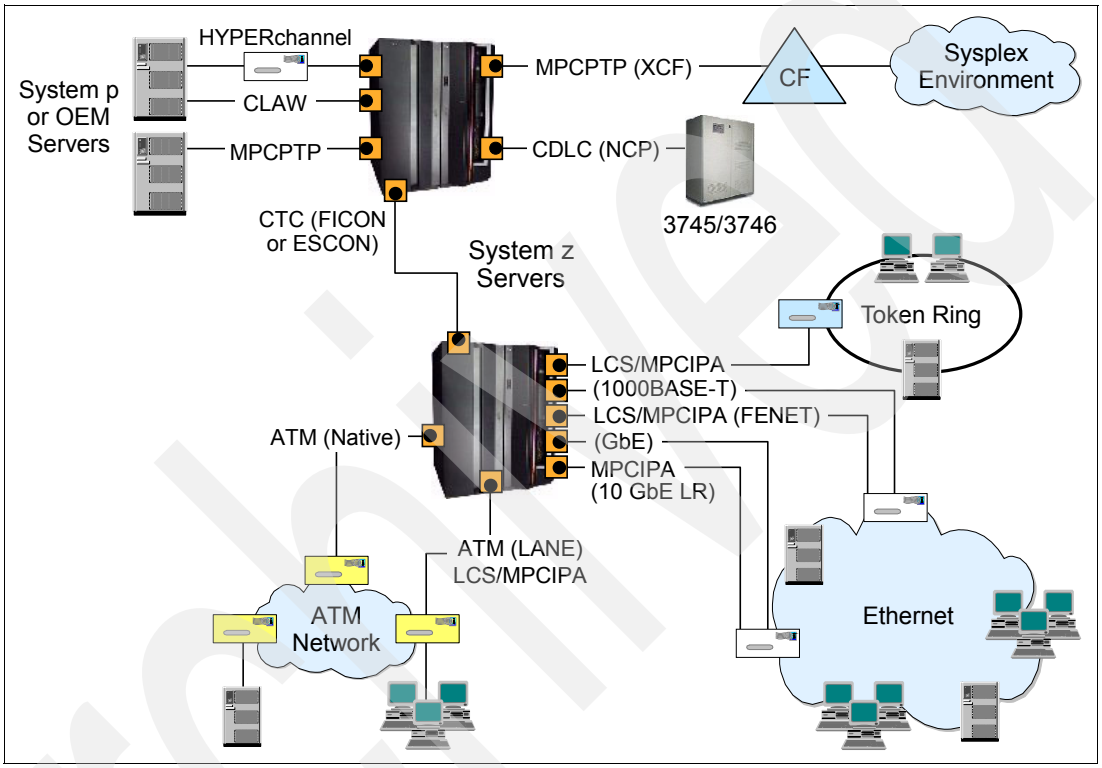


Figure 4-1 System z - physical interfaces

4.1.1 System z network connectivity

System z network connectivity is handled by the physical and logical interfaces to enable the transport of IP datagrams. Using the OSI model as an example, it spans Layer 1 (physical layer) and Layer 2 (Data link control Layer). The z/OS Communications Server supports several types of interfaces connecting 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). The supported IPv4 interfaces are listed in Table 4-1 on page 103.

Table 4-1 System z network interfaces

Interface type	Attachment type	Protocol type	Description
Asynchronous transfer mode (ATM)	ATM Native mode via OSA-Express	ATM network	Enables TCP/IP to send data to an ATM network using an OSA-Express ATM adapter.
Channel data link control (CDLC)	Network Control Program via 3745/3746	Point-to-point	ESCON® attachments can be used to provide native IP transport between the 3746 IP and host systems running the z/OS Communications Server.
Common link access to workstation (CLAW)	IBM System p™ Channel attached routers	Point-to-point Point-to-Multipoint	Provides access from IBM System p server directly to a TCP/IP stack over a channel. Can also be used to provide connectivity to other vendor platforms.
Channel-to-channel (CTC)	FICON/ESCON channel	Point-to-point)	Provides access to TCP/IP hosts by way of a CTC connection established over a FICON or ESCON channel.
HYPERchannel	Series A devices	Point-to-Multipoint	Provides access to TCP/IP hosts by way of a series A devices and series DX devices that function as series A devices.
LAN Channel Station (LCS)	OSA-Express: ► 1000BASE-T ► Fast Ethernet ► Token Ring ► ATM Native and LAN Emulation	LAN: ► IEEE802.3 ► IEEE802.3 ► IEEE802.5 ► ATM network	A variety of channel adapters support a protocol called the LCS. The most common are OSA-Express features.
MultiPath Channel IP Assist (MPCIPA)	HiperSockets[a] OSA-Express: ► 10 Gigabit Ethernet ► Gigabit Ethernet ► 1000BASE-T ► Fast Ethernet ► Token Ring ► ATM LAN Emulation	Internal LAN LAN: ► IEEE802.3 ► IEEE802.3 ► IEEE802.3 ► IEEE802.3 ► IEEE802.5 ► ATM network	Provides access to TCP/IP hosts, using OSA-Express in Queued Direct I/O (QDIO) mode and HiperSockets using the internal Queued Direct I/O (iQDIO).
MultiPath Channel Point-to-Point (MPCPTP)	IUTSAMEH (XCF link)	Point-to-point	Provides access to directly connect z/OS hosts or z/OS LPARs, or by configuring it to utilize Coupling Facility links (if it is part of a sysplex).
SAMEHOST (Data Link Control)	SNALINK LU0 SNALINK LU6.2 X25NPSI	Point-to-point Point-to-point X.25 network	Enables communication between CS for z/OS IP and other servers running on the same MVS image.

a. Can also be used in conjunction with DYNAMICXCF

For further information about these protocols, refer to *z/OS Communications Server: IP Configuration Reference*, SC31-8776.

4.2 Recommended interfaces

This section discusses the recommended interfaces supported by System z hardware and z/OS Communications Server. We highly recommend their use 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

z/OS Communications Server supports high-bandwidth and high-speed networking technologies provided by OSA-Express and HiperSockets.

- The OSA-Express features comply with the most commonly used IEEE standards, used in LAN environments.
- HiperSockets is used for transporting IP traffic between TCP/IP stacks running in logical partitions (LPARs) within a System z server at memory speed.

Both interfaces use the System z I/O architecture called queued direct input/output (QDIO).

QDIO is a highly efficient data transfer mechanism that satisfies the increasing volume of applications and 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.

HiperSockets implementation is based on the OSA-Express QDIO protocol, hence the name internal QDIO (iQDIO). The System z microcode for HiperSockets 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.

Recommendation: Some OSA-Express features also support LCS (known as non-QDIO mode). However, we recommend the use of QDIO mode in conjunction with the OSA-Express Ethernet features wherever possible.

With QDIO, 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.

z/OS Communications Server can only transport IP traffic over OSA-Express in QDIO mode and HiperSockets. However, SNA can be transported over IP connections using encapsulation technologies such as Enterprise Extender (EE) and TN3270.

For more information about EE, refer to *Enterprise Extender Implementation Guide,* SC24-7359. For TN3270 details, refer to *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 2: Standard Applications*, SG24-7533.

4.2.1 OSA-Express (MPCIPA)

As mentioned, the OSA-Express can use the I/O architecture called QDIO when defined as channel type (CHPID) OSD. QDIO 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.

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.

The Dynamic OSA Address Table (OAT) allows for IP addresses defined in the HOME statement of the TCP/IP stack to be downloaded into the OSA-Express when the interface is started.

Table 4-2 lists the OSA-Express2 and OSA-Express Ethernet features that are available on the System z 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

OSA-Express VLAN support

The OSA-Express Ethernet features also support IEEE standards 802.1p/q (priority tagging and VLAN identifier tagging). Deploying VLAN IDs enables 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. It allows a TCP/IP stack to register a specific VLAN ID for both IPv4 and IPv6 for the same OSA-Express port. Note that the VLAN ID for IPv4 can be different than the VLAN ID for IPv6.

When a VLAN ID is configured for an OSA-Express interface in the TCP/IP stack, the following occurs:

- The TCP/IP stack becomes VLAN-aware or enabled, and the OSA-Express port is considered to be part of a VLAN.
- During activation, the TCP/IP stack registers the VLAN ID value to the OSA-Express port.
- A VLAN tag is added to all outbound packets.
- The OSA-Express port filters all inbound packets based on the configured VLAN ID.

If the TCP/IP stack is also configured with PRIRouter or SECRouter for an OSA-Express port that has a VLAN ID defined, then the stack serves as an IP router for the configured VLAN ID.

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. 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 based 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. In Figure 4-2 you see how the PRIRouter function works in a shared OSA environment.

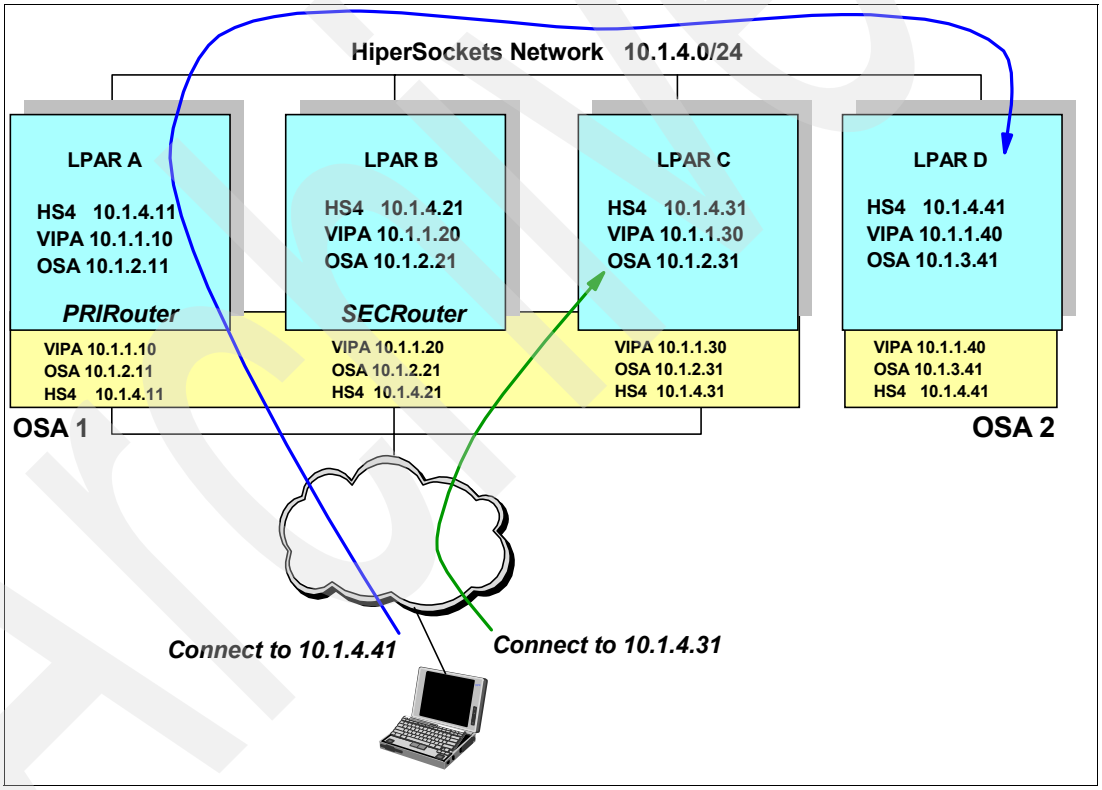


Figure 4-2 How PRIRouter works with a shared OSA

In Figure 4-2 the terminal user connects to 10.1.4.41. Note that each stack sharing OSA1 has registered the IP addresses in the OSA Address Table (OAT): addresses for VIPAs, the OSAs, and the HiperSockets. However, the address 10.1.4.41 is not represented in OSA1's OAT. Therefore, the packet from the terminal that arrives at OSA1 is sent to the primary routing TCP/IP stack in LPAR A. The TCP/IP stack in LPAR A uses its routing table to forward the packet to LPAR D, where IP address 10.1.4.41 resides.

The connection to IP address 10.1.4.31 is simpler. Because the address is represented in the OAT of OSA1, the OSA can immediately forward the request to the correct TCP/IP stack in LPAR C.

If LPAR A should become unavailable, then the TCP/IP stack in LPAR B or C will take over the routing responsibility for OSA1.

VLAN and primary/secondary router support

The OSA-Express primary router support takes into consideration VLAN ID support (VLAN ID registration and tagging) and interacts with it. OSA-Express supports a primary and secondary router on a per-VLAN basis (per registered VLAN ID).

Therefore, if an OSA interface 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 primary routers (one per VLAN) has many advantages and preserves traffic isolation for each VLAN.

If OSA-Express ports are shared across multiple TCP/IP routing stacks, consider using virtual MAC support for your environment instead of the PRIRouter and SECRouter options. See Chapter 6, “Virtual Medium Access Control (VMAC) support” on page 195 for details.

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 LPAR-to-LPAR communications within the same sever (via memory). It also provides secure data flows between LPARs and high availability, if there is no network attachment dependency or exposure to adapter failures.

HiperSockets can be used to communicate among consolidated servers within a single System z 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 z server can do so at memory speeds, bypassing all the network overhead and delays.

HiperSockets can be customized to accommodate varying traffic sizes. With HiperSockets, a maximum frame size can be defined according to the traffic characteristics transported for each HiperSockets.

Because there is no server-to-server traffic outside the System z 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:

- HiperSockets has no external components. It provides a very secure connection. For security purposes, servers can be connected to different HiperSockets or VLANs within the same HiperSockets. All security features, like IPSec or IP 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 System z server to transfer IP packets to the target systems).

The HiperSockets device is represented by the IQD channel ID (CHPID) and its associated subchannel devices. All LPARs that are configured in HCD/IOCP to use the same IQD CHPID have internal connectivity and therefore have the capability to communicate using HiperSockets.

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 that supports HiperSockets and the CHPIDs has 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. System z servers allow up to 16 separate IQD CHPIDs, creating the capability of having up to 16 separate logical LANs within the same 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 4-3.

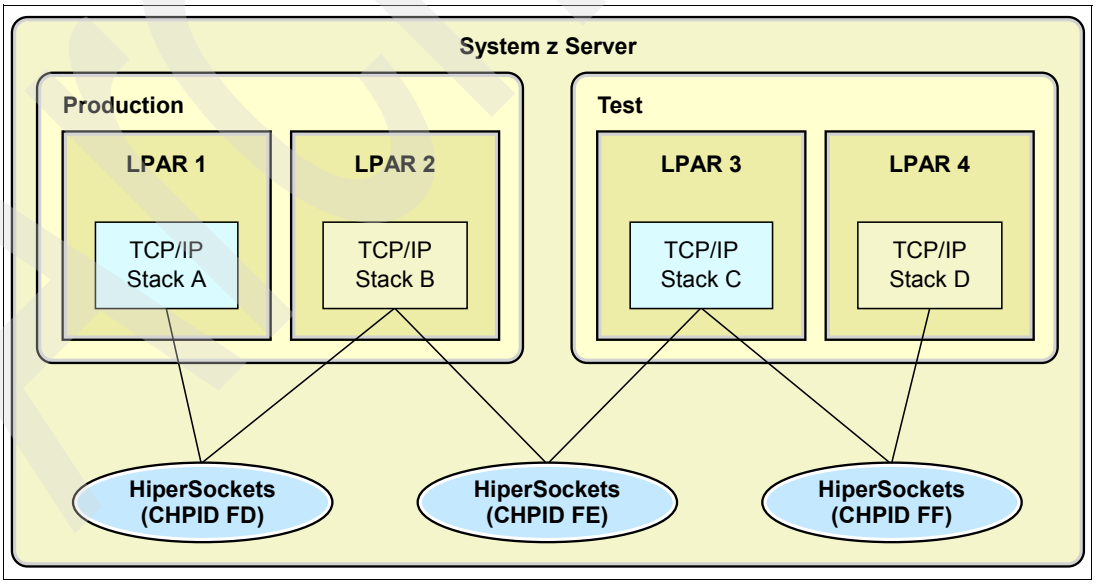


Figure 4-3 HiperSockets - multiple logical LANs

HiperSockets VLAN support

HiperSockets connections defined through DYNAMICXCF coding or through individual DEVICE and LINK statement coding also support VLAN tagging. This allows you to split the internal LAN represented by a single HiperSockets CHPID into multiple virtual LANs, providing isolation for security or administrative purposes. Only stacks attached to the same HiperSockets VLAN can communicate with each other. Stacks attached to a different HiperSockets VLAN on the same CHPID cannot use the HiperSockets path to communicate with the stacks on a different VLAN.

Note: The VLAN ID assigned to a HiperSockets device applies to both IPv4 and IPv6 connections over that CHPID.

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 z 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 4-4.

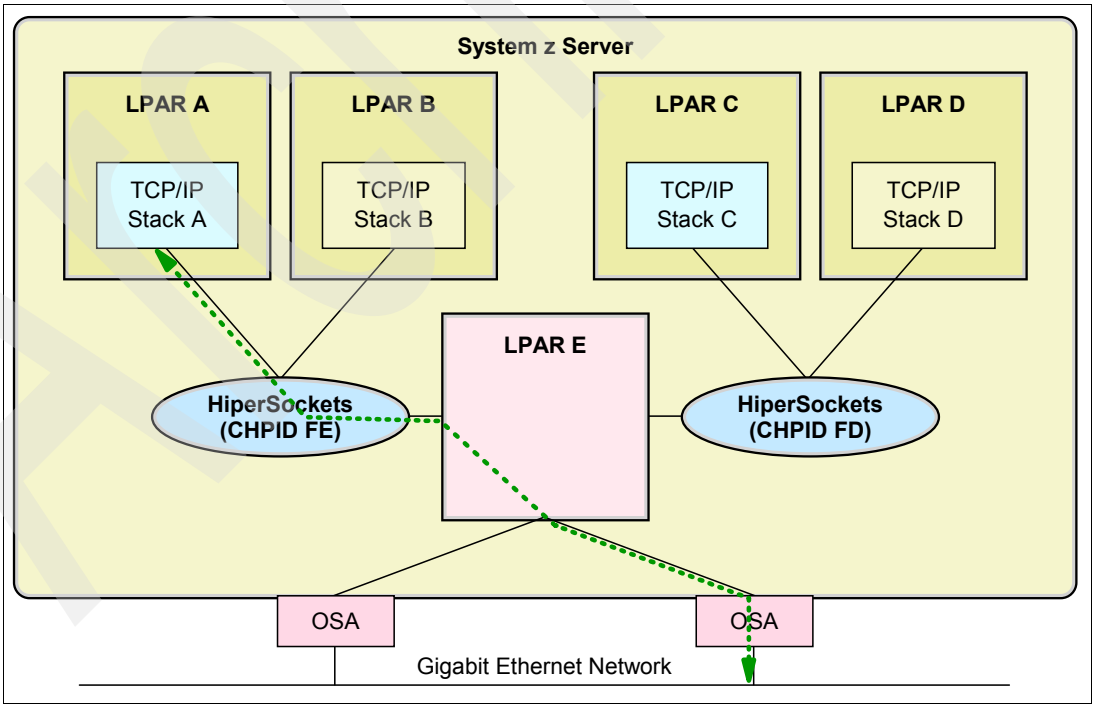


Figure 4-4 HiperSockets Accelerator

Note: This example is intended purely to demonstrate IP traffic flow. We do not recommend implementing HiperSockets Accelerator using a single LPAR.

Detailed information on the subject of HiperSockets is available in *HiperSockets Implementation Guide*, SG24-6816.

4.2.3 Dynamic XCF

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.

From an IP topology perspective, DYNAMICXCF establishes fully meshed IP connectivity to all other z/OS TCP/IP stacks in the sysplex. You 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 V1R9 TCP/IP Implementation Volume 3: High Availability, Scalability, and Performance*, SG24-7534).

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)

Dynamic XCF support is illustrated in Figure 4-5 on page 111.

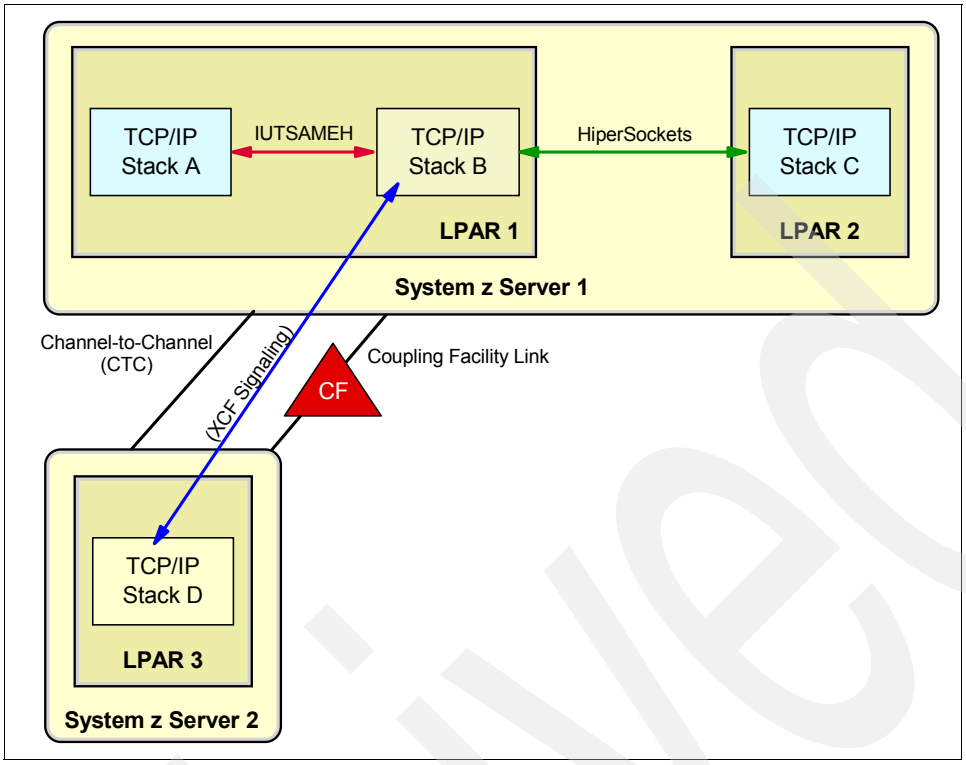


Figure 4-5 Dynamic XCF support

HiperSockets DYNAMICXCF connectivity

z/OS images within the same server with DYNAMICXCF coded will use HiperSockets DYNAMICXCF connectivity instead of 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 least, 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.

For additional information about dynamic XCF, Sysplex Distributor, and nondisruptive dynamic VIPA movement refer to *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 3: High Availability, Scalability, and Performance*, SG24-7534.

4.3 Connectivity for the z/OS environment

The subsequent sections focus on the interface implementation only, which means establishing Layer 2 and a subset of Layer 3 (IP addressing) connectivity. For details beyond the basic implementation of the immediate LAN environment, also refer to:

- Chapter 5, “Routing” on page 141, for IP routing details
- Chapter 6, “Virtual Medium Access Control (VMAC) support” on page 195 for use of virtual MAC addresses
- Chapter 7, “Sysplex subplexing” on page 203 for isolating TCP/IP stack in a sysplex

To design connectivity in a z/OS environment you must take the following considerations into account:

- As a server environment, 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 standalone 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, in general we 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, in the following sections we present best practice scenarios for building a z/OS Communications Server TCP/IP configuration, using OSA-Express (QDIO), HiperSockets (iQDIO), and dynamic XCF.

We build our connectivity scenarios with two OSA-Express 1000BASE-T features (two ports each) that are connected to the LAN environment (two Cisco 6500 switches). We also implement a HiperSockets internal LAN to interconnect all LPARs within the same System z9. And, finally, we use dynamic XCF connectivity for the sysplex environment.

The scenarios we discuss are as follows:

- "Configuring OSA-Express with VLAN ID" on page 116
- "Configuring HiperSockets" on page 123
- "Configuring DYNAMICXCF" on page 128

Note that in this chapter we are only defining our LPARs as end points.

For a complete picture of our implementation environment, refer to Appendix D, "Our implementation environment" on page 315.

IOCP definitions

Example 4-1 is an excerpt of the IOCP statements we used in our System z environment (only showing OSA-Express CHPID 02 and HiperSockets CHPID F4). These statements are required by the input/output subsystem and the operating system. Because all of our OSA-Express and HiperSockets connectivity will be used across all four LPARs, we defined the CHPIDs as shared.

Example 4-1 IOCP statements

```
ID    MSG2='SYS6.IODF09 - 2007-09-05 08:41',SYSTEM=(2094,1),  *
      TOK=('SCZP101',00800006991E2094084154100107248F00000000,*
      00000000,'07-09-05','08:41:54','SYS6','IODF09')
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,*
      65535))
CHPID PATH=(CSS(0,1,2),02),SHARED,                            *
      PARTITION=((CSS(0),(A08,A09,A0A),(=)),(CSS(1),(A18,A19,A*
      1A),(=)),(CSS(2),(A23,A24,A25,A26,A28,A29,A2A),(=))),   *
      PCHID=390,TYPE=OSD
CHPID PATH=(CSS(1,2),F4),SHARED,                              *
      PARTITION=((CSS(1),(A12),(=)),(CSS(2),(A23,A24,A25,A29),*
      (=))),TYPE=IQD

CNTLUNIT CUNUMBR=2080,                                        *
      PATH=((CSS(0),02),(CSS(1),02),(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(1),F4),(CSS(2),F4)),UNIT=IQD
IODEVICE ADDRESS=(E800,032),CUNUMBR=(E800),UNIT=IQD
```

VTAM definitions

Before getting started with configuring the scenarios in the following sections, it is important to understand the role of VTAM in the TCP/IP configuration.

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 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 configuration and control of the MultiPath Channel (MPC) interfaces are provided by VTAM. They are enabled in VTAM as TRLE minor nodes.

You must define 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 HiperSockets, the TRLE is dynamically generated by VTAM.

For details about defining a TRLE, refer to *z/OS Communications Server: SNA Resource Definition*, SC31-8778.

4.4 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. Therefore, we must have at least two OSA-Express features connecting to two different switches in the network.

Because 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 this 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 four LPARs (SC30, SC31, SC32 and SC33); see Figure 4-6 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.

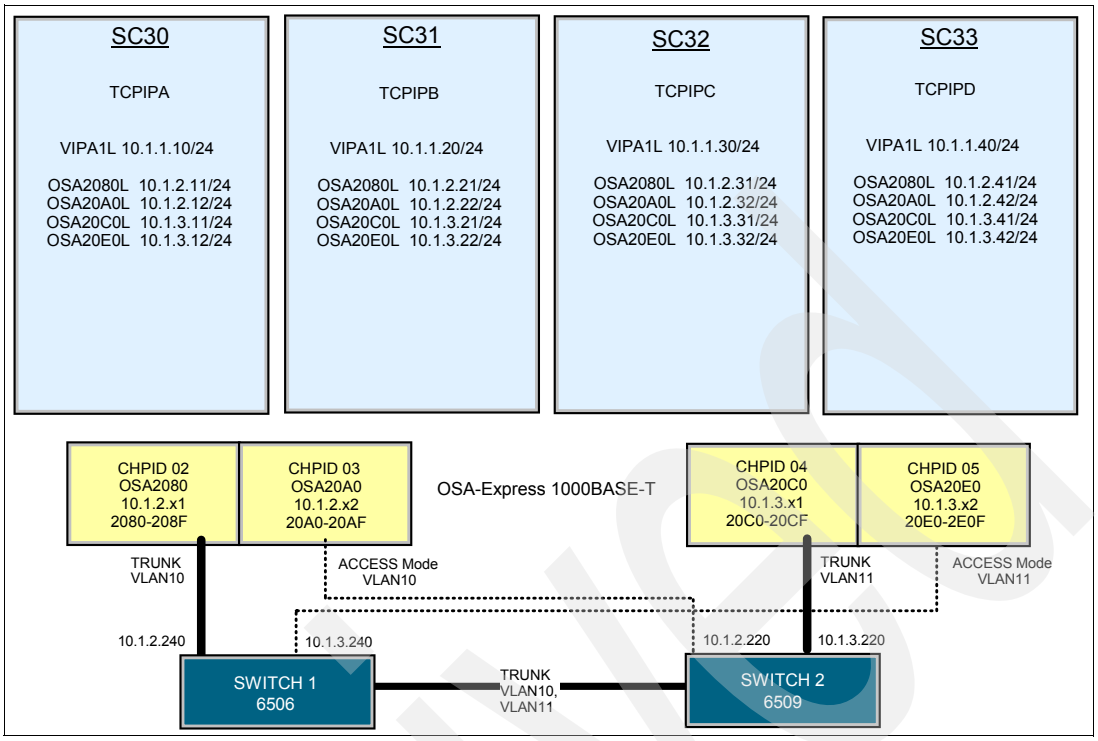


Figure 4-6 OSA-Express (QDIO) implementation

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 113).
- To define an OSA-Express port in QDIO mode, we 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.

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. Therefore, 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 106.

4.4.1 Configuring OSA-Express with VLAN ID

To implement OSA-Express (QDIO) in our environment, we performed 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. Define the characteristics of each LINK statement using BSDROUTINGPARMS.

We explain these tasks in more detail in the following sections.

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 the following information:

- The switch ports to which the OSA-Express ports are connected.

  Table 4-3 shows the OSA-Express and switch port assignment with VLAN IDs and mode type in our configuration.

Table 4-3 OSA-Express and switch port assignment with VLAN IDs

OSA-Express port	Connects to switch	Switch port	VLAN ID (mode)
CHPID 02 (2080)	Switch 1	Interface GIGA 1/41	10 (Trunk mode)
CHPID 03 (20A0)	Switch 2	interface GIGA 2/3	10 (Access mode)
CHPID 04 (20C0)	Switch 2	Interface GIGA 2/7	11 (Trunk mode)
CHPID 05 (20E0)	Switch 1	Interface GIGA 1/35	11 (Access mode)

- The IP subnetwork and mask.

  We used the following:

  – Subnetwork 10.1.2.0, mask 255.255.255.0 for VLAN 10
  – Subnetwork 10.1.3.0, 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 1 port 1/41

```
interface GigabitEthernet1/41
 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=(2082-2088),                                  *
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
;TRL MAJ NODE: OSA2080,OSA20a0,OSA20c0,and OSA20e0
DEVICE OSA2080    MPCIPA  1
LINK   OSA2080L   IPAQENET 2 OSA2080 VLANID 10 3
DEVICE OSA20C0    MPCIPA
LINK   OSA20C0L   IPAQENET    OSA20C0 VLANID 11
DEVICE OSA20E0    MPCIPA
LINK   OSA20E0L   IPAQENET    OSA20E0
DEVICE OSA20A0    MPCIPA
LINK   OSA20A0L   IPAQENET    OSA20A0
```

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.

Example 4-5 OSA-Express HOME addresses

```
HOME
   10.1.2.11     OSA2080L
   10.1.3.11     OSA20C0L
   10.1.3.12     OSA20E0L
   10.1.2.12     OSA20A0L
```

Define the characteristics of each LINK statement using BSDROUTINGPARMS

To define the link characteristics, such as MTU size 1 and subnet mask 2, we used the BSDROUTINGPARMS statements (see Example 4-6 on page 118).

If not supplied, defaults will be used from static routing definitions in BEGINROUTES or the OMPROUTE configuration (dynamic routing definitions), if implemented.

If the link characteristics, BEGINROUTES statements, or the OMPROUTE configuration are not defined, then the stack's interface layer (based on hardware capabilities) and the characteristics of devices and links are used. This, however, may not provide the performance or function desired.

Example 4-6 BSDRoutingparms statements

```
BSDROUTINGPARMS TRUE
;  Link name      MTU    Cost metric  Subnet Mask    Dest address
          VIPA1L 1492 1        0       255.255.255.252   0
        OSA2080L 1492          0       255.255.255.0   2 0
        OSA20A0L 1492          0       255.255.255.0     0
        OSA20C0L 1492          0       255.255.255.0     0
        OSA20E0L 1492          0       255.255.255.0     0
ENDBSDROUTINGPARMS
```

Note that static and dynamic routing definitions will override or replace the link characteristics defined through the BSDROUTINGPARMS statements. Refer to Chapter 5, “Routing” on page 141 for more information about static and dynamic routing.

4.4.2 Verifying the connectivity status

In this section, we verify the status of the OSA devices defined to the TCP/IP stack and 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 4-18 on page 128.

Example 4-7 Using command D TCPIP,TCPIPA,N,DEV to verify the Device status

```
D TCPIP,TCPIPA,N,DEV
EZD0101I NETSTAT CS V1R9 TCPIPA 597
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
DEVNAME: OSA2080           DEVTYPE: MPCIPA
  DEVSTATUS: READY
  LNKNAME: OSA2080L          LNKTYPE: IPAQENET   LNKSTATUS: READY
    NETNUM: N/A  QUESIZE: N/A  SPEED: 0000001000
    IPBROADCASTCAPABILITY: NO
    CFGROUTER: NON                 ACTROUTER: NON
    ARPOFFLOAD: YES                ARPOFFLOADINFO: YES
    ACTMTU: 8992
    VLANID: 10                     VLANPRIORITY: DISABLED
    DYNVLANREGCFG: NO              DYNVLANREGCAP: YES
    READSTORAGE: GLOBAL (4096K)    INBPERF: BALANCED
    CHECKSUMOFFLOAD: YES
    SECCLASS: 255                  MONSYSPLEX: NO
  BSD ROUTING PARAMETERS:
    MTU SIZE: 1492             METRIC: 100
    DESTADDR: 0.0.0.0          SUBNETMASK: 255.255.255.0
  MULTICAST SPECIFIC:
    MULTICAST CAPABILITY: YES
    GROUP            REFCNT        SRCFLTMD
    -----            ------        --------
    224.0.0.1        0000000001    EXCLUDE
      SRCADDR: NONE
  LINK STATISTICS:
```

```
BYTESIN                          = 352
INBOUND PACKETS                  = 4
INBOUND PACKETS IN ERROR         = 0
INBOUND PACKETS DISCARDED        = 0
INBOUND PACKETS WITH NO PROTOCOL = 0
BYTESOUT                         = 8416
OUTBOUND PACKETS                 = 92
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.

Because the device driver resources are provided by VTAM, you have the ability to display the resources using VTAM display commands.

To display a list of all TRLEs active in VTAM, use the command D NET,TRL, as shown in Example 4-8.

Example 4-8 D NET,TRL command output

```
D NET,TRL
IST097I DISPLAY ACCEPTED
IST350I DISPLAY TYPE = TRL 605
IST924I -------------------------------------------------------------
IST1954I TRL MAJOR NODE = ISTTRL
IST1314I TRLE = ISTT3032  STATUS = ACTIV       CONTROL = XCF
IST1314I TRLE = ISTT3031  STATUS = ACTIV       CONTROL = XCF
IST1314I TRLE = ISTT3033  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 = IUTIQDIO  STATUS = ACTIV       CONTROL = MPC
IST1314I TRLE = IUTSAMEH  STATUS = ACTIV       CONTROL = MPC
IST1454I 8 TRLE(S) DISPLAYED
IST924I -------------------------------------------------------------
IST1954I TRL MAJOR NODE = TRLTONET
IST1314I TRLE = MPCNET    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
```

You can also display information of TRLEs grouped by control type, such as MPC or XCF devices, as shown in Example 4-20 on page 130.

Example 4-9 D NET,TRL,CONTROL=MPC

```
D NET,TRL,CONTROL=MPC
IST097I DISPLAY ACCEPTED
IST350I DISPLAY TYPE = TRL 609
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 = TRLTONET
IST1314I TRLE = MPCNET    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 4-10, for an OSA-Express device.

Example 4-10 D NET,TRL,TRLE=OSA2080P

```
D NET,TRL,TRLE=OSA2080P
IST097I DISPLAY ACCEPTED
IST075I NAME = OSA2080P, TYPE = TRLE 613
IST1954I TRL MAJOR NODE = OSA2080
IST486I STATUS= ACTIV, DESIRED STATE= ACTIV
IST087I TYPE = LEASED            , CONTROL = MPC , HPDT = YES
IST1715I MPCLEVEL = QDIO       MPCUSAGE = SHARE
IST1716I PORTNAME = OSA2080    LINKNUM =   0   OSA CODE LEVEL = 087A
```

```
IST1577I HEADER SIZE = 4096 DATA SIZE = 0 STORAGE = ***NA***
IST1221I WRITE DEV = 2081 STATUS = ACTIVE     STATE = ONLINE
IST1577I HEADER SIZE = 4092 DATA SIZE = 0 STORAGE = ***NA***
IST1221I READ  DEV = 2080 STATUS = ACTIVE     STATE = ONLINE
IST1221I DATA  DEV = 2082 STATUS = ACTIVE     STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1717I ULPID = TCPIPA
IST1815I IQDIO ROUTING DISABLED
IST1918I READ STORAGE = 4.0M(64 SBALS)
IST1757I PRIORITY1: UNCONGESTED PRIORITY2: UNCONGESTED
IST1757I PRIORITY3: UNCONGESTED PRIORITY4: UNCONGESTED
IST2190I DEVICEID PARAMETER FOR OSAENTA TRACE COMMAND = 02-03-00-02
IST1801I UNITS OF WORK FOR NCB AT ADDRESS X'131B3010'
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 = 3
IST1221I DATA  DEV = 2083 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1221I DATA  DEV = 2084 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1221I DATA  DEV = 2085 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1221I DATA  DEV = 2086 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1221I DATA  DEV = 2087 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1221I DATA  DEV = 2088 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST314I END
```

4.5 HiperSockets connectivity

HiperSockets provides very fast TCP/IP communications between different logical partitions (LPARs) through the system memory of the System z server. The LPARs that are connected this way 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 environment we use three IQD CHPIDs (F4, F5, and F6). Each will create a separate logical LAN with its own subnetwork. Figure 4-7 depicts these interfaces to our scenario.

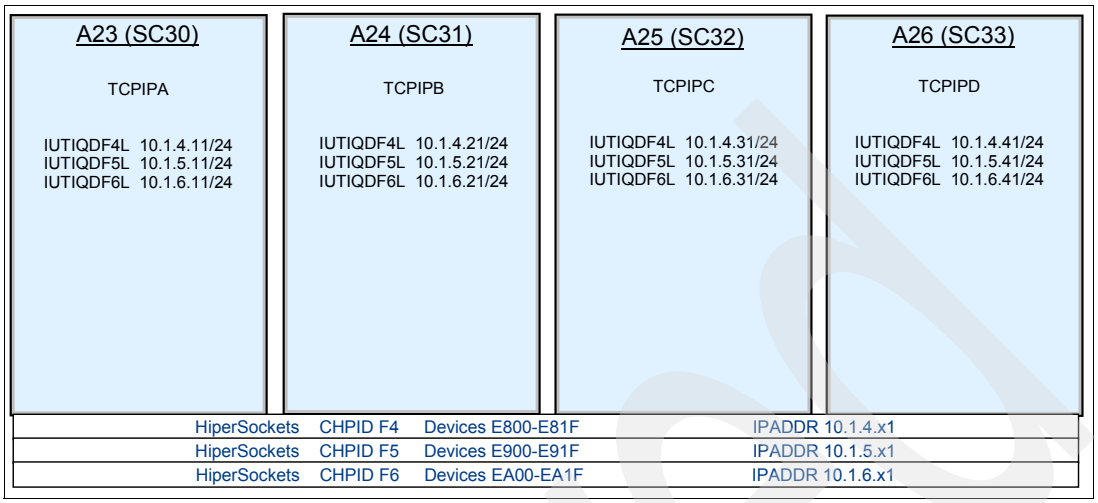


Figure 4-7 HiperSockets implementation scenario

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 113).
- 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, the 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 Communications Server: IP Configuration Reference*, SC31-8776.

Considerations

For isolation of IP traffic between LPARs via HiperSockets, consider using VLANs, which means that you can logically subdivide the internal LAN for a HiperSockets CHPID into multiple virtual LANs. Therefore, stacks that configure the same VLAN ID for the same CHPID can communicate over that given HiperSockets, while stacks that have no VLAN ID or a different VLAN ID configured cannot.

For HiperSockets, the VLAN ID applies to IPv4 and IPv6 connections. HiperSockets VLAN IDs can be defined using the VLANID parameter on a LINK or INTERFACE statement. Valid VLAN IDs are in the range of 1 to 4094.

4.5.1 Configuring HiperSockets

The steps to implement HiperSockets are basically the same as with an OSA-Express interface. What changes is that there is no external configuration to be done, and the TRLE is created dynamically by VTAM.

The steps in the TCP/IP profile are as follows:

1. Create a DEVICE and LINK statements for each HiperSockets CHPID.
2. Create a HOME address to each defined LINK.
3. Define the characteristics of each LINK statement using BSDROUTINGPARMS.

Create a DEVICE and LINK statements for each HiperSockets CHPID

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.

Define the device and link statements for each HiperSockets CHPID being implemented, as shown in Example 4-11. 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 4-11 HiperSockets device and link definitions

```
;HiperSockets definition. The TRLE is dynamically created on VTAMs
DEVICE IUTIQDF4 MPCIPA    1
LINK   IUTIQDF4L IPAQIDIO  2   IUTIQDF4
DEVICE IUTIQDF5 MPCIPA    1
LINK   IUTIQDF5L IPAQIDIO  2   IUTIQDF5
DEVICE IUTIQDF6 MPCIPA    1
LINK   IUTIQDF6L IPAQIDIO  2   IUTIQDF6
```

Important: The hexadecimal value specified here represents the CHPID, and it cannot be the same value as 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 4-12.

Example 4-12 HiperSockets HOME addresses

```
HOME
   10.1.4.11     IUTIQDF4L
   10.1.5.11     IUTIQDF5L
   10.1.6.11     IUTIQDF6L
```

Define the characteristics of each LINK statement using BSDROUTINGPARMS

To define the link characteristics, such as MTU size (**1**) and subnet mask (**2**), we used the BSDROUTINGPARMS statements (see Example 4-13 on page 124). If not supplied, defaults will be used from static routing definitions in BEGINROUTES or the OMPROUTE configuration (dynamic routing definitions), if implemented.

If the link characteristics, BEGINROUTES statements, or the OMPROUTE configuration are not defined, then the stack's interface layer (based on hardware capabilities) and the

characteristics of devices and links are used. This, however, may not provide the performance or function desired.

Example 4-13 BSDRoutingparms statements

```
BSDROUTINGPARMS TRUE
;  Link name      MTU    Cost metric  Subnet Mask    Dest address
       VIPA1L 1492            0       255.255.255.252   0
       OSA2080L 1492          0       255.255.255.0     0
       OSA20A0L 1492          0       255.255.255.0     0
       OSA20C0L 1492          0       255.255.255.0     0
       OSA20E0L 1492          0       255.255.255.0     0
       IUTIQDF4L 8192 1       0       255.255.255.0 2   0
       IUTIQDF5L 8192         0       255.255.255.0     0
       IUTIQDF6L 8192         0       255.255.255.0     0
ENDBSDROUTINGPARMS
```

Note that static and dynamic routing definitions will override or replace the link characteristics defined through the BSDROUTINGPARMS statements. Refer to Chapter 5, “Routing” on page 141 for more information on static and dynamic routing.

4.5.2 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 shown in Example 4-14.

Example 4-14 Using command D TCPIP,TCPIPA,N,DEV to verify the HiperSockets connection

```
DEVNAME: IUTIQDF4          DEVTYPE: MPCIPA
   DEVSTATUS: READY
   LNKNAME: IUTIQDF6L          LNKTYPE: IPAQIDIO   LNKSTATUS: READY
     NETNUM: N/A  QUESIZE: N/A
     IPBROADCASTCAPABILITY: NO
     CFGROUTER: NON                  ACTROUTER: NON
     ARPOFFLOAD: YES                 ARPOFFLOADINFO: YES
     ACTMTU: 8192
     READSTORAGE: GLOBAL (2048K)
     SECCLASS: 255                   MONSYSPLEX: NO
   BSD ROUTING PARAMETERS:
     MTU SIZE: 8192              METRIC: 90
     DESTADDR: 0.0.0.0           SUBNETMASK: 255.255.255.0
   MULTICAST SPECIFIC:
     MULTICAST CAPABILITY: YES
     GROUP             REFCNT        SRCFLTMD
     -----             ------        --------
     224.0.0.6         0000000001    EXCLUDE
       SRCADDR: NONE
     224.0.0.5         0000000001    EXCLUDE
       SRCADDR: NONE
     224.0.0.1         0000000001    EXCLUDE
       SRCADDR: NONE
       SRCADDR: NONE
    LINK STATISTICS:
      BYTESIN                          = 34931365
      INBOUND PACKETS                  = 311531
      INBOUND PACKETS IN ERROR         = 0
```

```
INBOUND PACKETS DISCARDED         = 0
INBOUND PACKETS WITH NO PROTOCOL  = 0
BYTESOUT                          = 16987569
OUTBOUND PACKETS                  = 154897
OUTBOUND PACKETS IN ERROR         = 15
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.

Because 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*, as explained here:

- *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.

To display a list of all TRLEs active in VTAM, use the command D NET,TRL, as shown in Example 4-15.

Example 4-15 D NET,TRL command output

```
D NET,TRL
IST097I DISPLAY ACCEPTED
IST350I DISPLAY TYPE = TRL 605
IST924I -------------------------------------------------------------
IST1954I TRL MAJOR NODE = ISTTRL
IST1314I TRLE = ISTT3032  STATUS = ACTIV       CONTROL = XCF
IST1314I TRLE = ISTT3031  STATUS = ACTIV       CONTROL = XCF
IST1314I TRLE = ISTT3033  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 = IUTIQDIO  STATUS = ACTIV       CONTROL = MPC
IST1314I TRLE = IUTSAMEH  STATUS = ACTIV       CONTROL = MPC
IST1454I 8 TRLE(S) DISPLAYED
IST924I -------------------------------------------------------------
```

The **D NET,TRL,TRLE** command used to obtain information about a HiperSockets device is shown in Example 4-16.

Example 4-16 D NET,TRL,TRLE=IUTIQDF6

```
D NET,TRL,TRLE=IUTIQDF6
IST097I DISPLAY ACCEPTED
IST075I NAME = IUTIQDF6, TYPE = TRLE 620
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 = *NA*
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'0EB5B010'
IST1802I P1 CURRENT = 0 AVERAGE = 1 MAXIMUM = 1
IST1802I P2 CURRENT = 0 AVERAGE = 1 MAXIMUM = 1
IST1802I P3 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
IST1802I P4 CURRENT = 0 AVERAGE = 1 MAXIMUM = 2
IST1221I DATA  DEV = EA03 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1221I DATA  DEV = EA04 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1221I DATA  DEV = EA05 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1221I DATA  DEV = EA06 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1221I DATA  DEV = EA07 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1221I DATA  DEV = EA08 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1221I DATA  DEV = EA09 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST314I END
```

4.6 Dynamic XCF connectivity

The last connectivity scenario we are going to add to our 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.

After being 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 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 shown in Figure 4-8.

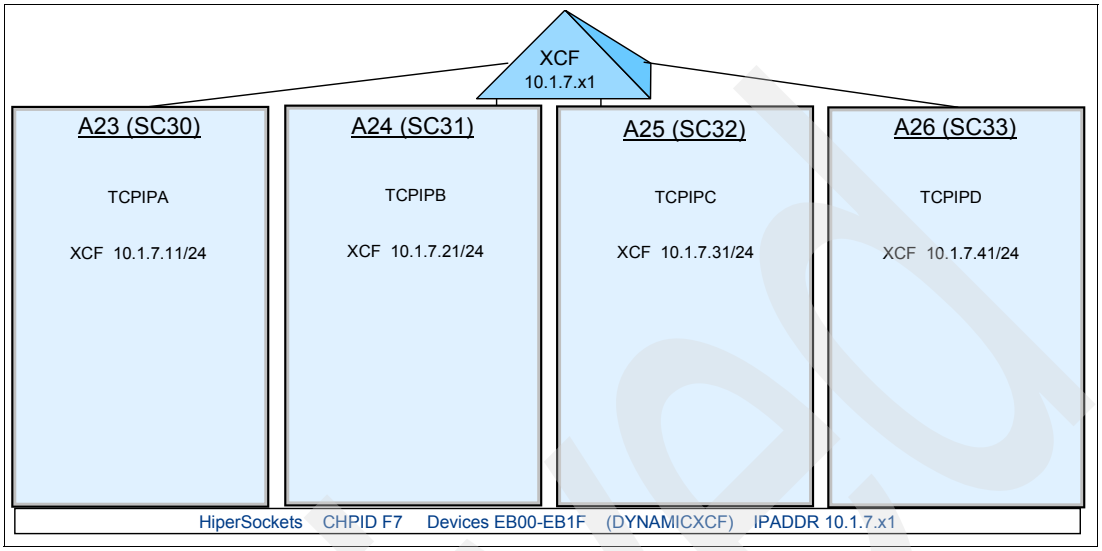


Figure 4-8 Dynamic XCF environment

Dependencies

The dependencies are as follows:

- 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 Communications Server: SNA Network Implementation*, SC31-8777.
- 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 option, IQDCHPID, can be used to identify which IQD CHPID will be used by DYNAMICXCF.

Considerations

Communications Server 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. It allows for the use of OSA-Express features such as 1000BASE-T Ethernet, Gigabit Ethernet and 10 Gigabit Ethernet. For details refer to *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 3: High Availability, Scalability, and Performance*, SG24-7534.

Communications Server supports sysplex subplexing in conjunction with HiperSockets and DYNAMICXCF. Refer to Chapter 7, “Sysplex subplexing” on page 203, for details about restricting data traffic flow among certain TCP/IP stacks in a sysplex environment.

4.6.1 Configuring DYNAMICXCF

To implement XCF connections in our 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 4-8 on page 127 shows our DynamicXCF implementation, using HiperSockets CHPID F7.

When you use dynamic XCF for sysplex configuration, make sure that XCFINIT=YES or XCFINIT=DEFINE is coded in the VTAM start options.

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.

To implement DYNAMICXCF in our environment, we coded the IPCONFIG definitions in the TCP/IP profile, as shown in Example 4-17. 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 4-17 IPCONFIG DYNAMICXCF configuration

```
IPCONFIG DATAGRAMFWD SYSPLEXROUTING IPSECURITY
DYNAMICXCF 10.1.7.11 255.255.255.0 1 1
```

4.6.2 Verifying 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 the 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 4-18.

Example 4-18 Using command D TCPIP,TCPIPA,N,DEV to verify the device status

```
D TCPIP,TCPIPA,N,DEV
EZD0101I NETSTAT CS V1R9 TCPIPA 597
DEVNAME: IUTSAMEH          DEVTYPE: MPCPTP
  DEVSTATUS: SENT SETUP
  LNKNAME: EZASAMEMVS        LNKTYPE: MPCPTP     LNKSTATUS: NOT ACTIVE
    NETNUM: N/A  QUESIZE: N/A
    ACTMTU: UNKNOWN
    SECCLASS: 255
```

```
BSD ROUTING PARAMETERS:
  MTU SIZE: 65535             METRIC: 01
  DESTADDR: 0.0.0.0           SUBNETMASK: 255.255.255.0
MULTICAST SPECIFIC:
  MULTICAST CAPABILITY: YES
  GROUP             REFCNT        SRCFLTMD
  -----             ------        --------
  224.0.0.6         0000000001    EXCLUDE
    SRCADDR: NONE
  224.0.0.5         0000000001    EXCLUDE
    SRCADDR: NONE
  224.0.0.1         0000000001    EXCLUDE
    SRCADDR: NONE
LINK 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       = 8
DEVNAME: IUTIQDIO          DEVTYPE: MPCIPA
  DEVSTATUS: READY
  LNKNAME: IQDIOLNK0A01070B  LNKTYPE: IPAQIDIO   LNKSTATUS: READY
    NETNUM: N/A  QUESIZE: N/A
    IPBROADCASTCAPABILITY: NO
    CFGROUTER: NON                 ACTROUTER: NON
    ARPOFFLOAD: YES                ARPOFFLOADINFO: YES
    ACTMTU: 8192
    READSTORAGE: GLOBAL (2048K)
    SECCLASS: 255
  BSD ROUTING PARAMETERS:
    MTU SIZE: 65535            METRIC: 90
    DESTADDR: 0.0.0.0          SUBNETMASK: 255.255.255.0
  MULTICAST SPECIFIC:
    MULTICAST CAPABILITY: YES
    GROUP             REFCNT        SRCFLTMD
    -----             ------        --------
    224.0.0.6         0000000001    EXCLUDE
      SRCADDR: NONE
    224.0.0.5         0000000001    EXCLUDE
   SRCADDR: NONE
   224.0.0.1         0000000001    EXCLUDE
     SRCADDR: NONE
 LINK STATISTICS:
   BYTESIN                         = 34013600
   INBOUND PACKETS                 = 309605
   INBOUND PACKETS IN ERROR        = 0
   INBOUND PACKETS DISCARDED       = 0
   INBOUND PACKETS WITH NO PROTOCOL = 0
   BYTESOUT                        = 16502456
   OUTBOUND PACKETS                = 151347
   OUTBOUND PACKETS IN ERROR       = 11
   OUTBOUND PACKETS DISCARDED      = 0
```

Note that device IUTIQDIO has a dynamically defined link name of IQDIOLNK0A01070B. In the link name, 0A01070B is the hexadecimal value of the assigned IP address (10.1.7.11).

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.

Because 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*, as explained here:

- *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 4-19.

Example 4-19 D NET,TRL command output

```
D NET,TRL
IST097I DISPLAY ACCEPTED
IST350I DISPLAY TYPE = TRL 605
IST924I -------------------------------------------------------------
IST1954I TRL MAJOR NODE = ISTTRL
IST1314I TRLE = ISTT3032  STATUS = ACTIV       CONTROL = XCF
IST1314I TRLE = ISTT3031  STATUS = ACTIV       CONTROL = XCF
IST1314I TRLE = ISTT3033  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 = IUTIQDIO  STATUS = ACTIV       CONTROL = MPC
IST1314I TRLE = IUTSAMEH  STATUS = ACTIV       CONTROL = MPC
IST1454I 8 TRLE(S) DISPLAYED
IST924I -------------------------------------------------------------
IST314I END
```

You can display information of TRLEs grouped by control type, such as MPC or XCF devices, as shown in Example 4-20.

Example 4-20 D NET,TRL,CONTROL=XCF

```
D NET,TRL,CONTROL=XCF
IST097I DISPLAY ACCEPTED
IST350I DISPLAY TYPE = TRL 129
IST924I -------------------------------------------------------------
```

```
IST1954I TRL MAJOR NODE = ISTTRL
IST1314I TRLE = ISTT3032  STATUS = ACTIV       CONTROL = XCF
IST1314I TRLE = ISTT3031  STATUS = ACTIV       CONTROL = XCF
IST1314I TRLE = ISTT3033  STATUS = ACTIV       CONTROL = XCF
IST1454I 3 TRLE(S) DISPLAYED
IST924I -------------------------------------------------------------
```

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 624
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 = 0200002F00200002     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 628
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 = *NA*
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'13143010'
IST1802I P1 CURRENT = 0 AVERAGE = 1 MAXIMUM = 1
IST1802I P2 CURRENT = 0 AVERAGE = 1 MAXIMUM = 1
IST1802I P3 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
IST1802I P4 CURRENT = 1 AVERAGE = 1 MAXIMUM = 3
IST1221I DATA  DEV = EB03 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1221I DATA  DEV = EB04 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1221I DATA  DEV = EB05 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1221I DATA  DEV = EB06 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
```

```
IST1221I DATA  DEV = EB07 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1221I DATA  DEV = EB08 STATUS = RESET      STATE = N/A
IST1724I I/O TRACE = OFF  TRACE LENGTH = *NA*
IST1221I DATA  DEV = EB09 STATUS = RESET      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 634
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
```

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 4-24.

Example 4-24 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 EZBTCPCS
EZZ4324I CONNECTION TO 10.1.7.51 ACTIVE FOR DEVICE IUTSAMEH 1
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE IUTSAMEH
EZZ4324I CONNECTION TO 10.1.7.31 ACTIVE FOR DEVICE IUTSAMEH 1
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE IUTIQDIO 2
```

1 This message indicates that the TCPIPA stack has been connected to the other stacks through XCF using a SAMEHOST device.

2 This message indicates that XCF will also use HiperSockets to connect other TCP/IP stacks within the same server, using a IUTIQDIO device.

4.7 Controlling and activating devices

After all required connectivity definitions are defined in the TCP/IP profile and the stack is started, you have the option to start and stop devices, as well as activate modified device definitions. In this section we show the commands used to perform these tasks.

4.7.1 Starting a device

A device can be started by any of the following methods:

- Defining the START statement in the TCP/IP profile, as shown in Example 4-25 on page 133.

Example 4-25 Start statements in TCP/IP profile

```
START OSA2080
START OSA20C0
START OSA20E0
START OSA20A0
START IUTIQDF4
START IUTIQDF5
START IUTIQDF6
```

- 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 4-26.

Example 4-26 Starting a TCP/IP device

```
V TCPIP,TCPIPA,START,OSA2080
EZZ0060I PROCESSING COMMAND: VARY TCPIP,TCPIPA,START,OSA2080
EZZ0053I COMMAND VARY START COMPLETED SUCCESSFULLY
EZZ4313I INITIALIZATION COMPLETE FOR DEVICE OSA2080
```

4.7.2 Stopping a device

A device can be stopped by any of the following methods:

- 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 4-27.

Example 4-27 Stop command resulting messages

```
V TCPIP,TCPIPA,STOP,OSA2080
EZZ0060I PROCESSING COMMAND: VARY TCPIP,TCPIPA,STOP,OSA2080
EZZ0053I COMMAND VARY STOP COMPLETED SUCCESSFULLY
EZZ4329I LINK OSA20A0L HAS TAKEN OVER ARP RESPONSIBILITY FOR INACTIVE LINK OSA2080L
EZZ4315I DEACTIVATION COMPLETE FOR DEVICE OSA2080
```

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 4-27 indicates this action.

4.7.3 Activating modified device definitions

You can activate modified device definitions 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 4-28 on page 134.

Example 4-28 OBEYFILE example

```
;Original BSDROUTINGPARMS statement for link OSA2080

;BSDROUTINGPARMS TRUE
;  Link name      MTU    Cost metric  Subnet Mask    Dest address
;OSA2080L        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
OSA2080L         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.

4.8 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 for determining 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 the PING command as a debugging tool

PING command (direct network)	PING response	Possible cause and actions
ping 10.1.2.11 (intf osa2080l)	CS V1R9: Pinging host 10.1.2.11 sendMessage(): EDC8130I Host cannot be reached.	The interface being tested has a problem. Use the **Netstat** command to verify the interface status.

PING command (direct network)	PING response	Possible cause and actions
ping 10.1.2.11 (intf osa2080l	CS V1R9: Pinging host 10.1.2.11 Ping #1 timed out.	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 10.1.2.11 (intf osa2080l	CS V1R9: Pinging host 10.1.2.11 Ping #1 response took 0.000 seconds.	This is the expected response. The interface is working.

- **Netstat**

  You can use the **Netstat** command to verify the 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**

    This displays all defined interfaces and their IP addresses, even those interfaces dynamically created, as shown in Example 4-29.

Example 4-29 NETSTAT HOME command results

```
D TCPIP,TCPIPA,N,HOME
EZD0101I NETSTAT CS V1R9 TCPIPA
HOME ADDRESS LIST:
LINKNAME:   VIPA3L
  ADDRESS:  10.1.30.10
    FLAGS:
LINKNAME:   VIPA1L
  ADDRESS:  10.1.1.10
    FLAGS:  PRIMARY
LINKNAME:   VIPA2L
  ADDRESS:  10.1.2.10
    FLAGS:
LINKNAME:   OSA2080L
  ADDRESS:  10.1.2.11
    FLAGS:
LINKNAME:   OSA20C0L
  ADDRESS:  10.1.3.11
    FLAGS:
LINKNAME:   OSA20E0L
  ADDRESS:  10.1.3.12
    FLAGS:
LINKNAME:   OSA20A0L
  ADDRESS:  10.1.2.12
    FLAGS:
LINKNAME:   IUTIQDF4L
  ADDRESS:  10.1.4.11
    FLAGS:
LINKNAME:   IUTIQDF5L
  ADDRESS:  10.1.5.11
    FLAGS:
LINKNAME:   IUTIQDF6L
```

```
  ADDRESS:  10.1.6.11
    FLAGS:
LINKNAME:   EZASAMEMVS
  ADDRESS:  10.1.7.11
    FLAGS:
LINKNAME:   IQDIOLNK0A01070B
  ADDRESS:  10.1.7.11
    FLAGS:
LINKNAME:   VIPL0A01080A
  ADDRESS:  10.1.8.10
    FLAGS:
LINKNAME:   VIPL0A010814
  ADDRESS:  10.1.8.20
    FLAGS:  INTERNAL
LINKNAME:   LOOPBACK
  ADDRESS:  127.0.0.1
    FLAGS:
INTFNAME:   LOOPBACK6
  ADDRESS:  ::1
    TYPE:   LOOPBACK
    FLAGS:
16 OF 16 RECORDS DISPLAYED
END OF THE REPORT
```

- **Netstat DEVLINKS/-d**

  This displays the status of each interface, physical and logical, that is defined in the TCP/IP stack, as illustrated in Example 4-30 (only one interface shown as a sample).

Example 4-30 NETSTAT DEVLINKS command results

```
DISPLAY TCPIP,TCPIPA,N,DEV,INTFN=OSA2080L
EZD0101I NETSTAT CS V1R9 TCPIPA 687
DEVNAME: OSA2080           DEVTYPE: MPCIPA
  DEVSTATUS: READY
  LNKNAME: OSA2080L          LNKTYPE: IPAQENET   LNKSTATUS: READY
    NETNUM: N/A  QUESIZE: N/A  SPEED: 0000001000
    IPBROADCASTCAPABILITY: NO
    CFGROUTER: NON                 ACTROUTER: NON
    ARPOFFLOAD: YES                ARPOFFLOADINFO: YES
    ACTMTU: 8992
    VLANID: 10                     VLANPRIORITY: DISABLED
    DYNVLANREGCFG: NO              DYNVLANREGCAP: YES
    READSTORAGE: GLOBAL (4096K)    INBPERF: BALANCED
    CHECKSUMOFFLOAD: YES
    SECCLASS: 255                  MONSYSPLEX: NO
  BSD ROUTING PARAMETERS:
    MTU SIZE: 1492             METRIC: 100
    DESTADDR: 0.0.0.0          SUBNETMASK: 255.255.255.0
  MULTICAST SPECIFIC:
    MULTICAST CAPABILITY: YES
    GROUP             REFCNT        SRCFLTMD
    -----             ------        --------
    224.0.0.5         0000000001    EXCLUDE
      SRCADDR: NONE
    224.0.0.1         0000000001    EXCLUDE
      SRCADDR: NONE
  LINK STATISTICS:
    BYTESIN                          = 22976
    INBOUND PACKETS                  = 186
    INBOUND PACKETS IN ERROR         = 0
```

```
    INBOUND PACKETS DISCARDED         = 0
    INBOUND PACKETS WITH NO PROTOCOL  = 0
    BYTESOUT                          = 18164
    OUTBOUND PACKETS                  = 174
    OUTBOUND PACKETS IN ERROR         = 0
    OUTBOUND PACKETS DISCARDED        = 0
IPV4 LAN GROUP SUMMARY
LANGROUP: 00003
  LNKNAME           LNKSTATUS   ARPOWNER          VIPAOWNER
  -------           ---------   --------          ---------
  OSA2080L          ACTIVE      OSA2080L          YES
  OSA20A0L          ACTIVE      OSA20A0L          NO
1 OF 1 RECORDS DISPLAYED
END OF THE REPORT
```

– **Netstat ARP/-R** (for OSA-Express devices)

This is used to query the ARP cache for a given address. Use this command when the remote host does not answer as expected, to check whether 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 4-31.

Example 4-31 D TCPIP,TCPIPA,N,ARP command results

```
DISPLAY TCPIP,TCPIPA,N,ARP
EZD0101I NETSTAT CS V1R9 TCPIPA 690
QUERYING ARP CACHE FOR ADDRESS 10.1.2.41
LINK: OSA2080L          ETHERNET: 00096B1A7490
QUERYING ARP CACHE FOR ADDRESS 10.1.2.42
LINK: OSA2080L          ETHERNET: 00096B1A7490
QUERYING ARP CACHE FOR ADDRESS 10.1.2.21
LINK: OSA2080L          ETHERNET: 00096B1A7490
QUERYING ARP CACHE FOR ADDRESS 10.1.2.22
LINK: OSA2080L          ETHERNET: 00096B1A7490
QUERYING ARP CACHE FOR ADDRESS 10.1.2.31
LINK: OSA2080L          ETHERNET: 00096B1A7490
QUERYING ARP CACHE FOR ADDRESS 10.1.2.32
LINK: OSA2080L          ETHERNET: 00096B1A7490
QUERYING ARP CACHE FOR ADDRESS 10.1.2.51
LINK: OSA2080L          ETHERNET: 00096B1A7490
QUERYING ARP CACHE FOR ADDRESS 10.1.2.52
LINK: OSA2080L          ETHERNET: 00096B1A7490
QUERYING ARP CACHE FOR ADDRESS 10.1.2.50
LINK: OSA2080L          ETHERNET: 00096B1A7490
QUERYING ARP CACHE FOR ADDRESS 10.1.2.11
LINK: OSA2080L          ETHERNET: 00096B1A7490
QUERYING ARP CACHE FOR ADDRESS 10.1.2.240
LINK: OSA2080L          ETHERNET: 0014F1464600
QUERYING ARP CACHE FOR ADDRESS 10.1.2.220
LINK: OSA2080L          ETHERNET: 000785F37302
QUERYING ARP CACHE FOR ADDRESS 10.1.3.41
LINK: OSA20C0L          ETHERNET: 00096B1A7454
QUERYING ARP CACHE FOR ADDRESS 10.1.3.42
LINK: OSA20C0L          ETHERNET: 00096B1A7454
QUERYING ARP CACHE FOR ADDRESS 10.1.3.11
LINK: OSA20C0L          ETHERNET: 00096B1A7454
QUERYING ARP CACHE FOR ADDRESS 10.1.3.21
LINK: OSA20C0L          ETHERNET: 00096B1A7454
QUERYING ARP CACHE FOR ADDRESS 10.1.3.22
LINK: OSA20C0L          ETHERNET: 00096B1A7454
QUERYING ARP CACHE FOR ADDRESS 10.1.3.31
```

```
LINK: OSA20C0L          ETHERNET: 00096B1A7454
QUERYING ARP CACHE FOR ADDRESS 10.1.3.32
LINK: OSA20C0L          ETHERNET: 00096B1A7454
QUERYING ARP CACHE FOR ADDRESS 10.1.3.51
LINK: OSA20C0L          ETHERNET: 00096B1A7454
QUERYING ARP CACHE FOR ADDRESS 10.1.3.52
LINK: OSA20C0L          ETHERNET: 00096B1A7454
QUERYING ARP CACHE FOR ADDRESS 10.1.3.220
LINK: OSA20C0L          ETHERNET: 000785F37302
QUERYING ARP CACHE FOR ADDRESS 10.1.3.240
LINK: OSA20C0L          ETHERNET: 0014F1464600
QUERYING ARP CACHE FOR ADDRESS 10.1.3.42
LINK: OSA20E0L          ETHERNET: 00096B1A73BF
QUERYING ARP CACHE FOR ADDRESS 10.1.3.41
LINK: OSA20E0L          ETHERNET: 00096B1A73BF
QUERYING ARP CACHE FOR ADDRESS 10.1.3.12
LINK: OSA20E0L          ETHERNET: 00096B1A73BF
QUERYING ARP CACHE FOR ADDRESS 10.1.3.22
LINK: OSA20E0L          ETHERNET: 00096B1A73BF
QUERYING ARP CACHE FOR ADDRESS 10.1.3.21
LINK: OSA20E0L          ETHERNET: 00096B1A73BF
QUERYING ARP CACHE FOR ADDRESS 10.1.3.32
LINK: OSA20E0L          ETHERNET: 00096B1A73BF
QUERYING ARP CACHE FOR ADDRESS 10.1.3.31
LINK: OSA20E0L          ETHERNET: 00096B1A73BF
QUERYING ARP CACHE FOR ADDRESS 10.1.3.52
LINK: OSA20E0L          ETHERNET: 00096B1A73BF
QUERYING ARP CACHE FOR ADDRESS 10.1.3.51
LINK: OSA20E0L          ETHERNET: 00096B1A73BF
QUERYING ARP CACHE FOR ADDRESS 10.1.3.240
LINK: OSA20E0L          ETHERNET: 0014F1464600
QUERYING ARP CACHE FOR ADDRESS 10.1.3.220
LINK: OSA20E0L          ETHERNET: 000785F37302
QUERYING ARP CACHE FOR ADDRESS 10.1.2.42
LINK: OSA20A0L          ETHERNET: 00096B1A74C2
QUERYING ARP CACHE FOR ADDRESS 10.1.2.41
LINK: OSA20A0L          ETHERNET: 00096B1A74C2
QUERYING ARP CACHE FOR ADDRESS 10.1.2.22
LINK: OSA20A0L          ETHERNET: 00096B1A74C2
QUERYING ARP CACHE FOR ADDRESS 10.1.2.21
LINK: OSA20A0L          ETHERNET: 00096B1A74C2
QUERYING ARP CACHE FOR ADDRESS 10.1.2.32
LINK: OSA20A0L          ETHERNET: 00096B1A74C2
QUERYING ARP CACHE FOR ADDRESS 10.1.2.31
LINK: OSA20A0L          ETHERNET: 00096B1A74C2
QUERYING ARP CACHE FOR ADDRESS 10.1.2.52
LINK: OSA20A0L          ETHERNET: 00096B1A74C2
QUERYING ARP CACHE FOR ADDRESS 10.1.2.50
LINK: OSA20A0L          ETHERNET: 00096B1A74C2
QUERYING ARP CACHE FOR ADDRESS 10.1.2.51
LINK: OSA20A0L          ETHERNET: 00096B1A74C2
QUERYING ARP CACHE FOR ADDRESS 10.1.2.12
LINK: OSA20A0L          ETHERNET: 00096B1A74C2
QUERYING ARP CACHE FOR ADDRESS 10.1.2.240
LINK: OSA20A0L          ETHERNET: 0014F1464600
QUERYING ARP CACHE FOR ADDRESS 10.1.2.220
LINK: OSA20A0L          ETHERNET: 000785F37302
QUERYING ARP CACHE FOR ADDRESS 10.1.4.51
LINK: IUTIQDF4L
QUERYING ARP CACHE FOR ADDRESS 10.1.4.41
```

```
LINK: IUTIQDF4L
QUERYING ARP CACHE FOR ADDRESS 10.1.4.31
LINK: IUTIQDF4L
QUERYING ARP CACHE FOR ADDRESS 10.1.4.21
LINK: IUTIQDF4L
QUERYING ARP CACHE FOR ADDRESS 10.1.4.11
LINK: IUTIQDF4L
QUERYING ARP CACHE FOR ADDRESS 10.1.5.51
LINK: IUTIQDF5L
QUERYING ARP CACHE FOR ADDRESS 10.1.5.41
LINK: IUTIQDF5L
QUERYING ARP CACHE FOR ADDRESS 10.1.5.31
LINK: IUTIQDF5L
QUERYING ARP CACHE FOR ADDRESS 10.1.5.21
LINK: IUTIQDF5L
QUERYING ARP CACHE FOR ADDRESS 10.1.5.11
LINK: IUTIQDF5L
QUERYING ARP CACHE FOR ADDRESS 10.1.6.51
LINK: IUTIQDF6L
QUERYING ARP CACHE FOR ADDRESS 10.1.6.41
LINK: IUTIQDF6L
QUERYING ARP CACHE FOR ADDRESS 10.1.6.31
LINK: IUTIQDF6L
QUERYING ARP CACHE FOR ADDRESS 10.1.6.21
LINK: IUTIQDF6L
QUERYING ARP CACHE FOR ADDRESS 10.1.6.11
LINK: IUTIQDF6L
QUERYING ARP CACHE FOR ADDRESS 10.1.7.51
LINK: IQDIOLNKOA01070B
QUERYING ARP CACHE FOR ADDRESS 10.1.7.41
LINK: IQDIOLNKOA01070B
QUERYING ARP CACHE FOR ADDRESS 10.1.7.31
LINK: IQDIOLNKOA01070B
QUERYING ARP CACHE FOR ADDRESS 10.1.7.21
LINK: IQDIOLNKOA01070B
QUERYING ARP CACHE FOR ADDRESS 10.1.7.11
LINK: IQDIOLNKOA01070B
66 OF 66 RECORDS DISPLAYED
END OF THE REPORT
```

These commands can help you to locate connectivity problems. If they do not, the next step in debugging a direct attached network problem is to gather documentation that shows more detailed information about traffic problems related to the interface and network.

To get this detailed information, the z/OS Communications Server typically uses the component trace to capture event data and save it to an internal buffer—or write 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

Important: 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 Chapter 8, “Diagnosis” on page 215.

- OSA-Express network traffic analyzer (OSAENTA) trace

  This trace provides a way to trace inbound and outbound frames for an OSA-Express2 feature in QDIO mode.

  – The function allows the z/OS Communications Server to control and format the tracing of frames collected in the OSA-Express2 feature at the network port.

  – It also provides the capability to trace frames discarded by the OSA-Express2 feature.

  SYSTCPOT is a new CTRACE component for collecting NTA trace data. The trace records can be formatted using the IPCS CTRACE command, specifying a component name of SYSTCPOT.

For more information about how to set up and enable the network traffic analyzer, refer to Chapter 8, “Diagnosis” on page 215.

4.9 References

For additional information, refer to:

- *HiperSockets Implementation Guide*, SG24-6816
- *OSA-Express Implementation Guide*, SG24-5948
- *z/OS Communications Server: IP Configuration Reference*, SC31-8776
- *z/OS Communications Server: SNA Resource Definition*, SC31-8778

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 topics.

Section	Topic
5.1, “Basic concepts” on page 142	The basic concepts of IP routing
5.2, “Routing in the z/OS environment” on page 148	Key characteristics of IP routing in z/OS Communications Server
5.3, “Dynamic routing protocols” on page 151	Detailed characteristics of dynamic routing protocols
5.4, “Implementing static routing in z/OS” on page 160	The implementation tasks and configuration examples for static routing
5.5, “Implementing OSPF routing in z/OS with OMPROUTE” on page 167	The implementation tasks and configuration examples for OSPF dynamic routing
5.6, “Problem determination” on page 184	Techniques for problem determination
5.7, “For additional information” on page 194	Additional materials related to routing

5.1 Basic concepts

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.

Bridging is often compared with routing, which might seem to accomplish precisely the same thing. However, note the primary difference between these functions:

- Bridging occurs at Layer 2 (the data link control layer) of the OSI reference model.
- Routing occurs at Layer 3 (the network layer).

This distinction provides bridging and routing 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 Terminology

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 toward 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 protocol used between dynamic routers running on TCP/IP hosts within a single autonomous system.
Exterior gateway protocols (EGP)	Dynamic route update protocols 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).

5.1.2 Direct routes, indirect routes, and default route

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

  The networks to which the host is directly attached are called direct routes. 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.

- Indirect routes

  The networks to which the host is not directly attached and reachable through one or more IP routers are called indirect routes. 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 routing entry with destination IP address and the IP address of the first router (the next 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.

- The default route

  The default route entry contains the IP address of the first router (the next hop) to be used when the destination IP address or network is not found in any of the direct or indirect routes.

We use Figure 5-1 to explain these types of routing entry.

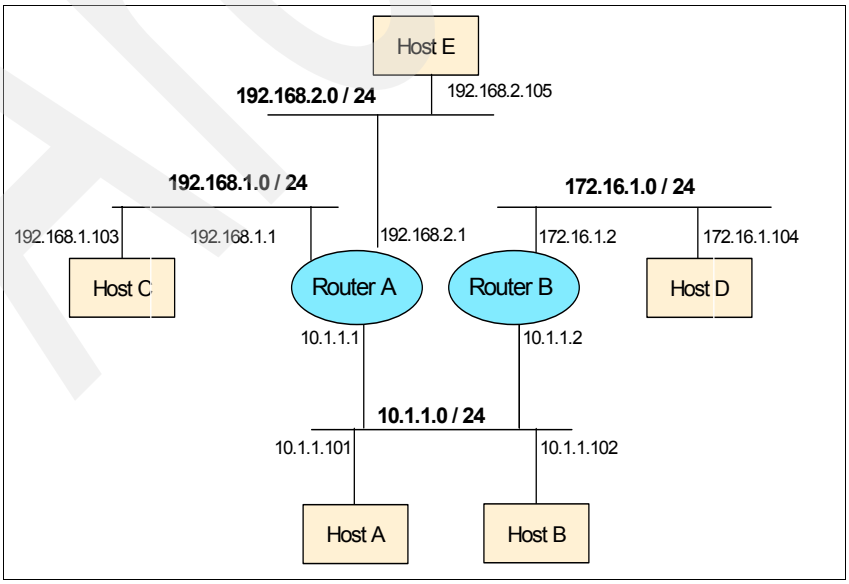


Figure 5-1 Sample network with multiple subnetworks

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 A in this example may contain the (symbolic) entries as listed in Table 5-2.

Table 5-2 IP routing table for Host A

Destination	IP address of Next Hop Router
10.1.1.0/24	Directly connected
192.168.1.0/24	10.1.1.1 (Router A)
172.16.1.0/24	10.1.1.2 (Router B)
Default	10.1.1.1 (Router A)
127.0.0.1	Loopback

The routing table contains routes to different routers in this network. When host A has an IP datagram to forward, it determines which IP address to forward it to using the IP routing algorithm and the routing table.Note that the /24 (a *prefix*), represents the length of subnet mask (a 24-bit mask, in this case).

Because Host A is directly attached to network 10.1.1.0/24, it maintains a direct route for this network. To reach other networks such as 192.168.1.0/24 and 172.16.1.0/24, it must have an indirect route through router A and router B, respectively, because these networks are not directly attached to it. Another option is to define a default route. If the indirect route to the network is not defined explicitly, the default route is used.

In this example, Host A reaches to Host B using the direct route. To reach Host C (192.168.1.103), it uses the indirect route to 192.168.1.0/24 and forwards the packet to Router A (10.1.1.1).

Likewise, to reach Host D, it uses the indirect route to 172.16.1.0/24 and forwards the packet to Router B (10.1.1.2). The indirect route to Host E (192.168.2.105) is not explicitly defined in the Host A. So, the default route is used and the Host A forwards the packet to Router A (10.1.1.1).

To reach any given IP network address, each host or router in the network needs to know only the next hop's IP address and not the full network topology.

5.1.3 Route selection

IP uses a unique algorithm to route an IP datagram. 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 network is directly attached:
   - 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:
   – If so, forward the packet directly to the destination.
   – If not, forward the packet to the next router as defined in the routing tables.

If two or more indirect routes are defined for the same destination, the route selection depends on the implementation of the routers or hosts. Some implementation always uses the top entry in the list, and some implementation uses all routes to distribute the packets. In some cases it is configurable with the provided parameters.

If two or more indirect routes are defined for the same destination but with different subnet mask length, the route with longest mask length is selected. This method is called *the longest match*.

5.1.4 Static routing and dynamic routing

In this section we explain the two ways to set up the necessary routing table in a system: using static routing, or dynamic routing.

Static routing

Static routing requires you to *manually* configure the routing tables yourself. This task is part of the configuration steps you follow when customizing 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 necessary routes may be simple for a small network. It offers 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.

The another drawback of static routing is that, even if the network failure occurs in the intermediate path to the destination, the routing table remains unchanged and keeps sending the packet according to the statically defined next hop routers. Sometimes it may cause the network to be unreachable. Also, if you fail to define the right next hop router in the route entry, the routers keep forwarding the packet using that entry. Even if there is a better route, the router does not change its next hop router until the changes are made to the static route entry.

If your network environment is small and manageable, with few to no network changes anticipated, then using static routes is an option (keeping in mind that your z/OS system is basically an application server environment). A good practice is to define only the default gateways to the exterior networks, and let the routers do the exterior routing. You can implement the static routing between the z/OS system and external router, and still let the external routers use the dynamic routing protocol to exchange route information.

Dynamic routing

Dynamic routing removes the need for static definition of the routing table. The network routing table is built dynamically, automatically exchanging route information among the routers in the network. This sharing of the routing information enables the routers to always calculate the best path through the network to any destination. When a network outage occurs in the intermediate route to the destination, the routers exchange the information about the outage and the best path is recalculated.

If your routing tables are complex due to network growth, or if the system must act as a gateway, it is far easier to let the system do the work for you by using dynamic routing.

The drawback of dynamic routing is the burden of route information exchange There are some configuration techniques you can use to reduce this burden, as explained in 5.2, “Routing in the z/OS environment” on page 148.

Dynamic routing protocols can be divided into two types: interior gateway protocols, and exterior gateway protocols.

Interior gateway protocols (IGPs) are dynamic route update protocols used between dynamic routers running on TCP/IP hosts within a single autonomous system. These protocols are used by the routers to exchange information about which IP routes the IP hosts that 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.

Exterior gateway protocols (EGPs) are dynamic route update protocols that are used between routers that are placed between two or more Autonomous Systems.

OSPF and RIP

In this section we discuss the interior gateway protocols OSPF, RIP version1, and RIP version2, which are supported by z/OS Communications Server.

- Open Shortest Path First (OSPF)

  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.

- Routing Information Protocol (RIP)

  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 OSPF

  IPv6 OSPF 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 (as with OSPF compared to RIP).

- IPv6 RIP

  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.

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

	RIP v1	RIP v2	IPv6 RIP	OSPF	IPv6 OSPF
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.

5.1.5 Choosing the routing method

The choice of a routing protocol is a major decision for the network administrator, and has a major impact on overall network performance. The selection depends on the network complexity, size, and administrative policies. The protocol chosen for one type of network may be inappropriate for other types of networks. Each unique environment must be evaluated against a number of fundamental design requirements, as explained here.

- 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, then link state or hybrid algorithms should be considered. Distance vector algorithms do not scale into these environments. Static routing also does not usually scale into large 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. Static routing can provide stability if the platform implements protocols like Virtual Router Redundancy Protocol (VRRP), Hot Standby Router Protocol (HSRP), or if redirected routes are accepted.

  On a z platform, OSAs can provide stability in a static routing environment through a feature called ARP Takeover. Refer to *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 3: High Availability, Scalability, and Performance*, SG24-7534., for more detailed information about ARP Takeover.

- 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 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. In static routing environments, convergence is a factor limited by the time it takes to update static routing tables manually.

- Metrics

  Metrics provide the ability to groom appropriate routing paths through the network. Link state algorithms consider bandwidth when calculating routes.

- 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, then standard routing protocols should be used. The IETF has dictated the operating policies for the distance vector and link state algorithms described in this IBM Redbook. Implementing these algorithms avoids any interoperability problems encountered with nonstandard protocols.

The administrator must assess the importance of each of these requirements when determining the appropriate routing protocol for an environment.

5.2 Routing in the z/OS environment

This section discusses the two IP routing methods provided by z/OS Communications Server. It also describes OMPROUTE and what to consider when implementing dynamic and static routes.

5.2.1 Static routing

In z/OS Communications Server, the static routes are defined with the BEGINROUTES statement block in the TCP/IP profile. The defined static routes are installed into the routing table of the TCP/IP stack. The GATEWAY statement also may be used in the TCP/IP profile to define static routes, but it is an obsolete statement. Instead, we recommend the use of the BEGINROUTES statement block.

Static routing can be combined with dynamic routing through the use of the OMPROUTE routing daemon. If the ROUTE statement in the BEGINROUTES statement block is coded with NOREPLACEABLE, then the static route is always preferred over the dynamically learned route for the same destination with the same subnet mask length.

If two or more routes to the same destination with same subnet mask length are defined in the z/OS Communications Server routing table, then the TCP/IP stack always uses the first *active* entry, by default. If you specify a IPCONFIG MULTIPATH statement in the TCP/IP profile, all routes for the same destination are used by per connection or per packet, depending on which option you specify for MULTIPATH.

5.2.2 Dynamic routing using OMPROUTE

In z/OS Communications Server IP, there is a multiprotocol routing daemon for dynamic routing called OMPROUTE. (The term *daemon* is used in UNIX to refer to a background server process.) It provides an alternative to the static TCP/IP routing 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 routing table.

Important: OMPROUTE is the only routing daemon included in the z/OS Communications Server V1R7 and later releases. The OROUTED routing daemon was 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 protocol in your configuration, use OMPROUTE to provide RIP support.

Supported dynamic routing protocols

OMPROUTE supports the OSPF, RIP version 1, and RIP version 2 routing protocols.

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).

How OMPROUTE works

OMPROUTE manages an OMPROUTE routing table. OMPROUTE installs the routes that are learned dynamically through other routers with routing protocol (OSPF and/or RIP) to the TCP/IP stack's routing table. When routing a packet to its destination, the TCP/IP stack makes decisions for route selection based on TCP/IP stack's routing table, not the OMPROUTE routing table.

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. The affinity to the TCP/IP stack is made by specifying the TCPIPJobname statement with the TCP/IP stack name in TCPIP.DATA file that OMPROUTE uses.

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.

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.

For IPv4, 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 with the INTERFACE configuration statement (unless it is a non-point-to-point interface and all default values specified on the INTERFACE statement are acceptable).

If both OSPF and RIP protocols are used in an OMPROUTE environment, then OSPF takes precedence over RIP. OSPF routes will be preferred over RIP routes to the same destination.

OMPROUTE allows the generation of multiple, equal-cost routes to a destination. If there are multiple routes for same destination with the same subnet mask length, the stack uses the first active route for all traffics. If you specify a IPCONFIG MULTIPATH statement in the

TCP/IP profile, the stack uses all routes for the same destination per connection or per packet, depending on which option you specify for MULTIPATH.

Considerations

Note that when coding static routes in BEGINROUTES statements, in conjunction with the OMPROUTE configuration, there are options for static routes: NOREPLACEABLE (the default) and REPLACEABLE.

OMPROUTE does *not* replace a NOREPLACEABLE static route, even it if has detected a dynamic route to the same destination, and the TCP/IP stack uses a NOREPLACEABLE static route to forward the packet. OMPROUTE replaces a REPLACEABLE static route if it detects a dynamic route to the same destination. The REPLACEABLE option enables the last resort to the destination in cases where OMPROUTE has not detected a dynamic route to the destination.

Also, 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.

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 and 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.

5.2.3 Policy-based routing

In a TCP/IP environment, the route is selected based on the destination IP address of the packet. The TCP/IP routing table is looked up for the matching entry for the destination IP address. This means that all types of packets destined to the same destination IP address, including interactive traffic (TSO, for example) and bulk traffic (FTP, for example), are forwarded to the same next hop router. In some cases, the bulk traffic may cause traffic congestion and can lead to a performance problem for interactive traffic.

z/OS V1R9 Communications Server introduces policy-based routing, which determines the destination based on the defined policy. Traffic descriptors such as TCP/UDP port numbers, application name, and source IP addresses can be used to define the policy to enable the optimized route selection.

Policy-based routing can use both static routes and dynamic routes, which are obtained with the OMPROUTE routing daemon.

For detailed information about policy-based routing, refer to *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 4: Security and Policy-Based Networking*, SG24-7535.

5.3 Dynamic routing protocols

z/OS Communications Server supports two different type of dynamic routing:

1. Open Shortest Path First
2. Routing Information Protocol (RIP)

5.3.1 Open Shortest Path First

This section provides a brief overview of the 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 consist merely of a description of the changed status. OSPF can divide its network into topology subsections, known as *areas*, within which broadcasts are confined. OSPF 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 so 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.
- There are no limitations to the routing metric. Although RIP restricts the routing metric to 16 hops, OSPF has virtually no restrictions.
- Multipath routing is allowed. 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

  This is a 32-bit number allocated to each router in the OSPF network protocol. This number is unique in the autonomous system. It represents the IP address of an interface defined on the OSPF node.

  For the z/OS implementation of the Router ID in OSPF, we recommend the use of a static VIPA address. Do not use a Dynamic VIPA as the Router ID, because the movement of the Router ID causes confusion in the OSPF routing protocol exchanges.

- 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, typically the backbone area. If you define more than one area, one of the areas must be the backbone area and the other area or areas are defined as non-backbone areas.

- Backbone area

  All OSPF networks should 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 router maintains 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. A virtual link can be used to satisfy this requirement.

- AS boundary routers

  These are the routers that connect the OSPF internetwork and exchange reachability information with other routers in other Autonomous Systems. They can 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

  This is a logical link that connects an area that does not have a physical link to a backbone area. The link is treated as a point-to-point link.

- Neighboring routers

  Routers that have interfaces to the same connection 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 after 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 after 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.

- Non-backbone area

  There are several types of non-backbone areas. A non-backbone area is identified by a four-octet area number that is not 0.0.0.0. There is a standard non-backbone area. There

are also two special types of non-backbone areas: the Stub area and the Totally Stubby Area.

- Stub Area

  A Stub Area is a non-backbone area that is connected to the backbone area through an Area Border Router. The Stub Area does not receive advertisements about destinations that are in other Autonomous Systems. Such advertisements are called “external link state advertisements” because they refer to Autonomous Systems external to this Autonomous System.

  The Stub Area knows only about intra-area destinations within the Stub Area. It knows about the Totally Stubby Area destinations that exist outside the Stub Area. It reaches external destinations through default routes sent to it by the ABR. With smaller link-state databases and smaller routing tables, Stub Areas consume less CPU storage and fewer CPU cycles.

- Totally Stubby Area

  Nodes in a Totally Stubby Area consume even less CPU storage and fewer CPU cycles for OSPF processing, because they maintain knowledge only of the intra-area destinations and the default routes to reach inter-area and external destinations.

  Note: We recommend that, if possible, you define z/OS OSPF nodes as members of a Totally Stubby Area in order to reduce the size of the link state database and reduce the CPU cycles required to produce a routing table. If Totally Stubby is not an option, then we recommend that you find other ways to minimize storage and CPU.

  For example, you might integrate a mainframe network running OSPF with a router network running Enhanced Interior Gateway Routing Protocol (EIGRP) to take advantage of the filtering capabilities of EIGRP, thus reducing the amount of protocol traffic between the OSPF network and the EIGRP network.

- Designated router

  A designated router (DR) is a router on a shared multi-access medium such as a LAN or ATM network. A DR performs most of the OSPF protocol activities for that network, like synchronizing database information and informing members of the broadcast network of changes to the network. The DR must be adjacent to all other routers on the broadcast medium. Every network or subnetwork on a broadcast network must have a DR and preferably a backup designated router (BDR).

Note: We recommend that you define non-z/OS routers attached to z/OS OSPF LAN broadcast networks as the DRs. z/OS CPU utilization is reduced if a non-z/OS router performs the work of the DR.

There is one exception to this rule when dealing with a HiperSockets network. A HiperSockets network is also a broadcast network; however, only z/OS, z/VM®, or Linux on System z nodes participate in a HiperSockets network. Therefore, at least some node inside the mainframe must be a DR on a HiperSockets LAN.

Complications can occur if the z/OS node is the DR on a LAN network when parallel interfaces into the LAN over a shared OSA exist. Shared OSAs can route over the shared OSA port without entering the network.

If the packet arrives over the backup interface instead of the primary parallel interface, the recipient discards the packet. The databases at the nodes become corrupted due to missing information, and lost adjacencies can result.

Therefore, we recommend that you not allow z/OS nodes with parallel interfaces and shared LANs to be the DR. If a z/OS node must be the DR, it should be connected to the broadcast medium via a non-shared OSA port.

- Backup designated router (BDR)

  The BDR is also adjacent to all other routers on the medium. It listens to DR conversations, and takes over if the DR fails. After the DR fails, the BDR becomes the DR and a new BDR 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, configure it with a router priority of zero.

- Transit Area

  A Transit Area is 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, and the database 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, and the status of an individual network component.

There are five types of LSAs:

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. After 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.

After 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

  This refers to 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

  This refers to 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

  This refers to 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 and receiving Hello packets. This is a typical example of how OSPF exploits a broadcast network. OSPF utilizes multicast in a broadcast network if implemented.

- Nonbroadcast multiaccess (NBMA)

  This refers to networks that support more than two attached routers, but have no broadcast capabilities. Because NBMA does not support multicasting, the OSPF Hello packets must be specifically addressed to each router. And because 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.

5.3.2 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, then 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 (such as OMPROUTE).

There are various types of PDUs, but the two most important PDUs are:

REQUEST PDU This PDU is sent from an RIP server as a request to other RIP servers to transmit their routing tables immediately.

RESPONSE PDU This PDU is sent from an RIP server to other RIP 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

Because RIP is designed for a specific network environment, it has some limitations as described here. Consider these limitations 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 belong 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.

5.3.3 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, then 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). Because 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 (this 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; because the designated router floods on behalf of 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* (because multiple IPv6 prefixes per link are allowed and expected, routing by subnet/prefix makes less sense). In OSPFv2, 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.

5.4 Implementing static routing in z/OS

In this section we implement a static routing scenario, as illustrated in Figure 5-2 on page 161. We only provide definition examples for the TCPIPA stack on SC30 because the examples for the TCPIPB stack on SC31 are similar. On TCPIPA, we define direct routes for interfaces such as OSA and HiperSockets and indirect routes for TCPIPB VIPAs. We also define default routes via our two Cisco 6500 series switches (layer 3 switches).

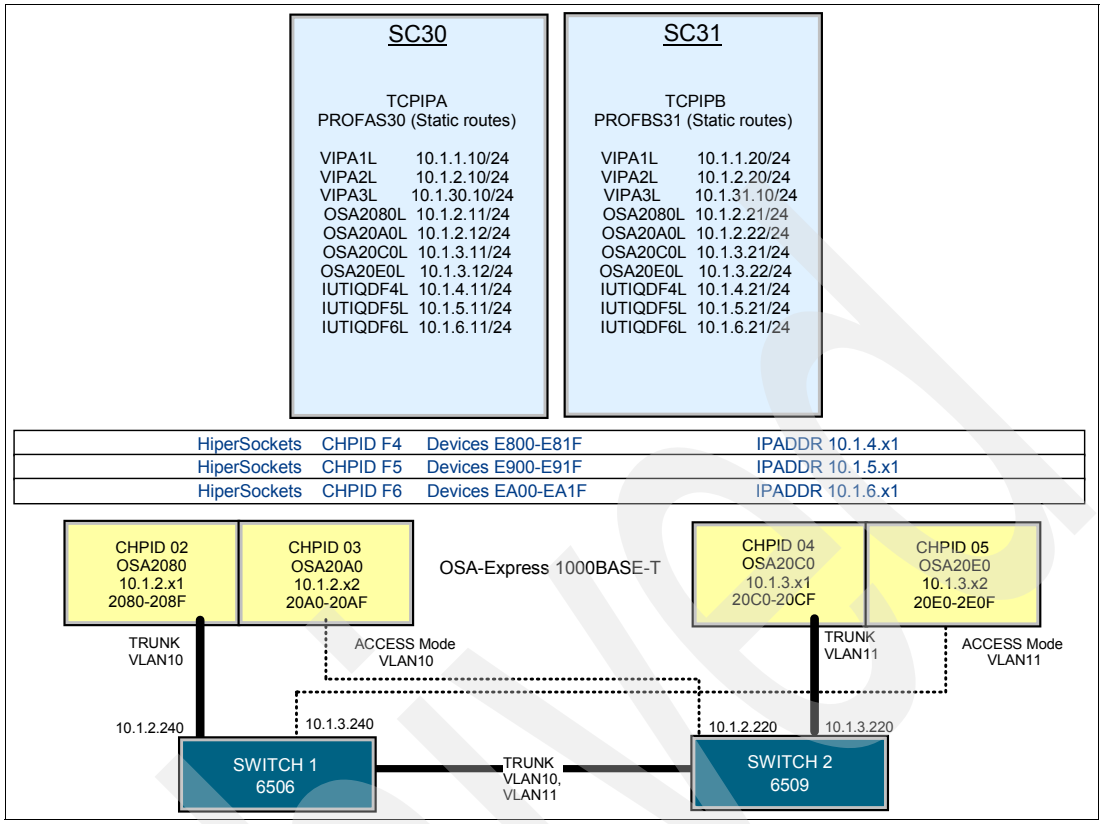


Figure 5-2 Static routing scenario

Dependencies

All subnetworks defined in the TCP/IP stack that are used by the application servers, including static and dynamic VIPAs, must also have static routing definitions in the routers In our case, the layer 3 switches (routers) do not need static route definitions for direct routes. We defined indirect routes for TCPIPA and TCPIPB VIPAs in the routers.

Considerations

When planning to design a static routing environment on a z/OS Communication Server, you need to address some issues. Keep in mind that 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.

The routing table's management is manual, thus increasing the possibility of outages caused by definition errors. If a destination (sub)network becomes unreachable, then the static routes for that (sub)network will remain in the routing table, and packets will still be forwarded to the destination. The only way to remove static routes from the routing table is for the network administrator to update the routing table.

We recommend that you define as few static routing definitions as possible when implementing a static routing environment, keeping in mind that our z/OS system is basically an application server environment. It is good practice to define only the default gateways to the exterior networks, and let the routers do the exterior routing. You can implement the static routing between the z/OS system and external router, and still let the external router use the dynamic routing protocol.

In the routers, we recommend that you define only the route definitions to the VIPA subnetworks. The interior subnetworks, such as XCF and HiperSockets, do not usually need to be reached by the corporate network, so they do not need to be defined.

5.4.1 Implementation tasks

To implement the static routing scenario, follow these steps:

1. Update the TCP/IP profile.
2. Configure the routers.

Update the TCP/IP profile

In the TCP/IP profile, use the BEGINROUTES block and ROUTE statement to define the following routes:

- A direct route to all local interfaces (except static VIPAs, dynamic VIPAs, or XCF)

  To define a direct route, specify = for its First Hop.
- An indirect route to the subnetwork

  To define a direct route, specify the IP address of the next hop router for its first hop.
- Default gateway statements to route all packets being sent to unknown destinations

Example 5-1 shows our definition example.

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 5-1 Direct routes configuration

```
; ****************************************
; TCPIPA.TCPPARMS(PROFA30S)
; ****************************************
.....
BEGINRoutes
; Direct Routes - Routes that are directly connected to my interfaces
;      Destination     Subnet Mask  First Hop Link Name     Packet Size
 ROUTE 10.1.2.0       255.255.255.0 = 1       OSA2080L       MTU 1492
 ROUTE 10.1.2.0/24                  = 1       OSA20A0L       MTU 1492
 ROUTE 10.1.3.0/24                  = 1       OSA20C0L       MTU 1492
 ROUTE 10.1.3.0/24                  = 1       OSA20E0L       MTU 1492
 ROUTE 10.1.4.0/24                  = 2       IUTIQDF4L      MTU 8192
 ROUTE 10.1.5.0/24                  = 2       IUTIQDF5L      MTU 8192
 ROUTE 10.1.6.0/24                  = 2       IUTIQDF6L      MTU 8192
;
; Indirect Routes - Routes that are not directly connected to my interfaces
;      Destination     Subnet Mask  First Hop    Link Name  Packet Size ;
 ROUTE 10.1.1.20/32                 10.1.4.21    IUTIQDF4L  MTU 8192
 ROUTE 10.1.2.20/32                 10.1.4.21    IUTIQDF4L  MTU 8192
 ROUTE 10.1.31.10/32                10.1.4.21    IUTIQDF4L  MTU 8192
 ROUTE 10.1.100.0/24                10.1.2.240 3 OSA2080L   MTU 1492
 ROUTE 10.1.100.0/24                10.1.2.240 3 OSA20A0L   MTU 1492
 ROUTE 10.1.100.0/24                10.1.3.240 3 OSA20C0L   MTU 1492
 ROUTE 10.1.100.0/24                10.1.3.240 3 OSA20E0L   MTU 1492
;
; Default Routes - Routes irectly connected to my interfaces
;      Destination     Subnet Mask  First Hop    Link Name  Packet Size ;
 ROUTE DEFAULT                      10.1.2.240 4 OSA2080L   MTU 1492
 ROUTE DEFAULT                      10.1.2.220 4 OSA2080L   MTU 1492
```

```
 ROUTE DEFAULT                     10.1.2.240 4 OSA2OA0L   MTU 1492
 ROUTE DEFAULT                     10.1.2.220 4 OSA2OA0L   MTU 1492
 ROUTE DEFAULT                     10.1.3.240 4 OSA2OC0L   MTU 1492
 ROUTE DEFAULT                     10.1.3.220 4 OSA2OC0L   MTU 1492
 ROUTE DEFAULT                     10.1.3.240 4 OSA2OE0L   MTU 1492
 ROUTE DEFAULT                     10.1.3.220 4 OSA2OE0L   MTU 1492
;
ENDROUTES
```

- 1 Define the direct routes for OSA interfaces. Specify the subnet mask with decimal format (such as 255.255.255.0) or the *prefix* length.
- 2 Define the direct routes for HiperSockets interfaces.
- Note that the first hop parameter is defined as an equal sign (=) 1 to identify this as a direct route.
- 3 Define the indirect routes to reach the external network. The next hop is router 1 (10.1.2.240 and 10.1.3.240).
- 4 Define the default routes, to reach the external network, which are not explicitly defined as indirect routes. The next hop is router 1 (10.1.2.240 and 10.1.3.240) and router 2 (10.1.2.220 and 10.1.3.220).

Configure the routers

Define the static routes to the VIPA or the physical interfaces which are not on the subnet that the routers are directly connected to (HiperSockets, for example). In our example, 10.1.2.0/24 and 10.1.3.0/24 are direct routes of the router, and we do not need to define the static routes for those subnets.

Example 5-2 shows the example of router (layer 3 switch) configuration.

Example 5-2 Static route definition in router

```
.....
ip route 10.1.1.10 255.255.255.255 10.1.2.11    1
ip route 10.1.2.10 255.255.255.255 10.1.2.11    1
ip route 10.1.30.10 255.255.255.255 10.1.2.11    1
.....
```

- 1 Define the static route to the static VIPA in TCPIPA. The next hop address is the IP address of the OSA physical interface. In our example we define this static route with 32-bit mask (255.255.255.255), but you can use a mask length shorter than 32-bit.

5.4.2 Activation and verification

To activate and verify the static routing scenario, follow these steps:

1. Apply changes to TCP/IP profile.
2. Verify the connectivity.

Apply changes to TCP/IP profile

To apply the changes to static routes, do *one* of the following:

- Restart the TCP/IP stack.
- Modify the TCP/IP definition with VARY TCPIP,*procname*,OBEYFILE command.

After you perform one of these tasks, then all static routes are listed in the TCP/IP routing table.

Example 5-3 illustrates applying changes by using the OBEYFILE command.

Important: When using the OBEYFILE command, include all static routes that you want to define. The OBEYFILE command replaces the entire BEGINROUTES block.

Example 5-3 Applying changes with the OBEYFILE command

```
V TCPIP,TCPIPA,O,DSN=TCPIPA.TCPPARMS(PROFA3OS)
EZZ0060I PROCESSING COMMAND: VARY TCPIP,TCPIPA,O,DSN=TCPIPA.TCPPARMS(P
ROFA3OS)
EZZ0300I OPENED OBEYFILE FILE 'TCPIPA.TCPPARMS(PROFA3OS)'
EZZ0309I PROFILE PROCESSING BEGINNING FOR 'TCPIPA.TCPPARMS(PROFA3OS)'
.....
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'TCPIPA.TCPPARMS(PROFA3OS)'
.....
```

Verify the connectivity

To verify if the static routing table is built as expected, the following commands are useful.

Note: Netstat commands can be executed as TSO commands, z/OS UNIX shell commands, or Display commands on the system console. Our examples are the result of Display commands on the system console, but their output is identical to the TSO and z/OS UNIX shell output.

Display the device status

Use the D TCPIP,TCPIPA,Netstat,DEVlink 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.

Example 5-4 Netstat DEVlink command display

```
D TCPIP,TCPIPA,N,DEV
EZD0101I NETSTAT CS V1R9 TCPIPA 867
DEVNAME: OSA2080           DEVTYPE: MPCIPA
  DEVSTATUS: READY 1
  LNKNAME: OSA2080L          LNKTYPE: IPAQENET   LNKSTATUS: READY 1
    NETNUM: N/A  QUESIZE: N/A  SPEED: 0000001000
    IPBROADCASTCAPABILITY: NO
    CFGROUTER: NON                   ACTROUTER: NON
    ARPOFFLOAD: YES                  ARPOFFLOADINFO: YES
    ACTMTU: 8992
    VLANID: 10                       VLANPRIORITY: DISABLED
    DYNVLANREGCFG: NO                DYNVLANREGCAP: YES
    READSTORAGE: GLOBAL (4096K)      INBPERF: BALANCED
    CHECKSUMOFFLOAD: YES
    SECCLASS: 255                    MONSYSPLEX: NO
  BSD ROUTING PARAMETERS:
    MTU SIZE: 1492             METRIC: 100
    DESTADDR: 0.0.0.0          SUBNETMASK: 255.255.255.0
  MULTICAST SPECIFIC:
    MULTICAST CAPABILITY: YES
    GROUP             REFCNT        SRCFLTMD
    -----             ------        --------
    224.0.0.1         0000000001    EXCLUDE
      SRCADDR: NONE
  LINK STATISTICS:
    BYTESIN                          = 2492
```

```
    INBOUND PACKETS                   = 14
    INBOUND PACKETS IN ERROR          = 0
    INBOUND PACKETS DISCARDED         = 0
    INBOUND PACKETS WITH NO PROTOCOL  = 0
    BYTESOUT                          = 536
    OUTBOUND PACKETS                  = 6
    OUTBOUND PACKETS IN ERROR         = 0
    OUTBOUND PACKETS DISCARDED        = 0
.....
```

- 1 Make sure the DEVSTATUS and LNKSTATUS are both READY.

Display routing table

Use the Netstat ROUTe command to display the routing table in a TCP/IP stack. A sample of the command is shown in Example 5-5.

Example 5-5 Netstat ROUTe resulting display

```
D TCPIP,TCPIPA,N,ROUTE
EZD0101I NETSTAT CS V1R9 TCPIPA 066
IPV4 DESTINATIONS
DESTINATION         GATEWAY          FLAGS     REFCNT  INTERFACE
DEFAULT             10.1.2.240       UGS 4     000000  OSA2080L  1
DEFAULT             10.1.2.220       UGS       000000  OSA2080L
DEFAULT             10.1.2.240       UGS       000001  OSA20A0L
DEFAULT             10.1.2.220       UGS       000000  OSA20A0L
DEFAULT             10.1.3.240       UGS       000000  OSA20C0L
DEFAULT             10.1.3.220       UGS       000000  OSA20C0L
DEFAULT             10.1.3.240       UGS       000000  OSA20E0L
DEFAULT             10.1.3.220       UGS       000000  OSA20E0L
10.1.1.10/32        0.0.0.0          UH        000000  VIPA1L
10.1.1.20/32        10.1.4.21        UGHS      000000  IUTIQDF4L 2
10.1.2.0/24         0.0.0.0          US        000000  OSA2080L  3
10.1.2.0/24         0.0.0.0          US        000000  OSA20A0L
10.1.2.10/32        0.0.0.0          UH        000000  VIPA2L
10.1.2.11/32        0.0.0.0          UH        000000  OSA2080L
10.1.2.12/32        0.0.0.0          UH        000000  OSA20A0L
10.1.2.20/32        10.1.4.21        UGHS      000000  IUTIQDF4L
10.1.3.0/24         0.0.0.0          US        000000  OSA20C0L
10.1.3.0/24         0.0.0.0          US        000000  OSA20E0L
10.1.3.11/32        0.0.0.0          UH        000000  OSA20C0L
10.1.3.12/32        0.0.0.0          UH        000000  OSA20E0L
10.1.4.0/24         0.0.0.0          US        000000  IUTIQDF4L
10.1.4.11/32        0.0.0.0          UH        000000  IUTIQDF4L
10.1.5.0/24         0.0.0.0          US        000000  IUTIQDF5L
10.1.5.11/32        0.0.0.0          UH        000000  IUTIQDF5L
10.1.6.0/24         0.0.0.0          US        000000  IUTIQDF6L
10.1.6.11/32        0.0.0.0          UH        000000  IUTIQDF6L
10.1.8.10/32        0.0.0.0          UH        000000  VIPL0A01080A
10.1.30.10/32       0.0.0.0          UH        000000  VIPA3L
10.1.31.10/32       10.1.4.21        UGHS      000000  IUTIQDF4L
10.1.100.0/24       10.1.2.240       UGS       000000  OSA2080L
10.1.100.0/24       10.1.2.240       UGS       000000  OSA20A0L
10.1.100.0/24       10.1.3.240       UGS       000000  OSA20C0L
10.1.100.0/24       10.1.3.240       UGS       000000  OSA20E0L
127.0.0.1/32        0.0.0.0          UH        000001  LOOPBACK
IPV6 DESTINATIONS
DESTIP:   ::1/128
  GW:     ::
  INTF:   LOOPBACK6         REFCNT:  000000
  FLGS:   UH                MTU: 65535
```

```
41 OF 41 RECORDS DISPLAYED
END OF THE REPORT
```

- 1 The default route is defined. If there are multiple default route entries as shown in the example, only the first active entry (interface OSA2080L) is used.
- 2 The indirect route to VIPA in TCPIPB is defined.
- 3 The direct route for OSA physical interface is defined.
- 4 S in the FLAG field stands for non-replaceable static route entry. For replaceable static route entries, FLAG Z would be displayed.

Check the connectivity using PING command

The PING command can be executed using the TSO PING command or the z/OS UNIX **ping** command. Example 5-6 shows the display of the TSO PING command; the ping is successful.

In a CINET environment where multiple TCP/IP stacks are configured, use the TCP option for the TSO PING command and the **-p** option for the z/OS UNIX **ping** command to specify the TCP/IP stack name you want to issue the ping command from.

You do not need to specify these options if the user issuing this command is already associated to the TCP/IP stack (with SYSTCPD DD, for example).

You do not need to specify these options if your environment is an INET environment where only one TCP/IP stack is configured.

Example 5-6 TSO PING command display

```
TSO PING 10.1.1.20 (TCP TCPIPA
CS V1R9: Pinging host 10.1.1.20
 Ping #1 response took 0.000 seconds.
 ***
```

Example 5-7 shows the display of z/OS UNIX **ping** command.

Example 5-7 z/OS UNIX ping command display

```
CS06 @ SC30:/u/cs06>ping -p TCPIPA 10.1.1.20
CS V1R9: Pinging host 10.1.1.20
Ping #1 response took 0.000 seconds.
```

Verify the selected route with TRACEROUTE command

Traceroute can be invoked by either the TSO TRACERTE command or the z/OS UNIX shell **traceroute/otracert** command. Example 5-8 shows the example of the display. We see the router 1 (10.1.2.240) is the next hop router to reach the destination IP address 10.1.100.221.

In a CINET environment where multiple TCP/IP stacks are configured, use the TCP option for TSO TRACERTE command and the **-a** option for the z/OS UNIX **traceroute** command to specify the TCP/IP stack name you want to issue the TRACEROUTE command from.

You do not need to specify these options if the user issuing this command is already associated to the TCP/IP stack (with SYSTCPD DD, for example).

You do not need to specify these options if your environment is an INET environment where only one TCP/IP stack is configured.

Example 5-8 Tracerte command results

```
TSO TRACERTE 10.1.100.221 (TCP TCPIPA
```

```
CS V1R9: Traceroute to 10.1.100.221 (10.1.100.221)
 1 router1 (10.1.2.240)  0 ms  0 ms  0 ms
 2 10.1.100.221 (10.1.100.221)  0 ms  0 ms  0 ms
 ***
```

5.5 Implementing OSPF routing in z/OS with OMPROUTE

In this scenario we show a dynamic routing implementation. In our example we configure OSPF for 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 146). Although OSPF requires higher CPU processing, we can reduce that requirement by making the z/OS Communications Server a part of the OSPF Stub Area or Totally Stubby Area.

Figure 5-3 depicts the environment we use for the OSPF scenario. The TCPIPA stack is running on SC30. We create the OMPROUTE procedure OMPA to establish affinity to TCPIPA. Likewise, we create OMPB for TCPIPB on SC31.

We define a z/OS TCP/IP to be a member of OSPF Totally Stubby Area. The external routers (layer 3 switches) represent the ABRs between the Totally Stubby Area and the backbone area. We made the external routers to be DR or BDR to reduce the routing workloads required in the z/OS.

Because the configuration examples for TCPIPB and OMPB on SC31 are similar to those examples for TCPIPA and OMPA on SC30, we only show the configuration examples on SC30.

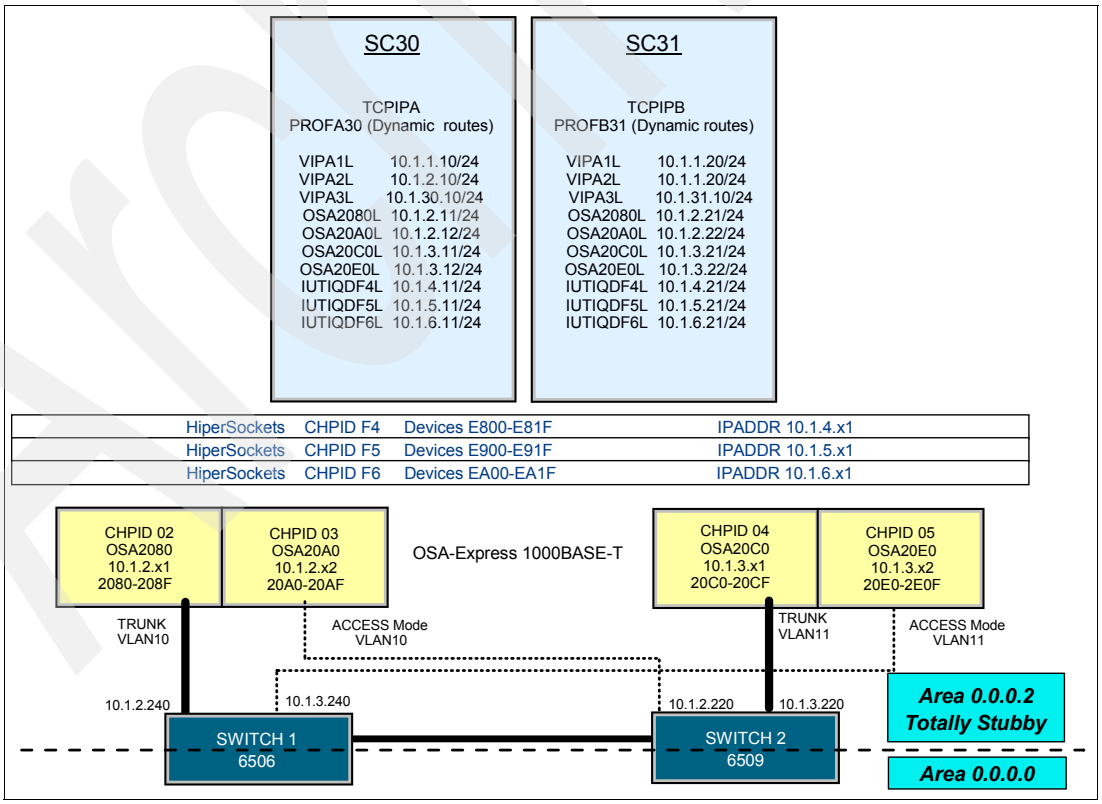


Figure 5-3 Dynamic routing scenario, using OSPF

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 be consistent with those defined to the IP routers.

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, take care 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 accessibility.

Recommendations

We recommend that you define the z/OS Communications Server environment as an OSPF Stub Area to reduce the CPU process needed for managing the routing table. 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.

We also recommend that you make the external routers be DR or BDR, and do not allow z/OS systems to be DR or BDR, in order to reduce the routing burden for z/OS systems. DR or BDR is selected in each LAN segment or VLAN. However, on HiperSockets links, z/OS systems are the only participants. One of the z/OS on the HiperSockets network has to take the role of DR (and optionally, another one can take the role of BDR).

5.5.1 Implementation tasks

To implement and configure OMPROUTE in the z/OS Communications Server, follow these steps:

1. Create the OMPROUTE cataloged procedure.
2. Define the OMPROUTE environment variables.
3. Update the TCPIP.DATA file.
4. RACF-authorize user IDs for starting OMPROUTE.
5. Start syslogd.
6. Change port 520 and 521 definitions to NOAUTOLOG.
7. Create the OMPROUTE configuration file.
8. Configure routers.

We only show the configuration examples for the TCPIPA stack and omit the examples for the TCPIPB stack.

We do not define any static routes in TCP/IP profile in conjunction with the dynamic routing.

Create the OMPROUTE cataloged procedure

We create the OMPROUTE cataloged procedure by copying the sample in hlq.SEZAINST(OMPROUTE) to our PROCLIB. We specify the STDENV file name and OMPCFG file name, as shown in Example 5-9 on page 169.

Example 5-9 OMPROUTE cataloged procedure

```
//OMPA30 PROC STDENV=OMPENA&SYSCLONE 1
//OMPA30 EXEC PGM=OMPROUTE,REGION=0M,TIME=NOLIMIT,
//        PARM=('POSIX(ON) ALL31(ON)',
//            'ENVAR("_BPXK_SETIBMOPT_TRANSPORT=TCPIPA"',
//            '"_CEE_ENVFILE=DD:STDENV")/') 2
//STDENV   DD DISP=SHR,DSN=TCPIP.SC&SYSCLONE..STDENV(&STDENV)
//SYSOUT   DD SYSOUT=*
//OMPCFG   DD DSN=TCPIPA.TCPPARMS(OMPA&SYSCLONE.),DISP=SHR 3
//CEEDUMP  DD SYSOUT=*,DCB=(RECFM=FB,LRECL=132,BLKSIZE=132)
```

The descriptions for the tags shown in Example 5-9 are as follows:

- 1 Specifies the STDENV variable. We can use a common procedure for all images within the same server environment by specifying the &SYSCLONE variable. The &SYSCLONE value for this LPAR is 30.
- 2 Each OMPROUTE procedure in the same server will have its own environment variables based on this DD.
- 3 The OMPCFG DD card is new in z/OS V1R9. It permits you to specify the OMPROUTE configuration file within the JCL. The DD card enables the use of an MVS system symbol that can make the procedure shareable across TCP/IP stacks. If you specify the configuration file here, you can omit the statement OMPROUTE_FILE from the STDENV file.

Tip: OMPROUTE can be started as a z/OS procedure, from the z/OS shell, or from AUTOLOG.

Define the OMPROUTE environment variables

To define our OMPROUTE environment variables we use a STDENV file, pointed to by the STDENV DD statement in our OMPROUTE procedure. Example 5-10 shows the STDENV file we use in our example.

Example 5-10 OMPROUTE environment variables

```
; *************************************
; TCPIP.SC30.STDENV(OMPENA30)
; *************************************
RESOLVER_CONFIG=//'TCPIPA.TCPPARMS(DATAA&SYSCLONE.)' 1
;OMPROUTE_FILE=//'TCPIPA.TCPPARMS(OMPA30)' 2
OMPROUTE_DEBUG_FILE=/etc/omproute/debug30a
OMPROUTE_DEBUG_FILE_CONTROL=100000,5
```

- 1 Specify the TCPIP.DATA file. The &SYSCLONE value for this LPAR is 30. If you do not want to use MVS system symbols, you can define the hard-coded member name as shown:

  ```
  RESOLVER_CONFIG=//'TCPIPA.TCPPARMS(DATAA30)'
  ```

- 2 We can omit the OMPROUTE_FILE statement if we have coded the OMPCFG DD statement in the OMPROUTE started procedure.

  With the appropriate naming conventions, we can make both the OMPROUTE environment variable file and the OMPROUTE started procedure shareable across multiple TCP/IP stacks.

Important: When defining the STDENV (_CEE_ENVFILE) file with a z/OS data set, the data set must be allocated with RECFM=V. Using RECFM=F or RECFM=FB is not recommended, because the fixed setting enables padding with blanks for the environment variables.

Update the TCPIP.DATA file

Our test environment is running under CINET. With CINET there is often a global TCPIP.DATA file and a stack-specific local TCPIP.DATA file. The keywords specified in the global TCPIP.DATA cannot be overridden with parameters in any local TCPIP.DATA files.

In the CINET environment, the Global Resolver configuration file contains keywords that are shared with all TCP/IP stacks on the z/OS image, and should omit the stack-specific keywords such as TCPIPJobname and Hostname. Those parameters should be specified in the local TCPIP.DATA file. If a specific parameter is not found in the global TCPIP.DATA, the local TCPIP.DATA file is searched according to the search order. You can read more about the Resolver in Chapter 2, “The Resolver” on page 19.

Example 5-11 shows the global TCPIP.DATA file used in our example.

Example 5-11 Global TCPIP.DATA file

```
; ****************************************
; TCPIPA.TCPPARMS(GLOBAL)
; ****************************************
DOMAINORIGIN  ITSO.IBM.COM
NSINTERADDR  10.12.6.7
NSPORTADDR 53
RESOLVEVIA UDP
RESOLVERTIMEOUT 10
RESOLVERUDPRETRIES 1
LOOKUP LOCAL DNS
```

Then each stack has a stack-specific local TCPIP.DATA file identifying stack-specific parameters such as TCPIPJobname and Hostname. Example 5-12 shows the local TCPIP.DATA file used in our example.

Example 5-12 Local TCPIP.DATA file

```
; ****************************************
; TCPIPA.TCPPARMS(DATAA30)
; ****************************************
TCPIPJOBNAME TCPIPA                1
HOSTNAME WTSC30A
DATASETPREFIX TCPIPA               2
MESSAGECASE MIXED
```

- 1 Specify the TCP/IP stack name that OMPROUTE should establish affinity to, using the TCPIPJobname statement.
- 2 Specify the data set prefix (hlq) that OMPROUTE should use.

In an INET environment, usually only a global TCPIP.DATA file is used. It should contain the keywords (TCPIPJobname and DATASETPREFIX) used by OMPROUTE. The TCPIPJobname parameter specifies the name of TCP/IP stack that OMPROUTE establishes an affinity with.

RACF-authorize user IDs for starting OMPROUTE

To reduce the 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. (0). In our test environment, we executed the command shown in Example 5-13.

Example 5-13 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
```

- 1 Specify the OMPROUTE cataloged procedure name for ID parameter.

Important: OMPROUTE must be started by a RACF-authorized user ID.

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. Update the syslogd.conf file to isolate all OMPROUTE messages to a specific output destination file. Example 5-14 shows how we configured the syslogd to receive all error, warning, info, and notice messages in a syslog file.

Example 5-14 Syslogd configuration file

```
##********************************************************************
#*                                                                    *
#* syslog.conf - Defines the actions to be taken for the specified    *
#*               facilities/priorities by the syslogd daemon.         *
#*                                                                    *
*.OMPA*.*.err /tmp/syslog/ompa.err.log 1
```

- 1 Specify the syslog output destination file name for the OMPA-related messages.

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 5-15.

Example 5-15 Ports 520 and 521 defined as NOAUTOLOG

```
; ****************************************
; TCPIPA.TCPPARMS(PROFA30)
; ****************************************
.....
  520 UDP OMPA NOAUTOLOG  ; OMPROUTE IPv4 RIPV2
  521 UDP OMPA NOAUTOLOG  ; OMPROUTE IPv6 RIPV2
.....
```

Important: If you fail to take one of these actions, OMPROUTE will be periodically canceled and restarted by TCP/IP.

Create the OMPROUTE configuration file

We defined the parameters for OSPF implementation in the OMPROUTER configuration file. Example 5-16 shows the configuration we used in our example. We defined a z/OS TCP/IP to be a Totally Stubby Area, the interfaces as part of the Stub Area, and other parameters.

The search order for OMPROUTE configuration file is:

1. OMPCFG DD statement in the OMPROUTE started procedure
2. OMPROUTE_FILE environment variable
3. /etc/omproute.conf
4. hlq.ETC.OMPROUTE.CONF

Example 5-16 OMPROUTE configuration file

```
Area Area_Number=0.0.0.2        1
     Stub_Area=YES              2
     Authentication_type=None
     Import_Summaries=Yes;      3
OSPF
   RouterID=10.1.30.10           4
   Comparison=Type2
   Demand_Circuit=YES;
Global_Options
 Ignore_Undefined_Interfaces=YES          5
;
Routesa_Config Enabled=No;
; Static vipa
OSPF_Interface IP_address=10.1.30.10      6
          Name=VIPA3L                     7
          Subnet_mask=255.255.255.0
          Attaches_To_Area=0.0.0.2        8
          Advertise_VIPA_Routes=HOST_ONLY 9
          Cost0=10                        10
          MTU=65535;
; Static vipa
OSPF_interface ip_address=10.1.1.10
          name=VIPA1L
          subnet_mask=255.255.255.0
          Advertise_VIPA_Routes=HOST_ONLY
          attaches_to_area=0.0.0.2
          cost0=10
          mtu=65535
OSPF_interface ip_address=10.1.2.10
          name=VIPA2L
          subnet_mask=255.255.255.0
          Advertise_VIPA_Routes=HOST_ONLY
          attaches_to_area=0.0.0.2
          cost0=10
          mtu=65535
; OSA Qdio VLAN10
OSPF_Interface IP_address=10.1.2.*        11
          Subnet_mask=255.255.255.0
          Router_Priority=0               12
          Attaches_To_Area=0.0.0.2
          Cost0=100
          MTU=1492;
; OSA Qdio VLAN11
OSPF_Interface IP_address=10.1.3.*
          Subnet_mask=255.255.255.0
          Router_Priority=0
```

```
            Attaches_To_Area=0.0.0.2
            Cost0=100
            MTU=1492;
; Hipersockets 10.1.4.x
OSPF_Interface IP_address=10.1.4.*
            Subnet_mask=255.255.255.0
            Router_Priority=1
            Attaches_To_Area=0.0.0.2
            Cost0=90
            MTU=8192;
; Hipersockets 10.1.5.x
OSPF_Interface IP_address=10.1.5.*
            Subnet_mask=255.255.255.0
            Router_Priority=1
            Attaches_To_Area=0.0.0.2
            Cost0=90
            MTU=8192;
; Hipersockets 10.1.6.x
OSPF_Interface IP_address=10.1.6.*
            Subnet_mask=255.255.255.0
            Router_Priority=1
            Attaches_To_Area=0.0.0.2
            Cost0=90
            MTU=8192;
;
; Dynamic vipa VIPADEFINE
ospf_interface ip_address=10.1.8.*
            subnet_mask=255.255.255.0
            Advertise_VIPA_Routes=HOST_ONLY
            attaches_to_area=0.0.0.2
            cost0=10
            mtu=65535
;
; Dynamic vipa VIPADEFINE
ospf_interface ip_address=10.1.9.*
            subnet_mask=255.255.255.0
            Advertise_VIPA_Routes=HOST_ONLY
            attaches_to_area=0.0.0.2
            cost0=10
            mtu=65535
;
;AS_Boundary_routing                    13
; Import_Direct_Routes=yes;
```

The descriptions for the tags shown in Example 5-16 on page 172 are as follows:

- 1 Define the OSPF Area (Area 2).
- 2 Indicates Area 2 is a Stub Area.
- 3 Import_Summaries has meaning only if coded in an ABR. It makes the area connected to the ABR a Totally Stubby Area. If you coded the parameter in OMPROUTE on a Stub Area, it is ignored but it functions as a reminder that the ABR -- the layer 3 switch -- is defining our Stub Area as a Totally Stubby Area.
- 4 Defines the router internal IP address to be represented as the router ID.

Note: The RouterID must be unique on each OMPROUTE configuration. We highly recommend that you code the RouterID statement (4), either with the static VIPA address or with an interface IP address, because the dynamic VIPAs (DVIPAs) that can move between z/OS hosts within a sysplex.

OMPROUTE cycles through all the OSPF interfaces until it finds a non-DVIPA interface if no RouterID is coded. A message (EZZ8134I) will be issued if a Dynamic VIPA is explicitly coded or chosen as a RouterID, since this is not recommended.

The interface with an IP address which represents RouterID (6) has to be coded explicitly as shown in Example 5-16, not using the wild card (*). Therefore, if the RouterID is hard-coded as it is in this example, only portions of the configuration file may be shared across systems.

- 5 Tells OSPF not to build an INTERFACE statement automatically for interfaces not defined in the OMPROUTE configuration file.
- 6 Defines a specific interface to be an OSPF interface. When the specific IP address is coded (not the wildcard as in 10) the NAME statement must also be configured (see 7).
- 7 The NAME statement identifies the link as an OSPF interface. The NAME statement must match the name specified in the LINK statement in the TCP/IP profile.
- 8 Defines the interface should belong to Area 2.
- 9 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. By default both the host and the subnet routes are advertised. Only the VIPA host route is advertised when this option is set to HOST_ONLY. We recommend the use of HOST_ONLY for VIPAs unless you have a compelling reason to advertise the subnet route.
- 10 We define the interface cost of VIPA to be 10, HiperSockets to be 90, and OSA to be 100. We made the cost of HiperSockets smaller than OSA so that the HiperSockets route is preferred for the mainframe-to-mainframe communication.
- 11 When defining OSPF_Interface IP_address with a wild card (*), all interfaces within the defined range will be seen as OSPF interfaces. Individual definitions with wild cards may be used for seeding other OMPROUTE configuration files. Remember that the unique RouterID of each file makes the entire file unshareable unless MVS system symbolics are employed. (See
- 12 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.
- 13 Stub Areas do not permit importation of OSPF external, direct, or static routes; although the z/OS Communications Server on this node can learn about them, they will not be advertised. Therefore, the AS_Boundary_Routing statement is useless in this configuration and it is commented out. If it is not commented out, it is ignored when the node belongs to a Stub Area or Totally Stubby Area.

Our next example of an OMPROUTE configuration file can be shared across multiple stacks using MVS system symbols. The use of MVS system symbols in the OMPROUTE configuration file is introduced in z/OS V1R9 Communications Server. We have omitted some of the definitions to shorten the example and focus on what is important.

Example 5-17 Shareable OMPROUTE configuration file using MVS system symbols

```
OSPF
   RouterID=10.1.&SYSCLONE..10 1
   Comparison=Type2
   Demand_Circuit=YES;
Global_Options
 Ignore_Undefined_Interfaces=YES
```

```
Routesa_Config Enabled=No;
; Static vipa
OSPF_Interface IP_address=10.1.&SYSCLONE..10 2
          Subnet_mask=255.255.255.0
          Name=VIPA3L
          Attaches_To_Area=0.0.0.2
          Advertise_VIPA_Routes=HOST_ONLY
          Cost0=10
          MTU=65535;
.....
```

This OMROUTE configuration file is now completely shareable. We have fully exploited wildcards and MVS system symbolics.

- 1 We used a MVS system symbol &SYSCLONE value to express the OSPF_Interface that is also used to represent our RouterID of 10.1.0.10 (the &SYSCLONE value resolves to 30 on this LPAR; we used only one digit starting with the second digit of the &SYSCLONE value).
- 2 We used the same SYSCLONE value to define the OSPF_Interface.

Configure routers

In our Cisco routers we created a router service 100, which is analogous to an OMPROUTE procedure (or OSPF process). We defined the OSPF configuration for a Totally Stubby Area under this process.

To configure router 1 (Catalyst™ 6500 series Layer 3 switch), we use the configuration statements shown in Example 5-18.

Example 5-18 Router A configuration statements

```
interface Loopback1
 ip address 10.1.200.1 255.255.255.0
!
interface Vlan10
 ip address 10.1.2.240 255.255.255.0
 ip ospf cost 100
 ip ospf priority 100
!
interface Vlan11
 ip address 10.1.3.240 255.255.255.0
 ip ospf cost 100
 ip ospf priority 100
!
router ospf 100 1
 router-id 10.1.3.240 2
 log-adjacency-changes
 area 2 stub no-summary 3
 network 10.1.2.0 0.0.0.255 area 2 4
 network 10.1.3.0 0.0.0.255 area 2 4
 network 10.1.100.0 0.0.0.255 area 2 4
 network 10.200.1.0 0.0.0.255 area 0 5
 default-information originate always metric-type 1
```

To configure Router 2 (Catalyst 6500 series layer 3 switch), we used the configuration statements shown in Example 5-19.

Example 5-19 Router B configuration statements

```
interface Loopback1
 ip address 10.1.200.2 255.255.255.0
!
interface Vlan10
 ip address 10.1.2.220 255.255.255.0
 ip ospf cost 100
 ip ospf priority 101
!
interface Vlan11
 ip address 10.1.3.220 255.255.255.0
 ip ospf cost 100
 ip ospf priority 101
!
router ospf 100 1
 router-id 10.1.3.220 2
 log-adjacency-changes
 area 2 stub no-summary 3
 network 10.1.2.0 0.0.0.255 area 2 4
 network 10.1.3.0 0.0.0.255 area 2 4
 network 10.1.200.0 0.0.0.255 area 0 5
 default-information originate always metric-type 1
```

The descriptions for the tags shown in Example 5-18 on page 175 and Example 5-19 on page 176 are as follows:

- 1 Designates the process 100 to be an OSPF routing service.
- 2 Defines the router ID of this process (100).
- 3 Creates an OSPF area to this process (Area 2) and defines it to be a Stub Area.
- 4 Defines the network range designated to Area 2. All interfaces within this IP address range (10.10.0.0) will belong to Area 2.
- 5 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.

5.5.2 Activation and verification

To activate and verify the OMPROUTE configuration, perform the following steps:

1. Start OMPROUTE.
2. Verify the configuration.

Start OMPROUTE

OMPROUTE can be started from an z/OS procedure, from the z/OS shell, or from AUTOLOG.

In our test environment we started the OMPROUTE from the z/OS procedure, as shown in Example 5-20.

Example 5-20 OMPROUTE initialization

```
S OMPA
$HASP100 OMPA     ON STCINRDR
IEF695I START OMPA     WITH JOBNAME OMPA     IS ASSIGNED TO USER
TCPIP   , GROUP TCPGRP
$HASP373 OMPA     STARTED
IEE252I MEMBER CTIORA00 FOUND IN SYS1.PARMLIB
EZZ7800I OMPA STARTING 1
EZZ7975I OMPA IGNORING UNDEFINED INTERFACE EZASAMEMVS 2
EZZ7475I ICMP WILL IGNORE REDIRECTS DUE TO ROUTING APPLICATION BEING
```

```
ACTIVE
EZZ8100I OMPA SUBAGENT STARTING
IEA989I SLIP TRAP ID=X13E MATCHED.  JOBNAME=RMFGAT  , ASID=0043.
EZZ7898I OMPA INITIALIZATION COMPLETE
```

- 1 The procedure name OMPA appears in several of the informational messages: EZZ7871I, EZZ975I, and EZZ8100I. This facilitates problem determination in a SYSPLEX or CINET environment with messages flowing to a single console.
- 2 Message EZZ975I shows the effect of “Ignore_Undefined_Interfaces=YES” coding in the OMPROUTE configuration file shown in Example 5-16 on page 172.

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 5-21).

Example 5-21 AUTOLOG statements

```
; ****************************************
; TCPIPA.TCPPARMS(PROFA30)
; ****************************************
.....
AUTOLOG 5
   OMPA ; OMPROUTE procedure
ENDAUTOLOG
.....
```

Verify the configuration

To verify that OMPROUTE is configured correctly as we defined, we can use either the DISPLAY or the MODIFY command.

In this section, we show some of the most useful DISPLAY commands and outputs. To see other display command options and to get more detailed information about specific commands, refer to *z/OS Communications Server: IP System Administrator’s Commands*, SC31-8781.

To display OSPF configuration information, use the Display OMPROUTE,OSPF,LIST,ALL command. The sample display is shown in Example 5-22.

Example 5-22 D TCPIP,TCPIPA,OMP,OSPF,LIST,ALL command display

```
D TCPIP,TCPIPA,OMPR,OSPF,LIST,ALL
EZZ7831I GLOBAL CONFIGURATION 396
    TRACE: 0, DEBUG: 0, SADEBUG LEVEL: 0
    STACK AFFINITY:          TCPIPA
    OSPF PROTOCOL:           ENABLED
    EXTERNAL COMPARISON:     TYPE 2
    AS BOUNDARY CAPABILITY:  DISABLED
    DEMAND CIRCUITS:         ENABLED

EZZ7832I AREA CONFIGURATION
AREA ID          AUTYPE          STUB? DEFAULT-COST IMPORT-SUMMARIES?
0.0.0.2          0=NONE           YES         1             YES         1
0.0.0.0          0=NONE           NO          N/A           N/A

EZZ7833I INTERFACE CONFIGURATION
IP ADDRESS       AREA            COST RTRNS TRDLY PRI HELLO  DEAD DB_E*
10.1.8.10        0.0.0.2           10   N/A   N/A N/A   N/A   N/A  N/A
10.1.6.11        0.0.0.2           90     5     1   1    10    40   40
```

```
10.1.5.11       0.0.0.2          90     5     1   1    10    40    40
10.1.4.11       0.0.0.2          90     5     1   1    10    40    40
10.1.2.12       0.0.0.2         100     5     1   0    10    40    40
10.1.3.12       0.0.0.2         100     5     1   0    10    40    40
10.1.3.11       0.0.0.2         100     5     1   0    10    40    40
10.1.2.11       0.0.0.2         100     5     1   0    10    40    40 2
10.1.2.10       0.0.0.2          10   N/A   N/A N/A   N/A   N/A   N/A
10.1.1.10       0.0.0.2          10   N/A   N/A N/A   N/A   N/A   N/A 3
10.1.30.10      0.0.0.2          10   N/A   N/A N/A   N/A   N/A   N/A

            ADVERTISED VIPA ROUTES
10.1.8.10      /255.255.255.255   10.1.2.10      /255.255.255.255
10.1.1.10      /255.255.255.255   10.1.30.10     /255.255.255.255
```

- 1 The Area 2 is defined as a Totally Stubby Area.
- 2 The OSA interface has Router_Priority=1, Hello_Interal=10, Dead_Interval=40 specified to establish neighbors with other routers.
- 3 The VIPA interface does not have Router_Priority, Hello_Interval, or Dead_Interval specified because they do not establish neighbors.

Display OSPF interfaces

Use the Display OMPROUTE,OSPF,INTERFACE command to display the defined OSPF interfaces and their current status. Our display example is shown in Example 5-23.

Example 5-23 D TCPIP,TCPIPA,OMP,OSPF,INTERFACE command display

```
D TCPIP,TCPIPA,OMPR,OSPF,INTERFACE
EZZ7849I INTERFACES 400
IFC ADDRESS     PHYS         ASSOC. AREA     TYPE   STATE  #NBRS #ADJS

10.1.8.10       VIPL0A01080A 0.0.0.2         VIPA    N/A    N/A    N/A
10.1.6.11       IUTIQDF6L    0.0.0.2         BRDCST   32      4      2
10.1.5.11       IUTIQDF5L    0.0.0.2         BRDCST   32      4      2
10.1.4.11       IUTIQDF4L    0.0.0.2         BRDCST   32      4      2
10.1.2.12       OSA20A0L     0.0.0.2         BRDCST    2      0      0
10.1.3.12       OSA20E0L     0.0.0.2         BRDCST    2      0      0
10.1.3.11       OSA20C0L     0.0.0.2         BRDCST   32      6      2
10.1.2.11       OSA2080L     0.0.0.2         BRDCST   32      6      2 1
10.1.2.10       VIPA2L       0.0.0.2         VIPA    N/A    N/A    N/A
10.1.1.10       VIPA1L       0.0.0.2         VIPA    N/A    N/A    N/A 2
10.1.30.10      VIPA3L       0.0.0.2         VIPA    N/A    N/A    N/A
* -- LINK NAME TRUNCATED
```

- 1 The OSA interface is attached to Area 2 and has six neighbors established. The number of adjacency is two because it only becomes adjacent to DR and BDR.
- 2 The VIPA interface is attached to Area 2 but does not establish neighbors.

Display OSPF neighbors

Use the Display OMPROUTE,OSPF,NBRS command to display the OSPF neighbors and their current status. Our display example is shown in Example 5-24.

Example 5-24 D TCPIP,TCPIPA,OMP,OSPF,NBRS command display

```
D TCPIP,TCPIPA,OMPR,OSPF,NBRS
EZZ7851I NEIGHBOR SUMMARY 392
NEIGHBOR ADDR   NEIGHBOR ID     STATE  LSRXL DBSUM LSREQ HSUP IFC
10.1.3.240      10.1.3.240        128      0     0     0  OFF OSA2OC0L  1
10.1.3.220      10.1.3.220        128      0     0     0  OFF OSA2OC0L  2
10.1.3.22       10.1.31.10          8      0     0     0  OFF OSA2OC0L  3
10.1.2.240      10.1.3.240        128      0     0     0  OFF OSA2080L  1
10.1.2.220      10.1.3.220        128      0     0     0  OFF OSA2080L  2
10.1.2.22       10.1.31.10          8      0     0     0  OFF OSA2080L  3
10.1.4.21       10.1.31.10        128      0     0     0  OFF IUTIQDF4L 4
10.1.5.21       10.1.31.10        128      0     0     0  OFF IUTIQDF5L 4
10.1.6.21       10.1.31.10        128      0     0     0  OFF IUTIQDF6L 4
* -- LINK NAME TRUNCATED
```

- 1 The neighbor with router 1 is established in each VLAN that OSA belongs to. The state is 128 (Full).
- 2 The neighbor with router 2 is established in each VLAN that OSA belongs to. The state is 128 (Full).
- 3 The neighbor with TCPIPB stack is established on the OSA interface. The state is 8 (2-way), because router 1 and router 2 are DR/BDR, so TCPIPA and TCPIPB are both DR other.
- 4 The neighbor with TCPIPB stack is established on the HiperSockets interface. The state is 128 (Full), because TCPIPB is the DR on the HiperSockets subnet.

Display OSPF routers

Use the Display OMPROUTE,OSPF,ROUTERS command to display the OSPF routes to ABRs and ASBRs. Our display example is shown in Example 5-24 on page 179.

Example 5-25 D TCPIP,TCPIPA,OMP,OSPF,ROUTERS command display

```
D TCPIP,TCPIPA,OMPR,OSPF,ROUTERS
EZZ7855I OSPF ROUTERS 402
DTYPE RTYPE DESTINATION       AREA           COST       NEXT HOP(S)
BR  SPF   10.1.3.240       0.0.0.2          100       10.1.2.240 * 1
BR  SPF   10.1.3.220       0.0.0.2          100       10.1.2.220 * 2
```

- 1 router 1 is one of the ABRs.
- 2 router 2 is another ABR.

Display OMPROUTE routing table

Use the Display OMPROUTE,RTTABLE command to display the OMPROUTE routing table. Our display example is shown in Example 5-26 on page 180.

Example 5-26 D TCPIP,TCPIPA,OMP,RTTABLE command display

```
D TCPIP,TCPIPA,OMPR,RTTABLE
EZZ7847I ROUTING TABLE 404
TYPE   DEST NET         MASK      COST    AGE     NEXT HOP(S)

SPIA   0.0.0.0                0  101     116     10.1.2.220       (8) 1
 DIR*  10.1.1.0         FFFFFF00  1       4206    10.1.1.10            2
 DIR*  10.1.1.10        FFFFFFFF  1       4206    VIPA1L               2
 SPF   10.1.1.20        FFFFFFFF  100     222     10.1.4.21        (4) 3
 SPF*  10.1.2.0         FFFFFF00  100     2528    OSA2080L         (2)
 DIR*  10.1.2.10        FFFFFFFF  1       4206    VIPA2L
 SPF   10.1.2.20        FFFFFFFF  100     222     10.1.4.21        (4)
 SPF*  10.1.3.0         FFFFFF00  100     2523    OSA20C0L         (2)
 SPF*  10.1.4.0         FFFFFF00  90      4202    IUTIQDF4L
 SPF*  10.1.5.0         FFFFFF00  90      4202    IUTIQDF5L
 SPF*  10.1.6.0         FFFFFF00  90      4202    IUTIQDF6L
 DIR*  10.1.8.0         FFFFFF00  1       4206    10.1.8.10
 DIR*  10.1.8.10        FFFFFFFF  1       4206    VIPL0A01080A
 SPF   10.1.8.20        FFFFFFFF  100     116     10.1.4.21        (8)
 DIR*  10.1.30.0        FFFFFF00  1       1398    10.1.30.10
 DIR*  10.1.30.10       FFFFFFFF  1       1398    VIPA3L
 SPF   10.1.100.0       FFFFFF00  101     2523    10.1.2.240       (4)

DEFAULT GATEWAY IN USE.

TYPE COST     AGE       NEXT HOP
SPIA 101      116       10.1.2.220       (8)
                       0 NETS DELETED, 3 NETS INACTIVE
```

- 1 Only the default route is advertised from ABR.
- 2 Direct routes to the subnet to which local interfaces belong are listed.
- 3 Indirect routes to the VIPA in TCPIPB are listed.

Display TCP/IP routing table

Use the Netstat command to display the OSPF routes to ABRs and ASBRs. Our display example is shown in Example 5-27.

Example 5-27 D TCPIP,TCPIPA,Netstat,ROUTE command display

```
D TCPIP,TCPIPA,N,ROUTE,MAX=*
EZD0101I NETSTAT CS V1R9 TCPIPA 406
IPV4 DESTINATIONS
DESTINATION         GATEWAY          FLAGS     REFCNT  INTERFACE
DEFAULT             10.1.2.220       UGO       000000  OSA2080L 1
DEFAULT             10.1.2.220       UGO       000000  OSA20A0L
DEFAULT             10.1.2.240       UGO       000001  OSA2080L 1
DEFAULT             10.1.2.240       UGO       000000  OSA20A0L
DEFAULT             10.1.3.220       UGO       000000  OSA20C0L
DEFAULT             10.1.3.220       UGO       000000  OSA20E0L
DEFAULT             10.1.3.240       UGO       000000  OSA20C0L
DEFAULT             10.1.3.240       UGO       000000  OSA20E0L
10.1.1.10/32        0.0.0.0          UH        000000  VIPA1L
10.1.1.20/32        10.1.4.21        UGHO      000000  IUTIQDF4L 2
10.1.1.20/32        10.1.5.21        UGHO      000000  IUTIQDF5L 2
10.1.1.20/32        10.1.6.21        UGHO      000000  IUTIQDF6L 2
10.1.2.0/24         0.0.0.0          UO        000000  OSA2080L
10.1.2.0/24         0.0.0.0          UO        000000  OSA20A0L
10.1.2.10/32        0.0.0.0          UH        000000  VIPA2L
10.1.2.11/32        0.0.0.0          UH        000000  OSA2080L
```

```
10.1.2.12/32       0.0.0.0         UH        000000  OSA2OA0L
10.1.2.20/32       10.1.4.21       UGHO      000000  IUTIQDF4L
10.1.2.20/32       10.1.5.21       UGHO      000000  IUTIQDF5L
10.1.2.20/32       10.1.6.21       UGHO      000000  IUTIQDF6L
10.1.3.0/24        0.0.0.0         UO        000000  OSA2OC0L
10.1.3.0/24        0.0.0.0         UO        000000  OSA2OE0L
10.1.3.11/32       0.0.0.0         UH        000001  OSA2OC0L
10.1.3.12/32       0.0.0.0         UH        000000  OSA2OE0L
10.1.4.0/24        0.0.0.0         UO        000000  IUTIQDF4L
10.1.4.11/32       0.0.0.0         UH        000000  IUTIQDF4L
10.1.5.0/24        0.0.0.0         UO        000000  IUTIQDF5L
10.1.5.11/32       0.0.0.0         UH        000000  IUTIQDF5L
10.1.6.0/24        0.0.0.0         UO        000000  IUTIQDF6L
10.1.6.11/32       0.0.0.0         UH        000000  IUTIQDF6L
10.1.8.10/32       0.0.0.0         UH        000000  VIPLOA01080A
10.1.8.20/32       10.1.4.41       UGHO      000000  IUTIQDF4L
10.1.8.20/32       10.1.5.41       UGHO      000000  IUTIQDF5L
10.1.8.20/32       10.1.6.41       UGHO      000000  IUTIQDF6L
10.1.8.20/32       10.1.4.21       UGHO      000000  IUTIQDF4L
10.1.8.20/32       10.1.5.21       UGHO      000000  IUTIQDF5L
10.1.8.20/32       10.1.6.21       UGHO      000000  IUTIQDF6L
10.1.30.10/32      0.0.0.0         UH        000000  VIPA3L
10.1.100.0/24      10.1.2.240      UGO       000000  OSA2080L
10.1.100.0/24      10.1.2.240      UGO       000000  OSA2OA0L
10.1.100.0/24      10.1.3.240      UGO       000000  OSA2OC0L
10.1.100.0/24      10.1.3.240      UGO       000000  OSA2OE0L
127.0.0.1/32       0.0.0.0         UH        000004  LOOPBACK
IPV6 DESTINATIONS
DESTIP:   ::1/128
  GW:     ::
  INTF:   LOOPBACK6          REFCNT:  000000
  FLGS:   UH                 MTU: 65535
END OF THE REPORT
```

- 1 The default routes to the router 1 and router 2 are listed. We do not have IPCONFIG MULTIPATH specified, so the first active default route entry (interface OSA2080L and gateway address 10.1.2.220) is always used for a destination that does not have an explicit route entry.
- 2 Indirect routes to the VIPA in TCPIPB stack are listed. We do not have IPCONFIG MULTIPATH specified, so the first active route entry (interface IUTIQDF4L) is always used for the VIPA destination.

Check the connectivity using PING command

The PING command can be executed with the TSO PING command or the z/OS UNIX **ping** command. Example 5-6 on page 166 shows the display of the TSO PING command. We see the ping is successful.

In a CINET environment where multiple TCP/IP stacks are configured, use the TCP option for the TSO PING command and the **-p** option for the z/OS UNIX **ping** command to specify the TCP/IP stack name from which you want to issue the ping command.

You do not need to specify those options if the user issuing this command is already associated to the TCP/IP stack (with SYSTCPD DD, for example). There is no need to specify these options if your environment is an INET environment where only one TCP/IP stack is configured.

Example 5-28 TSO PING command display

```
TSO PING 10.1.1.20 (TCP TCPIPA
```

```
CS V1R9: Pinging host 10.1.1.20
 Ping #1 response took 0.000 seconds.
 ***
```

Example 5-29 shows the display of z/OS UNIX **ping** command.

Example 5-29 z/OS UNIX ping command display

```
CS06 @ SC30:/u/cs06>ping -p TCPIPA 10.1.1.20
CS V1R9: Pinging host 10.1.1.20
Ping #1 response took 0.000 seconds.
```

Verify the selected route with the TRACEROUTE command

Traceroute can be invoked by either the TSO TRACERTE command or the z/OS UNIX shell **traceroute/otracert** command. Example 5-36 on page 187 shows an example of the display. We see that the router 1 (10.1.2.240) is the next hop router to reach destination IP address 10.1.100.221.

In a CINET environment where multiple TCP/IP stacks are configured, use the TCP option for the TSO TRACERTE command and the **-a** option for the z/OS UNIX **traceroute** command to specify the TCP/IP stack name from which you want to issue the TRACEROUTE command.

You do not need to specify those options if the user issuing this command is already associated to the TCP/IP stack (with SYSTCPD DD, for example). There is no need to specify those options if your environment is an INET environment where only one TCP/IP stack is configured.

Example 5-30 Tracerte command results

```
TSO TRACERTE 10.1.100.221 (TCP TCPIPA
CS V1R9: Traceroute to 10.1.100.221 (10.1.100.221)
 1 router1 (10.1.2.240)  0 ms  0 ms  0 ms
 2 10.1.100.221 (10.1.100.221)  0 ms  0 ms  0 ms
 ***
```

5.5.3 Managing OMPROUTE

You can manage OMPROUTE from a z/OS operator console. Commands are available to perform the following:

- Stop OMPROUTE.
- Modify OMPROUTE.
- Display OMPROUTE.

Stop OMPROUTE from z/OS console

OMPROUTE can be stopped from the z/OS console by issuing **STOP <procname> or MODIFY <procname>,KILL**.

Example 5-31 shows the example of the display.

Example 5-31 Stopping OMPROUTE from z/OS console

```
P OMPA
EZZ7804I OMPA EXITING
ITT120I SOME CTRACE DATA LOST, LAST 5 BUFFER(S) NOT WRITTEN
```

```
$HASP395 OMPA     ENDED
```

Stop OMPROUTE from z/OS UNIX shell

You can also stop OMPROUTE from a z/OS UNIX shell superuser ID by issuing the **kill** command to the process ID (PID) associated with OMPROUTE. To determine the PID, use one of the following methods:

1. 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 5-32 (1).

Example 5-32 Stopping OMPROUTE from z/OS UNIX

```
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
```

2. From the z/OS UNIX shell, issue the **ps -ef** command, as shown in Example 5-33 (1).

Example 5-33 ps -ef command in z/OS UNIX 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        1  - 14:43:14 ?        4:10 OMPROUTE 1
CS03 @ SC30:/u/cs03>kill 50397483 1
```

Modify 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: Used 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

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, as shown in the following samples:

- **F procname,RTTABLE** command: The resulting display provides us with the same contents as though we were using **D TCPIP,*procnamename*,OMP,RTTABLE**.
- **F *procname*,OSPF,LIST ALL** command: The resulting display provides us the same contents as though we were using **D TCPIP,*procnamename*,OMP,OSPF,LIST ALL**.

Start, stop, or change the level of OMPROUTE tracing and debugging

We can use the MODIFY (F) command to start, stop, or change the level of tracing and debugging.

- **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 Communications Server: IP System Administrator's Commands*, SC31-8781.

5.6 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 in a static routing environment and in a dynamic OSPF routing environment.

To debug a network problem in a z/OS environment, we suggest following the flow shown in Figure 5-4.

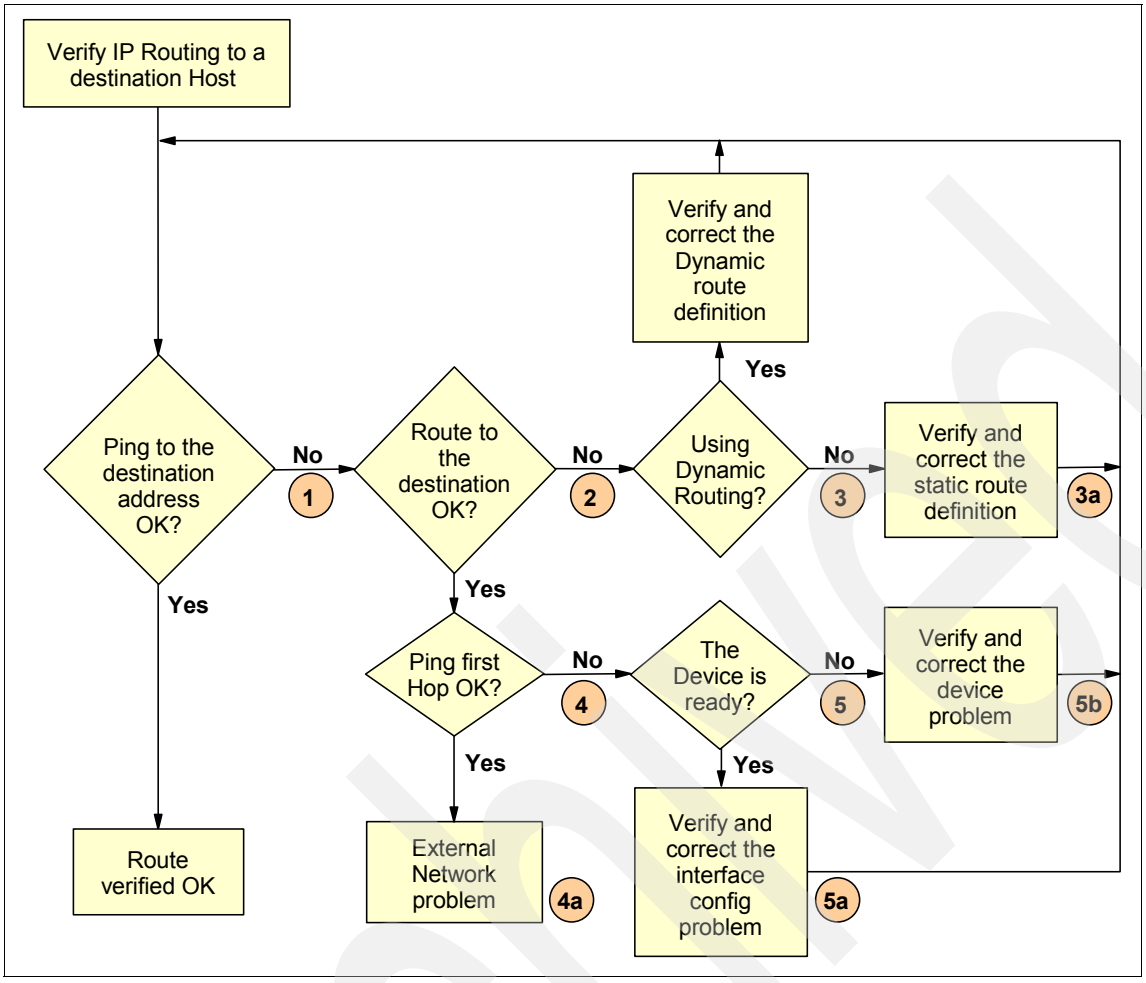


Figure 5-4 Routing problem determination flow

The descriptions for the tags shown in Figure 5-4 on page 185 are as follows:

1. Use the ping command to determine whether there is connectivity to the destination IP address. More information about the ping command can be found in “PING command (TSO or z/OS UNIX)” on page 186.
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 Example 5-5 on page 165. Verify that TCP/IP has a route to the destination address.

   If there is no route, proceed to step 3. If a route does exist, then 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. If it seems correct, then diagnose the problem using the debugging tools described in “Diagnosing an OMPROUTE problem” on page 188.

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, it means 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 that it is operational by issuing the **Netstat Devlink** command, as shown in Example 5-4 on page 164.

   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 that 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.

5.6.1 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 Communications Server: IP System Administrator's Commands*, SC31-8781.

PING command (TSO or z/OS UNIX)

We used the **ping** command to verify:

- The route to the remote network is defined correctly.
- The router is able to forward packets to the remote network.
- The remote host is able to send and receive packets on the network.
- The remote host has a route back to the local host.

The **ping** command can be executed with the TSO PING command or the z/OS UNIX **ping** command. Example 5-34 shows the display of TSO PING command. We see the ping is successful.

In a CINET environment where multiple TCP/IP stacks are configured, use the TCP option for the TSO PING command and the **-p** option for the /OS UNIX **ping** command to specify the TCP/IP stack name from which you want to issue the ping command. You do not need to specify those options if you are issuing this command in the associated TCP/IP stack (with SYSTCPD DD, for example). There is no need to specify this option if your environment is an INET environment where only one TCP/IP stack is configured.

Example 5-34 TSO PING command display

```
TSO PING 10.1.1.20 (TCP TCPIPA
CS V1R9: Pinging host 10.1.1.20
 Ping #1 response took 0.000 seconds.
```

```
***
```

Example 5-35 shows the display of the z/OS UNIX **ping** command.

Example 5-35 z/OS UNIX ping command display

```
CS06 @ SC30:/u/cs06>ping -p TCPIPA 10.1.1.20
CS V1R9: Pinging host 10.1.1.20
Ping #1 response took 0.000 seconds.
```

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 5-36.

In a CINET environment where multiple TCP/IP stacks are configured, use the TCP option for the TSO TRACERTE command and the **-a** option for the z/OS UNIX **traceroute** command to specify the TCP/IP stack name from which you want to issue the TRACEROUTE command.

Note that you do not need to specify those options if the user issuing this command is already associated to the TCP/IP stack (with SYSTCPD DD, for example). There is no need to specify those options if your environment is an INET environment where only one TCP/IP stack is configured.

Example 5-36 TSO Tracerte command results

```
TSO TRACERTE 10.1.100.221 (TCP TCPIPA
CS V1R9: Traceroute to 10.1.100.221 (10.1.100.221)
 1 router1 (10.1.2.240)  0 ms  0 ms  0 ms
 2 10.1.100.221 (10.1.100.221)  0 ms  0 ms  0 ms
 ***
```

Example 5-37 on page 187 shows the display of the z/OS UNIX **traceroute** command.

Example 5-37 z/OS UNIX traceroute command result

```
CS06 @ SC30:/u/cs06>traceroute -a TCPIPA 10.1.100.221
CS V1R9: Traceroute to 10.1.100.221 (10.1.100.221)
Enter ESC character plus C or c to interrupt
1 router1 (10.1.2.240)  0 ms  0 ms  0 ms
2 10.1.100.221 (10.1.100.221)  0 ms  0 ms  0 ms
```

Tip: Using a name instead of IP address would need the Resolver or DNS to do the translation. This adds more variables to the problem determination, and should be avoided when you are diagnosing network problems. Use the host IP address instead.

Netstat Devlink command (console or z/OS UNIX)

Use the D TCPIP,*procname*,NETSTAT, DEVLINK command to display the status and associated configuration values for a device and its defined interfaces. From the z/OS UNIX shell, use the **netstat -d -p *procname*** command. The results are identical in the console or the z/OS UNIX shell. Example 5-4 on page 164 shows a sample display.

Netstat Route command (console or z/OS UNIX)

Use the D TCPIP,*procname*,NETSTAT,ROUTE command to display the current routing tables for TCP/IP. From z/OS UNIX shell, use the **netstat -r -p *procname*** command. Example 5-27 on page 180 shows a sample display.

5.6.2 Diagnosing an OMPROUTE problem

Useful commands

In addition to the command we show in 5.6.1, “Commands to diagnose networking connectivity problems” on page 186, you can use additional commands to diagnose OMPROUTE problems, as described here.

D TCPIP,TCPIPA,OMP,OSPF,NBRS command

This command displays all the OSPF neighbors. Make sure you have established the neighbor with other routers as you expected. Example 5-24 on page 179 shows a sample display.

D TCPIP,TCPIPA,OMP,RTTABLE command

This command displays the OMPROUTE routing table. Make sure you have the expected route listed in the table. If you have multiple routes for the destination, with different costs, only the best route (least cost route) is added to the OMPROUTE and TCP/IP routing tables. Example 5-26 on page 180 shows a sample display.

D TCPIP,TCPIPA,OMP,RTTABLE,DELETED command

This command displays all of the route destinations that have been deleted from the OMPROUTE routing table since the initialization of OMPROUTE at this node. The routes that have changed the next hop are *not* considered deleted, and are therefore *not displayed* with this command. Example 5-38 shows the results of this display after OMPROUTE is terminated at SC31 (OMPB), another member of the SYSPLEX.

Example 5-38 Deleted OMPROUTE destinations

```
D TCPIP,TCPIPA,OMPR,RTTABLE,DELETED
EZZ8137I IPV4 DELETED ROUTES 182
TYPE   DEST NET         MASK      COST    AGE     NEXT HOP(S)

DEL    10.1.1.20        FFFFFFFF  16      66      NONE
DEL    10.1.2.20        FFFFFFFF  16      66      NONE
DEL    10.1.8.20        FFFFFFFF  16      66      NONE
       3 NETS DELETED, 2 NETS INACTIVE
```

Observing initialization messages and taking traces

If these commands do not help, use traces for further diagnosis and verify whether you 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 you to solve the problem, then prepare OMPROUTE to generate more detailed information by using the debug tools available in OMPROUTE. This can be activated by coding the Debug and Trace options in the startup procedure, or by using the MODIFY command to implement these options.

Using OMPROUTE trace and debug for initialization

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 faster 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 5-39.

Example 5-39 Trace options defined in the OMPROUTE startup 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')
//*
//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:
  - MODIFY omproute,TRACE=2 (TRACE6=2 for IPv6)
  - MODIFY omproute,DEBUG=1 (DEBUG6=1 for IPv6)
- To disable:
  - MODIFY omproute,TRACE=0 (TRACE6=0 for IPv6)
  - MODIFY omproute,DEBUG=0 (DEBUG6=0 for IPv6)

Trace output is sent to one of the following locations:

- A destination referenced by the 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 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 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 occur when using z/OS UNIX file system files to save the trace and debug output data can be resolved by using the CTRACE facility. To use this facility, we strongly recommend using the 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.

You can start the OMPROUTE CTRACE anytime 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.

In addition to specifying the trace options, you can also change the OMPROUTE trace buffer size. (Note that 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.

In this section we illustrate the steps needed to start the CTRACE for OMPROUTE and direct the trace output to an external writer:

1. Create a CTWTR procedure in your SYS1.PROCLIB, as shown in Example 5-40.

Example 5-40 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 5-41 shows a sample of CTIORA00 contents.

Example 5-41 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 5-41 on page 190 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 5-42.

Example 5-42 OMPROUTE 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')        a
//*
//STDENV   DD DISP=SHR,DSN=TCPIPA.OMPROUTE.&STDENV
```

The description for the tag shown in Example 5-42 on page 191 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 5-43.

Example 5-43 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 5-44.

Example 5-44 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 5-45.

Example 5-45 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 5-46.

Example 5-46 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 5-47.

Example 5-47 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, as shown in Example 5-48, to avoid performance problems using the MODIFY command.

Example 5-48 Modifying Debug and Trace level

```
F OMPA,TRACE=0
EZZ7866I OMPROUTE MODIFY COMMAND ACCEPTED
F OMPA,DEBUG=0
EZZ7866I OMPROUTE MODIFY COMMAND ACCEPTED
```

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 5-49.

Example 5-49 Formatting the OMPROUTE CTRACE

```
 CTRACE COMP(SYSTCPRT) FULL
```

The resulting display will show the Resolver process entries, as shown in Example 5-50.

Example 5-50 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)
```

For more information about OMPROUTE diagnosis, refer to *z/OS Communications Server: IP Diagnosis Guide*, GC31-8782.

5.7 For additional information

For more details on these topics, refer to:

- *z/OS Communications Server: IP Configuration Guide*, SC31-8775
- *z/OS Communications Server: IP Configuration Reference*, SC31-8776
- *z/OS Communications Server: IP Diagnosis Guide*, GC31-8782

6

Virtual Medium Access Control (VMAC) support

Virtual Medium Access Control (VMAC) support for z/OS Communications Server is a function that effects the operation of an OSA interface at the OSI layer 2 level - the Data Link Control (DLC) layer with its sub-layer Medium Access Control (MAC) layer.

VMAC support enables an OSA interface to have not only a physical MAC address, but also distinct virtual MAC addresses for each device or interface in a stack. That is, each stack may define one VMAC per protocol (IPv4 or IPv6) for each OSA interface.

With the use of VMACs, decisions at the layer 3 level, that is, the network layer (or IP layer) can be made for forwarding and routing of IP packets entering the sysplex environment via the OSA.

From a LAN perspective, the OSA interface with a VMAC appears as a dedicated device or interface to a TCP/IP stack.

This chapter discusses the following topics.

Section	Topic
6.1, “Virtual MAC overview” on page 196	The VMAC concept, and the environment in which it can be used
6.2, “Virtual MAC implementation” on page 199	An implementation example of VMACs
6.3, “References” on page 202	Sources of further information regarding VMACs

6.1 Virtual MAC overview

Prior to the introduction of the *virtual* MAC function, an OSA interface only had one MAC address. This restriction caused problems when using load balancing technologies in conjunction with TCP/IP stacks that share OSA interfaces.

The single MAC address of the OSA also causes a problem when using TCP/IP stacks as a forwarding router for packets destined to unregistered IP addresses.

With the use of the VMAC function, packets destined for a TCP/IP stack are identified by an assigned VMAC address and packets sent to the LAN from the stack use the VMAC address as the source MAC address. This means that all IP addresses associated with a TCP/IP stack are accessible via their own VMAC address, instead of sharing a single physical MAC address of an OSA interface.

6.1.1 Why use virtual MACs

A shared OSA environment can be a challenge in certain network designs, and it requires careful planning when selecting the correct TCP/IP stacks to act as routers.

As mentioned in “OSA-Express router support” on page 106, the PRIRouter and SECRouter functions enable routing through a TCP/IP stack to IP addresses that are not registered in the OSA. The stack that has the OSA interface defined with PRIRouter will receive packets destined for IP addresses that do not reside in the given stack. The stack will then forwards the packets to the next hop.

Only one PRIRouter can be defined per OSA interface, although multiple SECRouters can be defined to an OSA interface for other TCP/IP routing stacks. However, only one SECRouter function can take over services if the PRIRouter is not available. If the first SECRouter function is not available, then the next defined SECRouter will forward IP packets to the associated stack. This means the OSA interface cannot serve multiple TCP/IP routing stacks concurrently even with the use of the PRIRouter and SECRouter functions.

Another challenge with shared OSA interfaces is one that requires load balancing of traffic across multiple TCP/IP stacks and IP addresses. For example, certain load balancing technologies use a concept of distributing packets to the appropriate adjacent systems based on knowledge of the MAC address.

We use load balancing (LB) with Sysplex Distributor to illustrate this challenge. If there is a shared OSA environment, the MAC address is attached to the Sysplex Distributor and to the selected target system. However, the target IP address may reside on a system other than the Sysplex Distributor.

As a result, the LB forwarding agent sends the packets to be distributed to the OSA’s physical MAC address, but the OSA only knows to send the information to the system that has registered the target address; it does not know to forward the information to the actual target stack. Mechanisms that are in place to overcome this challenge are Generic Resource Encapsulation (GRE) and Network Address Translation (NAT).

VMAC is a solution for both these problems and we recommend defining VMAC whenever multiple TCP/IP stacks share an OSA interface. VMAC support can provide the following:

- Allow for multiple concurrent TCP/IP routing stacks sharing an OSA interface
- Simplify the LAN infrastructure
- Eliminate the need for PRIRouter/SECRouter
- Improve outbound routing

- Improve IP workload balancing
- Remove the dependency on GRE and NAT

Note that there are two modes that can be used with load balancing technologies:

1. *Directed mode* is where the load balancer converts the destination IP address (cluster IP address) to an IP address owned by the target system, using NAT. When IP packets from the target system are sent back to the clients, the load balancer converts the source IP address back to the cluster IP address. Therefore, the packets must return through the same load balancer that will recognize the changes and do the reverse mapping to ensure packets can flow from the original destination to the original source.
2. *Dispatch mode* does not convert IP addresses, therefore eliminating the need for performing NAT. This mode requires VMAC support if the target stacks share the OSAs. In addition, all target applications must bind to the IP address specified by INADDR_ANY (or in6addr_any for IPv6), and the cluster IP address must be defined to the stack. The cluster IP address must not be advertised through a dynamic routing protocol; otherwise some systems may not get work routed to them. This can be done by defining the cluster IP address in the HOME list as a loopback address.

For more information regarding load balancing modes (directed and dispatch), refer to *z/OS Communications Server: IP Configuration Guide*, SC31-8775.

6.1.2 Virtual MAC concept

Figure 6-1 on page 198 depicts how the definition of VMACs in the TCP/IP stacks gives the appearance of having a dedicated OSA interface on each stack. When packets arrive at the shared OSA interface, the individual VMAC assignments allow the packets to be forwarded directly to the correct stack. In the example shown, no individual stack needs to be defined as a primary or secondary router, thus offloading this function from a TCP/IP stack.

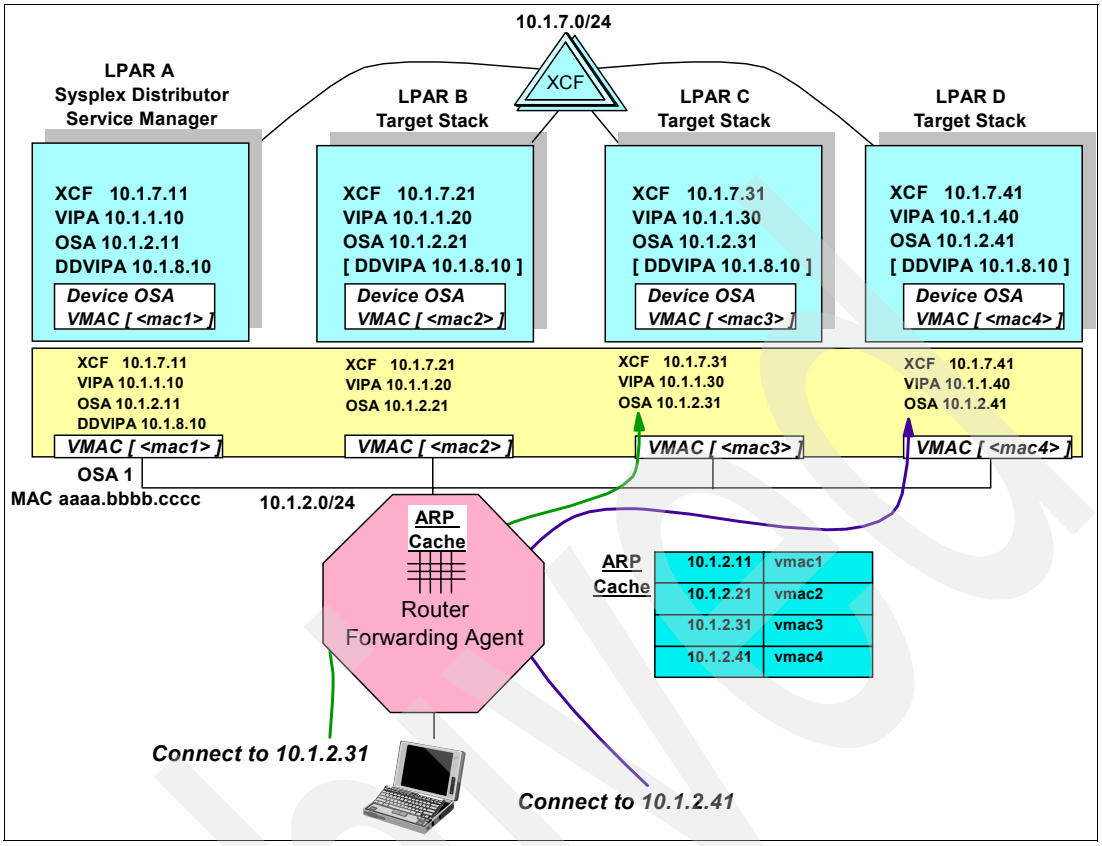


Figure 6-1 Forwarding packets to VMAC targets

This simplifies a shared OSA configuration significantly. Defining VMACs has very little administrative overhead. It is also an alternative to GRE or NAT when load balancing technologies are used. In Figure 6-1, the Dynamic VIPA targets are found without the use of GRE and without routing through the Sysplex Distributor. One of the options for defining VMACs permits the OSA to bypass IP address lookup. As a result, when the packet arrives at the correct VMAC, it is routed to the stack even though the DDVIPA is not registered in the OAT.

For IPV6, TCP/IP uses the VMAC address for all neighbor discovery address resolution flows for that stack's IP addresses, and likewise uses the VMAC as the source MAC address for all IPv6 packets sent from that stack. Again, from a LAN perspective, the OSA interface with a VMAC appears as a dedicated device to that stack.

Note: VMAC definitions on a device in a TCP/IP stack override any NONRouter, PRIRouter, or SECRouter parameters on devices in a TCP/IP stack. If necessary, selected stacks on a shared OSA may define the device with VMAC and others may define the device with PRIRouter and SECRouter capability.

6.1.3 Virtual MAC address assignment

The VMAC address can be defined in the stack, or generated by the OSA. If generated by the OSA, it is guaranteed to be unique from all other physical MAC addresses and from all other VMAC addresses generated by any OSA-Express feature.

Note: We recommend letting the OSA generate the VMACs instead of assigning an address in the TCP/IP profile. If VMACs are defined in the LINK statement, they must be defined as locally administered MAC addresses, and should be unique to the LAN on which they reside.

The same VMAC can be defined for both IPv4 and IPv6 usage, or a stack can use one VMAC for IPv4 and one for IPv6. Also, a VLAN ID can be associated with an OSA-Express device or interface defined with a VMAC.

6.2 Virtual MAC implementation

In this section, we show a scenario using VMAC as a replacement for PRIRouter and SECRouter. However the same implementation would apply to an environment using load balancing technologies. For details regarding load balancing technologies, refer to *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 3: High Availability, Scalability, and Performance*, SG24-7534.

When implementing VMAC support, keep in mind the following points:

- The VMAC function is only available for OSA interfaces configured in QDIO mode.
- Each stack may define one VMAC per protocol (IPv4 or IPv6) for each OSA interface.
- If a VMAC is defined, the stack will not receive any packets destined to the physical MAC.
- VLAN IDs also apply to VMACs like physical MACs.
- If OSAs are not shared, VMACs are not necessary.
- Allow the OSA to generate VMAC addresses.
- When configuring VMACs to solve load balancing issues, remember to:
  - Remove GRE tunnels as appropriate.
  - Change external load balancer configurations (such as directed mode to dispatch mode).

Note: VMAC support is only available with the IBM System z9 Enterprise Class or Business Class servers. For more information, see the 2094DEVICE and 2096DEVICE Preventive Service Planning (PSP) buckets.

6.2.1 IP routing when using VMAC

In our scenario, as illustrated in Figure 6-2 on page 200, we define VMACs to make two TCP/IP stacks act as forwarding stacks to route unregistered IP addresses, using OMPRoute. TCPIPA and TCPIPC share an OSA interface. We configured TCPIPA to forward packets to TCPIPB, and we configured TCPIPC to forward packets to TCPIPD.

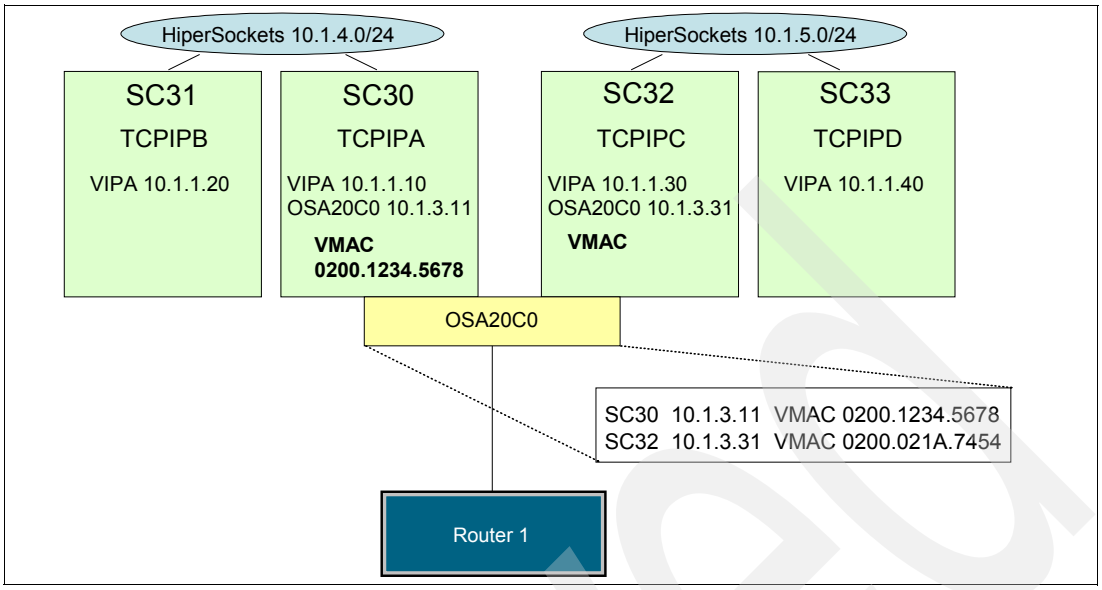


Figure 6-2 IP Routing using VMAC

We omitted the DEVICE, LINK, and HOME statements for OSA20C0 on TCPIPB and TCPIPD, and modified the IP routing definitions on all stacks.

Note that Figure 6-2 is used only for demonstration purposes. We do *not* recommend implementing any configuration with single-points-of-failure.

Configuring the VMAC

The VMAC is defined on the LINK statement in the TCP/IP profile. Example 6-1 and Example 6-2 show the VMAC definitions for TCPIPA and TCPIPC. In our example, we defined VMAC for OSA with VLAN ID. However, VLAN ID is not a prerequisite.

Example 6-1 Device and link statements - VMAC definition for TCPIPA

```
DEVICE OSA20C0    MPCIPA
LINK   OSA20C0L   IPAQENET    OSA20C0 VLANID 11 VMAC 020012345678 1
DEVICE IUTIQDF4   MPCIPA
LINK   IUTIQDF4L  IPAQIDIO    IUTIQDF4
DEVICE VIPA1      VIRTUAL 0
LINK   VIPA1L     VIRTUAL 0   VIPA1
```

If VMAC is defined without a MAC address 2, then OSA generates a VMAC using a part of the “burned-in” MAC address of the OSA. You may also specify the MAC address for VMAC 1. If you decide to specify a MAC address, it must be a locally administered address, which means bit 6 of the first byte is 1 and bit 7 of the first byte is 0.

Example 6-2 Device and link statements - VMAC definition for TCPIPC

```
DEVICE OSA20C0    MPCIPA
LINK   OSA20C0L   IPAQENET    OSA20C0 VLANID 11 VMAC 2
DEVICE IUTIQDF5   MPCIPA
LINK   IUTIQDF5L  IPAQIDIO    IUTIQDF5
DEVICE VIPA1      VIRTUAL 0
LINK   VIPA1L     VIRTUAL 0   VIPA1
```

There is no need to define PRIRouter or SECRouter on the DEVICE statement. When VMAC is specified on LINK statement, PRIRouter or SECRouter is ignored.

6.2.2 Verification

We verified that VMAC was correctly defined in TCPIPA (see Example 6-3). We specified a MAC address 1 for the OSA in TCPIPA, so VMACORIGIN is CFG 2.

Example 6-3 Display VMAC on TCPIPA

```
D TCPIP,TCPIPA,N,DEV
DEVNAME: OSA20C0           DEVTYPE: MPCIPA
  DEVSTATUS: READY
  LNKNAME: OSA20C0L          LNKTYPE: IPAQENET   LNKSTATUS: READY
    NETNUM: N/A  QUESIZE: N/A  SPEED: 0000000100
    IPBROADCASTCAPABILITY: NO
    VMACADDR: 020012345678 1 VMACORIGIN: CFG 2 VMACROUTER: ALL
```

We verified that VMAC was correctly defined in TCPIPC (see Example 6-4). Because we did not specify a MAC address for the OSA in TCPIPC, the OSA generated the MAC address 3. Because this is a OSA-generated MAC address, VMACORIGIN is OSA 4.

Example 6-4 Display VMAC on TCPIPC

```
D TCPIP,TCPIPC,N,DEV
DEVNAME: OSA20C0           DEVTYPE: MPCIPA
  DEVSTATUS: READY
  LNKNAME: OSA20C0L          LNKTYPE: IPAQENET   LNKSTATUS: READY
    NETNUM: N/A  QUESIZE: N/A  SPEED: 0000000100
    IPBROADCASTCAPABILITY: NO
    VMACADDR: 0200021A7454 3 VMACORIGIN: OSA 4 VMACROUTER: ALL
```

We can also see the VMAC in the OSA Address Table (OAT) queried by OSA/SF (see Example 6-5). OSA registers all IP addresses (including VIPA) in the TCP/IP stack, and maps them to the VMAC address.

Note that the last three bytes of the OSA-generated VMAC 7 are identical to that of the universal MAC address (“burned-in” address) of the OSA 5. The first byte of the OSA-generated VMAC is always 02, in order to make the VMAC a locally administered address. To make the VMAC unique among all TCP/IP stacks, the second and third bytes are used as a counter that is incremented each time OSA generates a MAC address.

Example 6-5 Display OAT queried with OSA/SF

```
c-Local MAC address --------------> 00096B1A7454
  Universal MAC address ----------> 00096B1A7454 5

************************************************************************
                              Image 2.3 (A23     ) CULA 0
00(20C0)* MPC       N/A     OSA20C0P   (QDIO control)        SIU    ALL
02(20C2)  MPC  00 No4  No6  OSA20C0P   (QDIO data)           SIU    ALL
                VLAN   11   (IPv4)

                VMAC            IP address
          REG   020012345678 6  010.001.001.010
          REG   020012345678    010.001.003.011
          REG   020012345678    010.001.004.011

************************************************************************
                              Image 2.5 (A25     ) CULA 0
00(20C0)* MPC       N/A     OSA20C0P   (QDIO control)        SIU    ALL
02(20C2)  MPC  00 No4  No6  OSA20C0P   (QDIO data)           SIU    ALL
                VLAN   11   (IPv4)
```

```
        VMAC            IP address
REG     0200021A7454 7  010.001.001.030
HOME    0200021A7454    010.001.003.031
REG     0200021A7454    010.001.005.031
```

Example 6-6 shows the ARP cache of the router. IP address 10.1.3.11 in TCPIPA is mapped to the VMAC defined in TCPIPA 8, and IP address 10.1.3.31 in TCPIPC is mapped to the VMAC defined in TCPIPC 9.

Note that each IP address is mapped to a different MAC address, even if these stacks share the same OSA interface. OSA responds to ARP requests for all registered IP addresses by using a VMAC instead of a “burned-in” MAC address.

According to the routing table, the router chooses 10.1.3.11 as the next hop for destination address 10.1.1.20, and chooses 10.1.3.31 as the next hop for destination address 10.1.1.40. The router forwards the packet with the destination IP address 10.1.1.20 to the destination MAC address 0200.1234.5678. When the packet reaches the OSA interface, OSA forwards the packet to TCPIPA, because OSA knows the VMAC 200.1234.5678 is mapped to TCPIPA. The same can be said for the TCPIPC VMAC.

Example 6-6 Display ARP cache in Router 1

```
Router1#sh arp
Internet  10.1.3.11             10   0200.1234.5678 8 ARPA   Vlan11
Internet  10.1.3.31             20   0200.021a.7454 9 ARPA   Vlan11
```

Example 6-7 shows that the two stacks (TCPIPA and TCPIPC) sharing one OSA interface are able to route packets correctly.

Example 6-7 Display traceroute from Router 1

```
Router1#traceroute 10.1.1.20
  1 10.1.3.11 0 msec 0 msec 0 msec
  2 10.1.1.20 0 msec 0 msec 0 msec

Router1#traceroute 10.1.1.40
  1 10.1.3.31 4 msec 0 msec 0 msec
  2 10.1.1.40 0 msec 0 msec 0 msec
```

6.3 References

For more information about the VMAC function, refer to the following documentation:

- *z/OS Communications Server: IP Configuration Guide*, SC31-8775
- *z/OS Communications Server: IP Configuration Reference*, SC31-8776
- *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 3: High Availability, Scalability, and Performance*, SG24-7534

7

Sysplex subplexing

In large sysplex environments, there may be strict security requirements to isolate access to certain VTAM nodes or TCP/IP stacks within the sysplex. A Communications Server for z/OS function, called *subplexing*, provides this type of support. It enables the user to implement automatically controlled access to subplex groups.

As mentioned, the subplexing support is also for VTAM nodes. However, this chapter only describes subplexing for TCP/IP stacks. For information about VTAM subplexing, refer to *SNA Network Implementation Guide,* SC31-8777.

This chapter discusses the following topics.

Section	Topic
7.1, “Introduction” on page 204	The subplexing concept, and the environment in which it can be used
7.2, “Subplex environment” on page 206	Our TCP/IP subplexing environment
7.3, “Subplex implementation” on page 207	Implementation examples of TCP/IP subplexing

7.1 Introduction

Prior to subplexing, VTAM and TCP/IP sysplex functions were deployed sysplex-wide and users had to implement complex resource controls and disable many of the dynamic XCF and routing functions to support multiple security zones. For example, as shown in Figure 7-1, TCP/IP stacks access different networks with diverse security requirements within the same sysplex:

- In the upper configuration, two TCP/IP stacks in the left LPARs access an internal network. The TCP/IP stacks in the right two LPARs access the external network. Presumably, the security requirements would include isolating external traffic from the internal network. However, all TCP/IP stacks in the sysplex can dynamically establish connectivity with all the other TCP/IP stacks in the sysplex.
- In the lower configuration, TCP/IP stacks in the same LPAR have different security requirements. The first stack in each LPAR connects to the internal network, and the second stack connects to the external network. Through the IUTSAMEH connection, the two stacks in each LPAR can dynamically establish connectivity with each other, therefore possibly violating security policies.

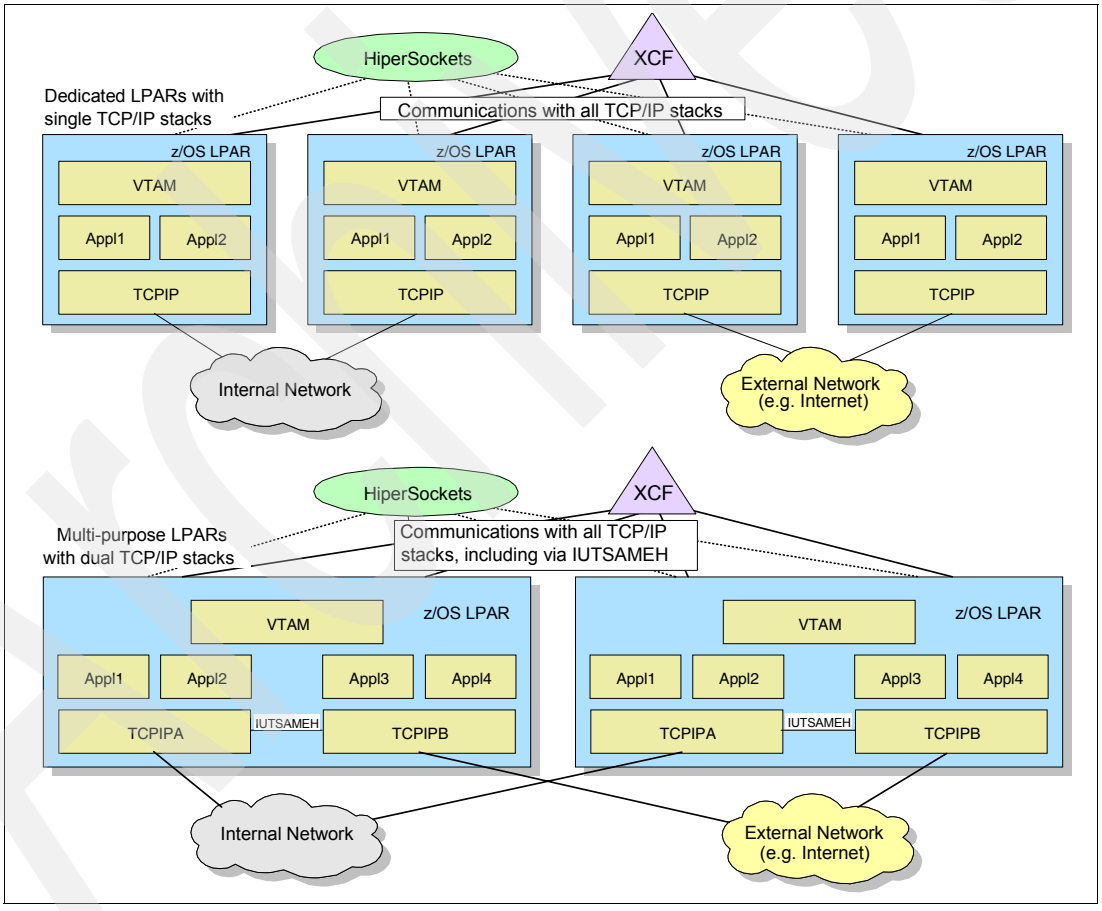


Figure 7-1 Sysplex connectivity - examples

With subplexing, you are able to build “security zones”. Thus, only members within the same security zone may communicate with each other. Subplex members are VTAM nodes and TCP/IP stacks that are grouped in security zones to isolate communication.

Concept of subplexing

A *subplex* is a subset of a sysplex that consists of selected members. Those members are connected and they communicate through dynamic cross-system coupling facility (XCF) groups to each other, using the following methods:

- XCF links (for cross-system IP and VTAM connections)
- IUTSAMEH (for IP connections within an LPAR)
- HiperSockets (IP connections cross-LPAR in the same server)

Subplexes do not communicate with members outside the subset of the sysplex. For example, in Figure 7-2, TCP/IP stacks with connectivity to the internal network can be isolated from TCP/IP stacks connected to the external network using subplexing.

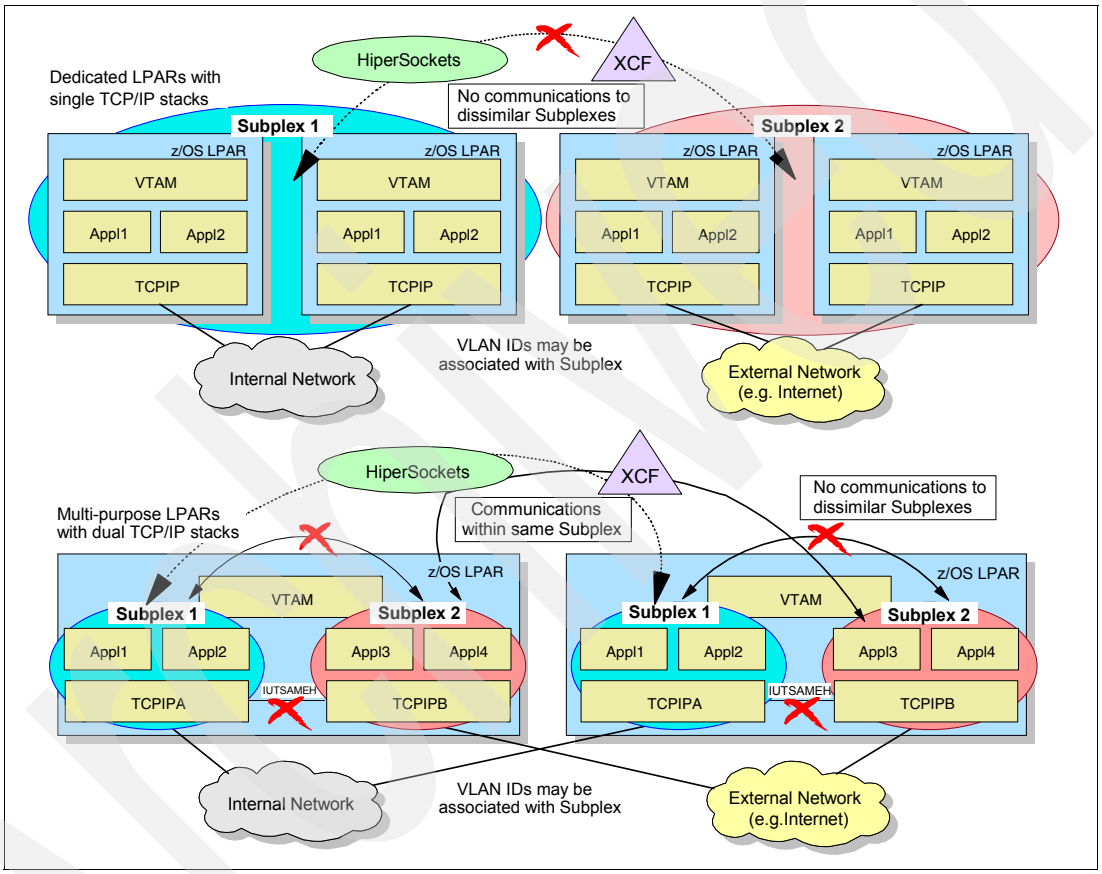


Figure 7-2 Subplexing multiple security zones

TCP/IP stacks are defined as members of a subplex group with a defined group ID. For example, in Figure 7-2 TCP/IP stacks within subplex 1 are able to communicate only with stacks within the same subplex group. They are not able to communicate with stacks in subplex 2.

In an environment where a single LPAR has access to internal and external networks via two TCP/IP stacks, those stacks are assigned to two different subplex group IDs. Even though IUTSAMEH is the communication method, it would be controlled automatically through the association of subplex group IDs, thus creating two separate security zones within the LPAR.

Recommendation: Network connectivity provided through an OSA port in a multiple security zone environment should *not* be shared across different subplex groups. The OSA ports and HiperSockets connections should be physically isolated or logically separated using firewall and VLAN technologies.

7.2 Subplex environment

In this section we describe the environment used to demonstrate subplexing in a multiple security zone, based on Figure 7-2 on page 205. All LPARs in our scenarios were configured in a single server with multiple stacks for demonstration purposes only.

Note: Although there are specialized cases where multiple stacks per LPAR can provide value, we generally recommend implementing only one TCP/IP stack per LPAR whenever possible.

Figure 7-3 illustrates our TCP/IP subplexing environment with the following attributes:

- The first subplex is a VTAM subplex, which is not within the scope of this book. However, when defining only a TCP/IP subplex, a default VTAM subplex is defined automatically.

  Note: A TCP/IP subplex uses VTAM XCF support for DYNAMICXCF connectivity. Therefore, a TCP/IP stack cannot span different VTAM subplexes.

- The second subplex was configured with TCP/IP C stacks running in LPARs A23 and A24, representing the internal subplex.
- The third subplex was configured with TCP/IP D stacks running in LPARs A24 and A25, representing the external subplex.

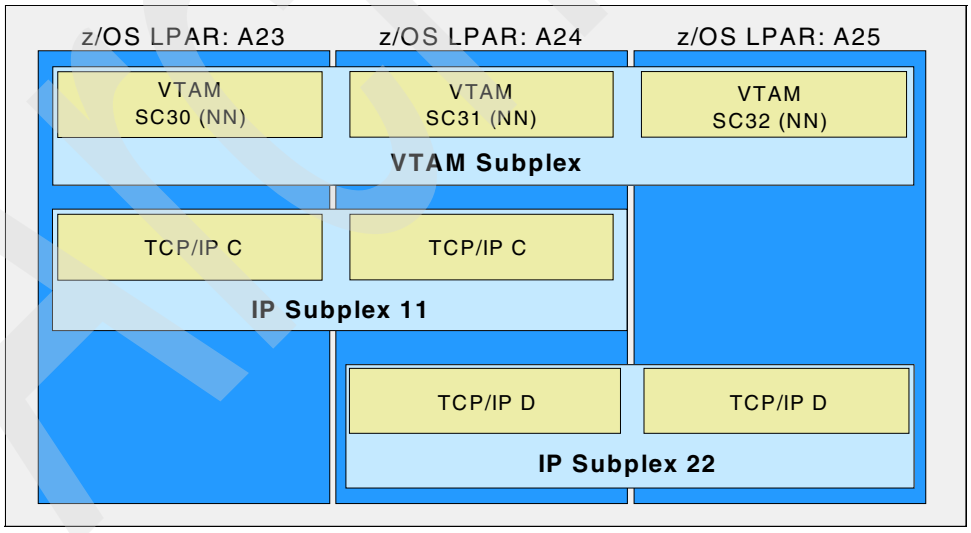


Figure 7-3 Our TCP/IP subplexing environment

We do not describe or discuss OSA connectivity in this chapter. For details regarding OSA functions and configuration information refer to Chapter 4, “Connectivity” on page 101.

7.3 Subplex implementation

TCP/IP stacks in the sysplex must be at z/OS V1R8 or later under the following conditions:

- Complete isolation between TCP/IP stacks in different subplexes is required.
- HiperSockets are used in support of dynamic XCF connectivity for TCP/IP stacks in a subplex.
- TCP/IP stacks in different subplexes accessing HiperSockets with the same CHPID.

An IP subplex is built through association of selected TCP/IP stack members to an XCF group. This is done by defining the XCFGRPID parameter in the GLOBALCONFIG statement of the TCP/IP profile. The subplex is created automatically at the start of the first stack member using this XCFGRPID definition plus the dynamic XCF IP address taken from the IPCONFIG statement DYNAMICXCF.

If the IP traffic for a defined subplex uses HiperSockets, which is the recommended method for cross-LPAR connectivity within the same server, then an additional parameter (IQDVLANID) in the GLOBALCONFIG is needed for the HiperSockets VLAN ID of the HiperSockets connection built with the DYNAMICXCF definition. Values from 2 to 31 are valid for XCFGRPID, while IQDVLANID allows values from 1 to 4094. If defining HiperSockets with DEVICE and LINK statements, the parameter VLANID on the LINK statement is required for assigning the VLAN for the subplex.

Requirement: A z890 or z990 at GA2 hardware level, or a z9 EC or z9 BC, is required to support VLAN IDs on HiperSockets.

Figure 7-4 depicts our subplexing environment. It shows three LPARs with a VTAM subplex, and two IP subplexes 11 and 22. Because we did not define the VTAM subplex, the XCFGRPID value for the VTAM subplex automatically defaults to CP.

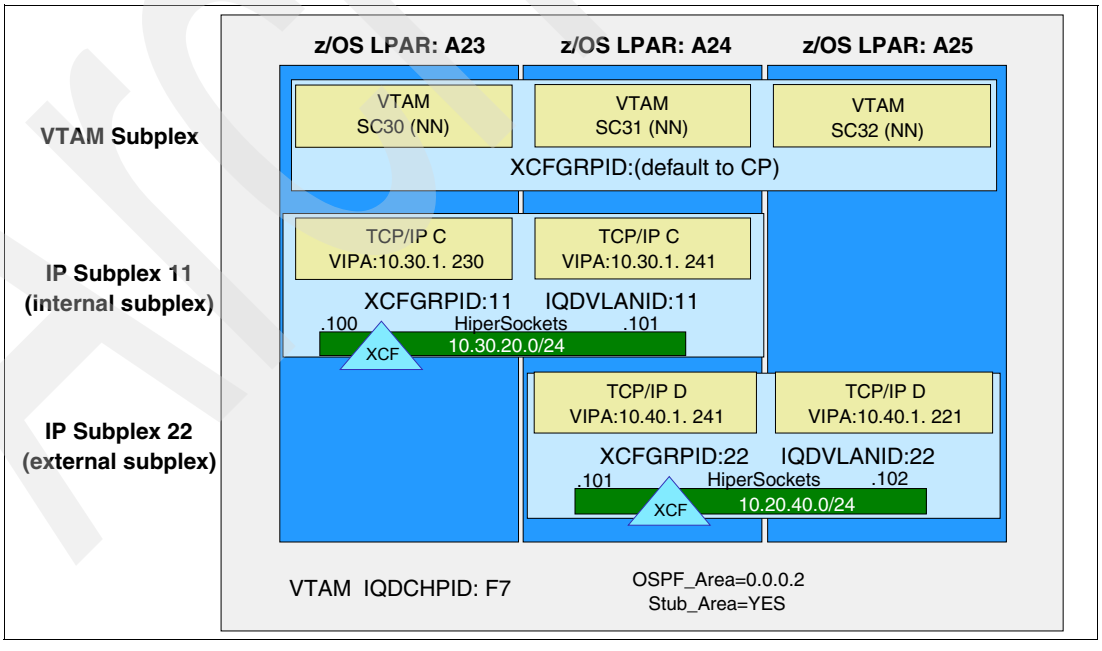


Figure 7-4 Subplex configuration

XCF group names

Basically, XCF group names for subplexes are created through the XCFGRPID parameter for the VTAM and TCP/IP environment; for example:

- For defining a VTAM subplex, use the XCFGRPID parameter in the VTAM start option. For detailed information about group and structure names, refer to *SNA Network Implementation Guide,* SC31-8777.
- For defining a TCP/IP subplex, use the XCFGRPID parameter on the GLOBALCONFIG statement in the TCP/IP profile.

For TCP/IP, both the VTAM group ID suffix and the TCP group ID suffix will be used to build the TCP/IP group name. This group name is also used to join the sysplex. Remember, when starting TCP/IP under Sysplex Autonomics control in previous z/OS releases, the stack joined the sysplex group with the name EZBTCPCS. You can check this using the D XCF,GROUP command.

EZBTCPCS is the default TCP/IP group name. The format of this group name is EZBTvvtt. The meanings of the last four characters are as follows:

- vv is a 2-digit VTAM group ID suffix, specified on the VTAM XCFGRPID start option. The default is CP if not specified.
- tt is a 2-digit TCP group ID suffix, specified on the XCFGRPID parameter of the GLOBALCONFIG statement. The default is CS if not specified.

In our scenario (see Example 7-3 on page 210 3), we defined XCFGRPID 11 for TCP/IP and we did not define an XCFGRPID for VTAM. The result was an XCF group name of EZBTCP11 (Example 7-4 on page 211 6).

You may recognize that both XCFGRPIDs are important in creating the subplex group name. Be aware that changing the VTAM XCFGRPID will change the XCF group name for the TCP/IP stack. Hence, the stack will no longer be a member of the previous TCP/IP subplex group.

For example, in our environment no VTAM XCFGRPID was defined and XCFGRPID 11 was specified for TCP/IP. Therefore, the XCF group name was dynamically built as EZBTCP11. If we add XCFGRPID=02 to the VTAM start option, then the new XCF group name will be EZBT0211.

Although nothing has been changed in the TCP/IP profile definitions in this example, the TCP/IP stack with the new subplex group name is no longer a member of the previous subplex (EZBTCP11). Thus, the TCP/IP stack will lose the connectivity to the subplex.

Important: If VTAM is brought down and restarted with a different XCFGRPID, the TCP/IP stacks must be stopped and restarted to pick up the new VTAM subplex group ID. Otherwise, the TCP/IP stacks will continue to act as if there were in the original sysplex group, resulting in unpredictable connectivity.

TCP/IP structures

This section is only important for TCP/IP implementations using functions for Sysplex-wide Security Associations (SWSA) or for SYSPLEXPORTS, which is needed for sysplex-wide source VIPA to use one source VIPA for all outbound TCP connections within the sysplex.

- SWSA list structure (EZBDVIPA)

  This stores information about IPSec tunnels addressed to distributed DVIPAs within the sysplex or subplex. This information is used to renegotiate IPSec tunnels in case of distributed DVIPA takeover. SWSA is enabled through definitions in the IPSEC statement.

- SYSPLEXPORTS list structure (EZBEPORT)

  This contains all the ephemeral ports allocated in support of the sysplex-wide source VIPA function. Ephemeral ports that establish connections with external servers and use the sysplex-wide source VIPA function are allocated as participating clients from TCP/IP stacks within the sysplex or subplex.

  This function is defined using TCPSTACKSOURCEVIPA on the IPCONFIG statement and SYSPLEXPORTS on the VIPADISTRIBUTE statement.

If TCP and VTAM Coupling Facility structures are used, names must also be unique for each subplex in order to preserve separation between the subplexes. This means that the TCP structures EZBDVIPA and EZBEPORT must be appended with the VTAM and TCP XCF group ID suffixes to the end of the structure names (for example, EZBDVIPA*vvtt* and EZBEPORT*vvtt*, where *vv* is the 2-digit VTAM group ID suffix specified on the XCFGRPID start option and *tt* is the 2-digit TCP group ID specified in the TCP/IP profile).

The default suffixes are as follows:

- If no VTAM XCFGRPID is specified, then the structure names will be EZBDVIPA01tt and EZBEPORT01tt.
- If no TCP/IP XCFGRPID is specified, then a null value is used for *tt* when the structure names are built.
- If no VTAM XCFGRPID and no TCP/IP XCFGRPID are specified, then vv and tt are both null.

The TCP structure names, including the suffixes, must be defined in the sysplex CFRM policy (see Example 7-1).

Example 7-1 TCP/IP structure example for SYSPLEXPORTS - subplex 11

```
STRUCTURE NAME(EZBEPORT0111)
          INITSIZE(4096)
          SIZE(8192)
          PREFLIST(FACIL02,FACIL01)
```

Note: Example 7-1 is only a sample. The size depends on the number of source DVIPAs and concurrently established TCP outbound connections from all TCPSTACKSOURCEVIPA of the participating stacks within the sysplex. The ephemeral port number for each connection is stored to avoid duplicate source port numbers.

For more information regarding TCP and VTAM structures, refer to *z/OS MVS Setting Up a Sysplex,* SA22-7625.

The following sections describe the implementation for each subplex in detail.

7.3.1 Subplex 11 - internal subplex

Figure 7-5 on page 210 depicts the configuration for IP Subplex 11 (the internal subplex).

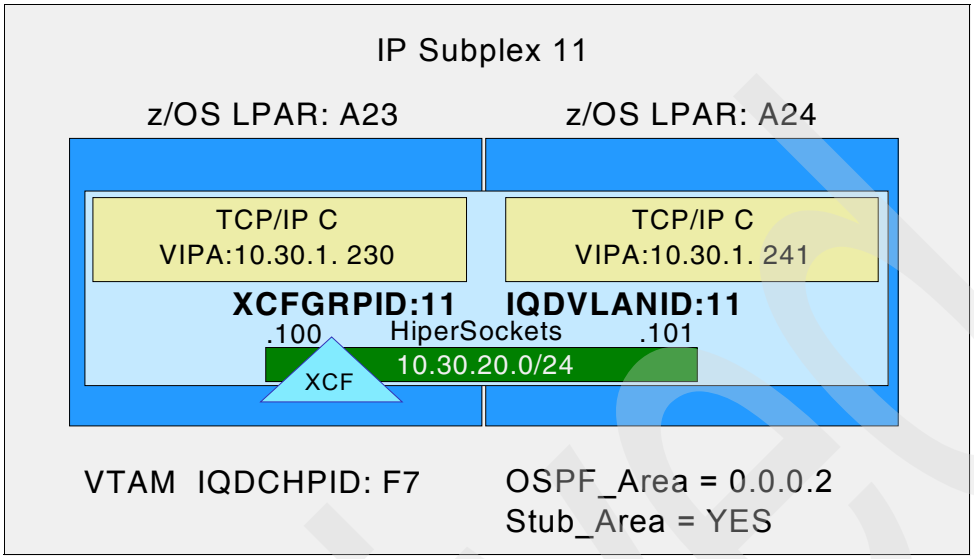


Figure 7-5 Subplex 11- internal subplex

TCP/IP profile definitions for subplex 11 in LPAR A24 stack C

Because we used automatically defined HiperSockets for the IP traffic within the subplex, we only had to add the VTAM start option IQDCHPID = F7 1. This CHPID is used when HiperSockets are implemented under z/OS.

The VTAM start option is needed by VTAM to automatically create the Transmission Resource List Element (TRLE) for the HiperSockets interface of the stack. The TRLE points to its IUTQDIO name, which is defined to the TCP/IP profile DEVICE name. The PORTNAME created by VTAM is IUTQD*xx*, where *xx* is the used Channel Path ID (CHPID).

The DYNAMICXCF function also requires the VTAM start option XCFINIT = YES 2, which dynamically creates the XCF major node.

Tip: You can check your VTAM start options by using the D NET,VTAMOPTS command.

Example 7-2 ATCSTRxx definitions needed for DYNAMICXCF and HiperSockets interface

```
SYS1.VTAMLST(ATCSTR31)
IQDCHPID=F7,                1
XCFINIT=YES                 2
```

Example 7-3 shows the TCP/IP profile definitions needed for assigning stack C in LPAR 24 to subplex 11. Based on the parameters XCFGRPID 3 and IQDVLANID 4, stack C belongs to subplex 11. The group interface is defined via the IPCONFIG parameter DYNAMICXCF with its IP address 10.30.20.101 5.

Example 7-3 TCP/IP profile - subplex definitions for stack C in LPAR 24

```
GLOBALCONFIG
  XCFGRPID 11  3
  IQDVLANID 11 4
;
```

```
IPCONFIG
DYNAMICXCF 10.30.20.101 255.255.255.0 8                    5
```

Note: We used the same value for XCFGRPID and IQDVILANID. These values do not have to match. XCFGRPID allows values from 2 to 31, while IQDVLANID allows values from 1 to 4094.

The definitions for LPAR A23 are not shown because the XCFGRPID is the same. Only the DNAMICXCF IP address 5 is different (10.30.20.100).

Verification of the subplex 11

The group name used is in the form EZBT*vvtt*, where *vv* is the 2-digit VTAM group ID suffix specified on the XCFGRPID start option or default (CP) and *tt* is the TCP group.

In our scenarios we did not define the VTAM start option XCFGRPID. A display from LPAR A24 TCP/IP stack C (see Example 7-4) shows that the stack is a member of the VTAM subplex group ID CP and TCP/IP subplex group 11, with the name EZBTCP11 6.

In the same LPAR there is another stack member of subplex group 22 with the name EZBTCP22 7 (see definitions in 7.3.2, “Subplex 22 - external subplex” on page 212).

Example 7-4 Displays of XCF groups

```
D XCF,GROUP
IXC331I  12.13.08  DISPLAY XCF 229
   GROUPS(SIZE):   ATRRRS(3)          COFVLFNO(3)       DBCDU(3)
                   EZBTCPCS(5)        EZBTCP11(2)   6
                   EZBTCP22(2)   7    IDAVQUIO(3)       IGWXSGIS(6)
                   IOEZFS(3)          IRRXCF00(3)       ISTCFS01(3)
                   ISTXCF(3)          IXCLO00A(3)       IXCLO00B(3)
                   IXCLO006(3)        SYSBPX(3)         SYSCNZMG(3)
```

The number between the parenthesis is related to the number of stacks active in the XCF group.

Example 7-5 displays that the stack in LPAR A24 is located in subplex 11 with its name EZBTCP11 9. The definitions for the subplex 22 (EZBTCP22) are described in 7.3.2, “Subplex 22 - external subplex” on page 212.

Example 7-5 Display of specific stacks that belong to an XCF group

```
D TCPIP,TCPIPC,SYSPLEX,GROUP
EZZ8270I SYSPLEX GROUP FOR TCPIPC   AT SC31     IS EZBTCP11   9

D TCPIP,TCPIPD,SYSPLEX,GROUP
EZZ8270I SYSPLEX GROUP FOR TCPIPD   AT SC31     IS EZBTCP22
```

The Netstat Config display shows the XCFGRPID 10 and the IQDVLANID 11.

Example 7-6 Netstat Config with XCFGRPID and IQDVLANID for stack C

```
D TCPIP,TCPIPC,NETSTAT,CONFIG
EZD0101I NETSTAT CS V1R8 TCPIPC 349
GLOBAL CONFIGURATION INFORMATION:
TCPIPSTATS: NO   ECSALIMIT: 0000000K  POOLLIMIT: 0000000K
MLSCHKTERM: NO   XCFGRPID:  11 10 IQDVLANID:  11 11
SYSPLEXWLMPOLL: 060
SYSPLEX MONITOR:
```

```
  TIMERSECS: 0060  RECOVERY: NO   DELAYJOIN: NO   AUTOREJOIN: NO
  MONINTF:   NO    DYNROUTE: NO
```

The command Netstat Device also shows the HiperSockets connection with VLANID **12**, which is the same value as IQDVLANID, as shown in Example 7-6 on page 211.

Example 7-7 Netstat Device showing the HiperSockets VLAN ID

```
D TCPIP,TCPIPC,NETSTAT,DEV
EZD0101I NETSTAT CS V1R8 TCPIPC 363
DEVNAME: IUTIQDIO          DEVTYPE: MPCIPA
  DEVSTATUS: READY
  LNKNAME: IQDIOLNK0A1E1465  LNKTYPE: IPAQIDIO   LNKSTATUS: READY
    NETNUM: N/A  QUESIZE: N/A
    IPBROADCASTCAPABILITY: NO
CFGROUTER: NON                   ACTROUTER: NON
    ARPOFFLOAD: YES                  ARPOFFLOADINFO: YES
    ACTMTU: 8192
    VLANID: 11  12                   VLANPRIORITY: DISABLED
    DYNVLANREGCFG: NO                DYNVLANREGCAP: NO
    READSTORAGE: GLOBAL (2048K)
    SECCLASS: 255
  BSD ROUTING PARAMETERS:
    MTU SIZE: 8192             METRIC: 15
    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                          = 57156
    INBOUND PACKETS                  = 548
    INBOUND PACKETS IN ERROR         = 0
    INBOUND PACKETS DISCARDED        = 0
    INBOUND PACKETS WITH NO PROTOCOL = 0
    BYTESOUT                         = 18296
    OUTBOUND PACKETS                 = 168
    OUTBOUND PACKETS IN ERROR        = 0
    OUTBOUND PACKETS DISCARDED       = 0
```

7.3.2 Subplex 22 - external subplex

Figure 7-6 on page 213 depicts the configuration for IP Subplex 22 (the external subplex). Note that both subplexes are using the same HiperSockets (CHPID F7).

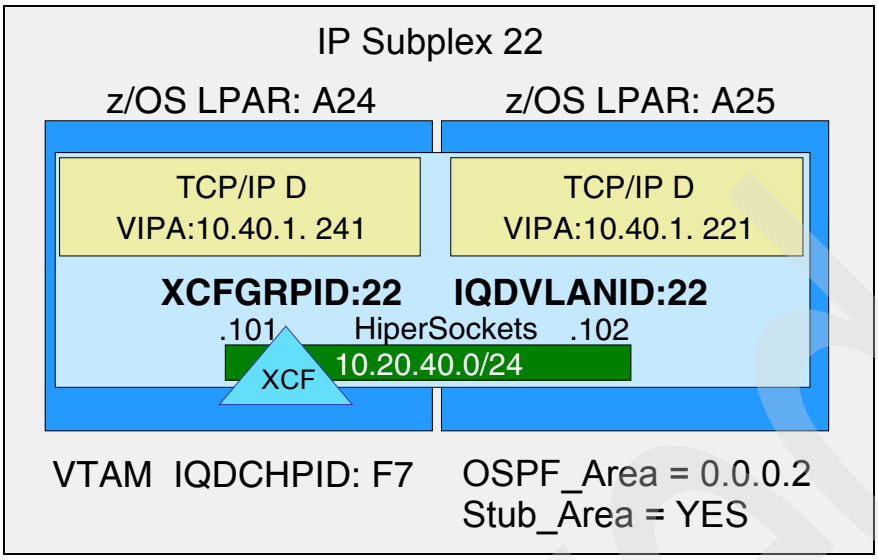


Figure 7-6 Subplex 22 - external subplex

TCP/IP profile definitions for subplex 22 in LPAR A24 stack D

If you compare the definitions for stack D with stack C in Example 7-3 on page 210, you will discover that only XCFGRPID 1, IQDVLANID 2, and DYNAMICXCF IP address 3 values are different.

Example 7-8 TCP/IP profile - subplex definitions for stack D in LPAR A24

```
GLOBALCONFIG
XCFGRPID 22 1
IQDVLANID 22 2
;
IPCONFIG
DYNAMICXCF 10.20.40.101 255.255.255.0 8                3
```

TCP/IP profile definitions for subplex 22 in LPAR A25 stack D

If you compare the definitions for stack D in LPAR 25 with stack D in LAPR 24 (see Example 7-8), you will discover that XCFGRPID 4 and IQDVLANID 5 have the same values. Only the DYNAMICXCF IP address value 6 is different.

Example 7-9 TCP/IP profile - subplex definitions for stack D in LPAR A25

```
GLOBALCONFIG
XCFGRPID 22 4
IQDVLANID 22 5
;
IPCONFIG
DYNAMICXCF 10.20.40.102 255.255.255.0 8                6
```

Verification of the subplex 22

The command **netstat config** shows the definitions used by the stack.

Example 7-10 Netstat config from LPAR 24 stack D

```
D TCPIP,TCPIPD,NETSTAT,CONFIG
EZD0101I NETSTAT CS V1R8 TCPIPD 286
GLOBAL CONFIGURATION INFORMATION:
```

```
TCPIPSTATS: NO   ECSALIMIT: 0000000K  POOLLIMIT: 0000000K
MLSCHKTERM: NO   XCFGRPID:  22  IQDVLANID:  22
SYSPLEXWLMPOLL: 060
SYSPLEX MONITOR:
  TIMERSECS: 0060  RECOVERY: NO   DELAYJOIN: NO   AUTOREJOIN: NO
  MONINTF:   NO    DYNROUTE: NO
```

7.3.3 Access verifications

We executed **ping** commands from all TCP/IP stacks in all LPARs. Example 7-11 shows a ping to IP address 10.30.20.101 (XCF and HiperSockets interface) from outside Subplex 11, which failed. All ping requests within each subplex were successful. Requests from other subplex groups or non-subplex groups were rejected.

Example 7-11 Ping test

```
===> ping 10.30.20.101
CS V1R8: Pinging host 10.30.20.101
 Ping #1 timed out
```

7.4 References

For more information on subplexing, refer to the following publications:

- *z/OS Communications Server: IP Configuration Reference*, SC31-8776
- *z/OS Communications Server: IP Configuration Guide*, SC31-8775
- *HiperSockets Implementation Guide,* SG24-6816

8

Diagnosis

A key topic in any TCP/IP network infrastructure is documenting and analyzing problems. In this chapter we describe both tools available in z/OS Communications Server, and techniques that can be used, to gather and diagnose problems related to the TCP/IP environment.

This chapter discusses the following topics.

Section	Topic
8.1, “Debugging a problem in a z/OS TCP/IP environment” on page 216	Problem determination techniques and the tools available to debug a problem in z/OS Communications Server - TCP/IP component
8.2, “Logs to diagnose CS for z/OS IP problems” on page 218	Why logs are important in problem analysis
8.3, “Useful commands to diagnose CS for z/OS IP problems” on page 219	Commands used to debug network problems
8.4, “Gathering traces in CS for z/OS IP” on page 225	Using z/OS Component Trace Service to capture trace data for the main z/OS Communications Server - TCP/IP component
8.5, “OSA-Express2 Network Traffic Analyzer” on page 239	Using OSAENTA to diagnose OSA problems
8.6, “Additional tools for diagnosing CS for z/OS IP problems” on page 257	Other tools that can be used to diagnose network problems

8.1 Debugging a problem in a z/OS TCP/IP environment

In a TCP/IP network, several different types of problems can arise. Therefore, the support staff needs to develop debugging techniques that can help them better understand, define, and debug such problems. In this section we discuss a problem determination approach that uses logs, standard commands, tools, and utilities.

When problems arise in a TCP/IP environment, they can sometimes be very challenging to isolate. Without the proper tools, techniques, and knowledge of the environment, it can be very difficult to debug any problem. The culprit could be any one of the many components between the affected endpoints.

Therefore, we suggest categorizing the problem. Problems in TCP/IP networks can usually be classified into three major categories:

- Network connectivity problems

  These occur when a z/OS server cannot establish a connection with another server or client because the node is unreachable (for example, it does not respond to the **ping** command).

- Application-related problems

  These occur when a host is reachable, but communication with the desired application fails.

- Stack-related problems

  These occur when the z/OS TCP/IP stack does not work as implemented, or ends with a dump.

Most problems can easily be associated to one of these categories and the information needed to debug them can be retrieved from logs, commands, or utilities.

Logs are the first and most important tool to help you understand the nature of the problem. In logs you will find messages that might explain what happened or even lead you to the actions needed to solve the problem.

Sometimes, however, problems like connectivity or routing do not provide messages that clearly show what went wrong. Therefore, you need further information, which can be obtained by using commands such as Netstat, Ping, or Traceroute. If the commands do not provide enough information to solve or isolate the problem, then you can invoke the z/OS Communications Server trace utilities that gather data as it passes through the devices and the stack.

Many problems related to the TCP/IP stack are due to configuration errors. Here you can use logs to find useful messages that indicate were the error is located.

If the TCP/IP stack happens to abend, a dump is generated. In such a situation, the dump and related information can be sent to IBM Support for further analysis.

8.1.1 An approach to problem analysis

When performing problem analysis, it is essential that you have readily available current, accurate documentation describing the physical and logical network environment. This documentation should include network diagrams, naming conventions, addressing schemes, and system configuration information.

When a problem occurs, the first step is to verify that the operating environment is behaving as expected. After this is confirmed, you can then focus on other areas. To help isolate the problem, a useful approach is to answer such basic questions including:

- Is the TCP/IP stack running correctly?

  This generic question can help determine whether the problem is stack-related. It can be answered by verifying the behavior of the entire Communications Server for z/OS IP environment.

  Usually the tools used to answer this question are the logs where messages related to the problem can be found (see 8.2, “Logs to diagnose CS for z/OS IP problems” on page 218) and tools that receive information via the Network Management Interface (see 8.6, “Additional tools for diagnosing CS for z/OS IP problems” on page 257).

  If the problem is an abend, save the generated dump for analysis. The configuration should also be checked for inconsistencies. If you conclude it is not a stack-related problem, then the next step will be based on your findings to determine whether it is a network- or application-related problem.

- Has this ever worked before? If so, what has changed?

  These two basic questions may seem obvious, but they are in fact the most common reasons for problems encountered in a Communications Server for z/OS IP environment.

  If the problem is with a production and stable environment, you must first check whether any changes have been made. In some cases, changes do not take affect until a system or stack recycle is done. The only useful approach in this case is to keep track of any changes and always use change management processes.

  If you are dealing with a new implementation, was a step-by-step approach being used? If so, you will probably know in which step the problem occurred and can adapt your problem determination procedure based on the step being implemented.

- Are the physical connections and interfaces active and working properly?

  This question is related to a connectivity problem, and it leads to checking interface definitions and status. You also need to look at the log files, and use commands to determine whether the interfaces are operational. The Netstat command can be used to verify this, as discussed in 8.3, “Useful commands to diagnose CS for z/OS IP problems” on page 219.

  If it is an intermittent problem, or if you cannot find the cause of the problem, Communications Server for z/OS IP provides a set of trace tools that you can use to gather more information; see 8.4, “Gathering traces in CS for z/OS IP” on page 225.

- Can the destination host be reached?

  In cases where the physical connections are up and running, but a specific host cannot be reached, the problem is probably related to routing. In this case, you need to look at the logs files for related error messages. You can also use commands such as **`ping`**, **`traceroute`**, and **`netstat`** to discover why you are not able to reach this host.

  If these steps do not provide you with enough information to isolate the problem, you will need to use the packet trace utility as described in 8.4, “Gathering traces in CS for z/OS IP” on page 225. A packet trace allows you to check if there is any data going to or coming from the host you are trying to reach.

- Is the problem affecting multiple connections?

  There might be a problem with the proper configuration of a firewall policy, an incorrect interface configuration, or an application problem.

  The approach in this case is to review the configuration files, looking for inconsistencies. Also examine the log files, which may contain error messages about this problem. If

necessary, you can also debug this problem using the component trace for event and packet tracing; see 8.4, “Gathering traces in CS for z/OS IP” on page 225.

- Is this problem related to a single application?

  To analyze application problems, you need general knowledge of the application protocol. You should know what transport protocol is used, which port numbers are used, how a connection is established, and the application protocol semantics.

  Mainly, the following tools are used to diagnose application problems:

  – Debugging commands
  – Specific application traces
  – Packet trace
  – Component trace

  You can check whether an application is running by using display commands. With TELNET, for example, the D TCPIP,,TELNET, and V TCPIP,,TELNET, commands can be used to verify and control Telnet connections.

  Specific application traces are useful for following the execution of an application (either client or server), and checking whether there are error messages. The application trace may not be sufficient to diagnose some problems because it shows the commands (rather than the data) exchanged during a connection.

  For an in-depth investigation you need to use a packet trace, which can be interpreted relatively easily for standard applications (see 8.4, “Gathering traces in CS for z/OS IP” on page 225), or a CTRACE (when requested by IBM).

8.2 Logs to diagnose CS for z/OS IP problems

To start the problem determination process, the most important step is to pull together reliable information to verify, classify, and define possible lines of action to resolve a problem. Examining logs is an excellent starting point in the problem determination process. Logs contain different types of messages (informational, error-based, and warning-based) that provide very useful information. Logs are very important in the problem determination process because they may be the only source of information about a problem.

For example, in a production environment, problems are often business- or service-related, and end users are the first to notice there is a problem (usually, because they are unable to access applications or execute services). The operational response taken when a problem is discovered is often based on business or service recovery; it is usually only after these actions are taken that support personal are called upon to evaluate and determine why the outage occurred in the first place. In some cases, there is no information given other than a problem description based on the business or service point of view, with no technical perspective.

In many situations, the information obtained during the problem determination process comes from separate logs (system, application, and stack logs). To be able to build a clear picture of the problem or outage, all significant information must be correlated.

Because of this, we recommend that you implement syslogd to control where all messages are sent. This way, you will have a single place to refer to when debugging a problem. The syslogd process is a UNIX process that logs UNIX application messages to one or more files.

TCP/IP services that run as UNIX processes log application messages using syslogd can consolidate logging information from several systems to one system via UDP communications.

For further information about setting up syslogd, refer to *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 2: Standard Applications*, SG24-7533.

8.3 Useful commands to diagnose CS for z/OS IP problems

To solve problems it is important to know what tools are available and how to make best use of them. Some commands or utilities can be used to review configuration options or settings; others can be used to test connectivity.

In this section we describe briefly the main commands that you can use to diagnose problems in a Communications Server for z/OS IP environment. For additional help and detailed information about the commands described and other commands that can be used for problem determination, refer to *z/OS Communications Server: IP System Administrator's Commands*, SC31-8781, and *z/OS Communications Server: IP Diagnosis Guide*, GC31-8782.

We describe these commands:

- PING command (TSO or z/OS UNIX)
- TRACEROUTE command
- Netstat command (console, TSO, or z/OS UNIX)

8.3.1 PING command (TSO or z/OS UNIX)

The **`ping`** command is relatively simple, but it is one of the best tools you can use to check basic connectivity. It sends an ICMP echo request message to the target system and waits for an ICMP echo reply message. Because this command uses only two ICMP messages (echo request and echo response), it cannot be used to test transport or application protocols. In order for ping to work, the sending system and all intermediate systems must be correctly set up for both the outbound and inbound journeys.

Typically, ping is used to verify:

- The route to a network is defined correctly.
- The router is able to forward packets to the network.
- The remote host is able to send and receive packets on the network
- The remote host has a route back to the local host.

Tip: Using names instead of IP address needs the Resolver or DNS to do the translation, thus adding more variables to the problem determination task. This should be avoided when diagnosing network problems. Use the host IP address instead.

In most cases, the default options of ping are used. However, in a z/OS Communications Server environment, using the default options might lead to a false conclusion, given the number of interfaces that can be used to transport the ICMP request.

This command provides several options that can be used to analyze a problem in more detail. For example, the **`intf`** option of the TSO **`ping`** command (or the **`-i`** option, if you use the z/OS UNIX **`ping`** command**)** specifies the local interface over which the packets are sent. This can be useful to determine if a remote host is reachable through the desired path.

Table 8-1 on page 220 shows the available options that can be used with the **`ping`** command in TSO and z/OS UNIX environments.

Table 8-1 Options available with ping

Options	TSO	z/OS UNIX
Address type (ipv4 /ipv6)	Addrtype	-A
Number of ping iteractions	Count	-c
Interface to be used as path	Intf	-i
Amount of data being sent	Length	-l
Do not resolve IP addresses to host names (used with Pmtu)	NOName	-n
Determine the path MTU size of a host in the network.	PMTU	-P
Source IP address	Srcip	-s
TCP/IP stack to be used.	TCP	-p
How long it waits for a response	Timeout	t
Help information	HELP or ?	-h or -?

Figure 8-1 illustrates the use of the **ping** command for problem determination.

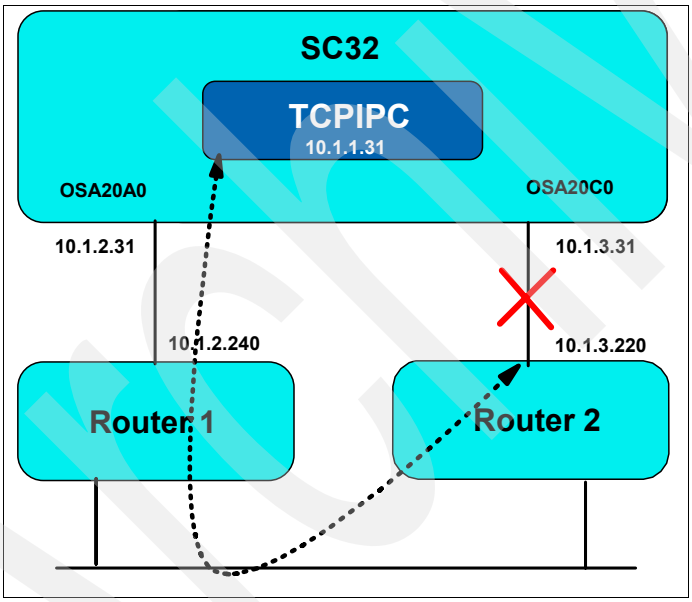


Figure 8-1 Using the ping command with the interface option

This is a situation where a ping might work even when the expected route to the end point is down. In this case the end point was accessed via an alternate route.

However, using the **ping** command *without* the correct option would hide the problem, as shown in Example 8-1.

Example 8-1 ping command without the intf option

```
ping 10.1.3.220 (tcp tcpipc
CS V1R9: Pinging host 10.1.3.220
Ping #1 response took 0.001 seconds.
 ***
```

To avoid such confusion, indicate which path to verify by using the interface (**intf**) option, as shown in Example 8-2.

Example 8-2 ping command with the intf option

```
   ping 10.1.3.220 (tcp tcpipc intf osa20c0l
   CS V1R9: Pinging host 10.1.3.220
   sendMessage(): EDC8130I Host cannot be reached. (errno2=0x74420291)
***
```

After using the correct command, you can see there is a problem using interface OSA20C0L, which is the direct connection to the 10.1.3.0 subnetwork.

Using the PMTU option

Use the PMTU option on the **ping** command to determine where fragmentation is necessary in the network. The PMTU YES option differs from the PMTU IGNORE option in the way the PING completes its echo function. Refer to *z/OS Communications Server: IP System Administrator's Commands*, SC31-8781, for a detailed discussion about the PMTU option.

Example 8-3 shows a ping with a very large packet size, with no PMTU option specified. We used the noname option to avoid a reverse DNS lookup on the IP address.

Example 8-3 Using ping without the PMTU option

```
ping 10.1.1.10 (noname tcp tcpipb l 25000 c 1 t 1
CS V1R9: Pinging host 10.1.1.10
Ping #1 response took 0.000 seconds.
***
```

Example 8-4 shows that by adding the **PMTU YES** option **1** to the **ping** command, you can determine at which hop **2** the fragmentation is necessary, and the MTU size **3**.

Example 8-4 Using ping with the PMTU option

```
ping 10.1.1.10 (noname tcp tcpipb l 25000 c 1 t 1 pmtu yes  1
CS V1R9: Pinging host 10.1.1.10
Ping #1 needs fragmentation at: 10.1.7.21                   2
  Next-hop MTU size is 8192                                 3
***
```

By varying the size of the ping packet, you can work your way through the path to the hop requiring fragmentation, as shown in Example 8-5.

Example 8-5 Varying the ping packet size

```
ping 10.1.1.10 (noname tcp tcpipb l 6000  c 1 t 1 pmtu yes
CS V1R9: Pinging host 10.1.1.10
Ping #1 response took 0.000 seconds.
***
ping 10.1.1.10 (noname tcp tcpipb l 8164  c 1 t 1 pmtu yes
CS V1R9: Pinging host 10.1.1.10
Ping #1 response took 0.000 seconds.
***

ping 10.1.1.10 (noname tcp tcpipb l 8165  c 1 t 1 pmtu yes
CS V1R9: Pinging host 10.1.1.10
Ping #1 needs fragmentation at: 10.1.5.21 (10.1.5.21)
  Next-hop MTU size is 8192
```

```
***
```

Example 8-6 illustrates the use of the **PMTU IGNORE** option.

Example 8-6 Ping with the PMTU IGNORE option

```
ping 10.1.1.10 (noname tcp tcpipb l 25000 c 1 t 1 pmtu ignore
CS V1R9: Pinging host 10.1.1.10
Ping #1 needs fragmentation at: 10.1.7.21
  Next-hop MTU size is 8192
***
```

8.3.2 TRACEROUTE command

The **traceroute** (z/OS UNIX) or **tracerte** (TSO) command is used to determine the route that IP datagrams follow through the network. Traceroute is based on ICMP and UDP. It sends an IP datagram with a time-to-live (TTL) of 1 to the destination host. The first router decrements the TTL to 0, discards the datagram, and returns an ICMP Time Exceeded message to the source. In this way, the first router in the path is identified. This process is repeated with successively larger TTL values to identify the exact series of routers in the path to the destination host.

On most platforms, the **traceroute** command sends UDP datagrams to the destination host. These datagrams reference a port number outside the standard range. The source knows when it has reached the destination host when it receives an ICMP 'Port Unreachable' message.

Traceroute displays the route that a packet takes to reach the requested target. The output generated by this command can be seen in Example 8-7.

Example 8-7 Tracerte command results

```
CS V1R9: Traceroute to 10.1.100.222 (10.1.100.222)
 1 10.1.2.240 (10.1.2.240)  0 ms  0 ms  0 ms
 2 10.1.100.222 (10.1.100.222)  0 ms  0 ms  0 ms
 ***
```

8.3.3 The netstat command (console, TSO, or z/OS UNIX)

The **netstat** command provides information about the status of the local host, including information about TCP/IP configuration, connections, network clients, gateways, and devices. It also has options to drop connections for users who have the MVS.VARY.TCPIP.DROP statement defined in their RACF profile.

As shown in Figure 8-2 on page 223, there are a variety of Netstat options and these may be further qualified by filter criteria, depending on the option you choose. The output may be displayed to the terminal (default), to a dataset (report), or to the REXX data stack. The Output Format (short or long) supports IPv6 addresses.

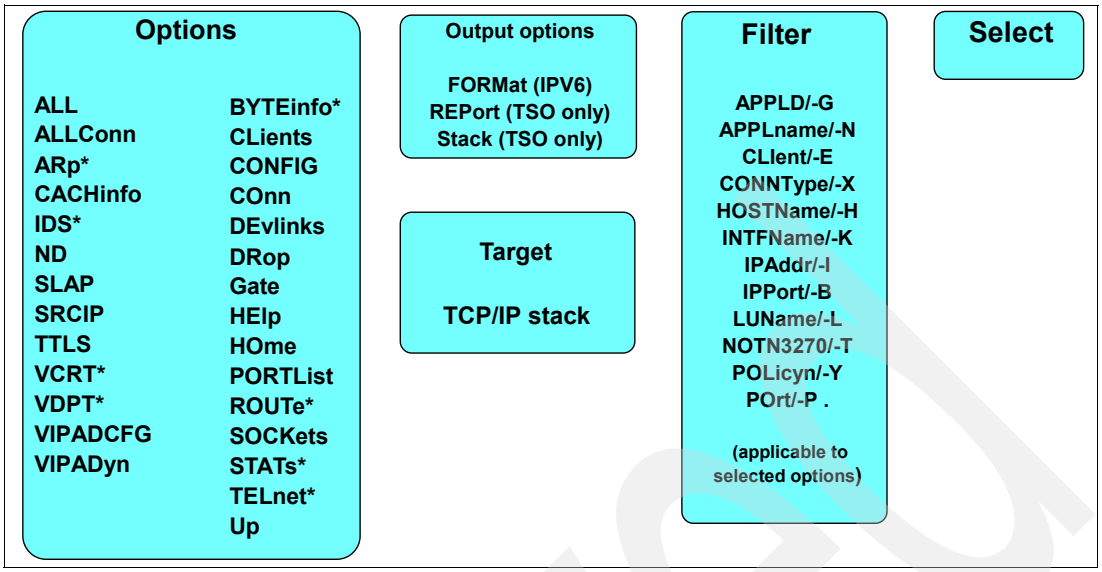


Figure 8-2 Netstat command options - target output (filter select)

The remainder of this section shows examples of **netstat** commands used for diagnostic purposes, and their outputs.

Example 8-8 displays the results of the **d tcpip,tcpipc,nestat,conn** command.

Example 8-8 d tcpip,tcpipc,nestat,conn command

```
D TCPIP,TCPIPC,NETSTAT,CONN
EZD0101I NETSTAT CS V1R9 TCPIPC 117
USER ID  CONN     STATE
FTPDC1   0000001A LISTEN
  LOCAL SOCKET:   ::..21
  FOREIGN SOCKET: ::..0
JES2S001 00000013 LISTEN
  LOCAL SOCKET:   ::..175
  FOREIGN SOCKET: ::..0
TN3270C  00000018 LISTEN
  LOCAL SOCKET:   ::..23
  FOREIGN SOCKET: ::..0
3 OF 3 RECORDS DISPLAYED
END OF THE REPORT
```

Use the **D TCPIP,*tcpipproc*,NETSTAT, DEVLINK** command to display the status and associated configuration values for a device and its defined interfaces. This command can be filtered to display only the interface you want, as shown in Example 8-9.

Example 8-9 D TCPIP,TCPIPC,N,DEV,INTFN=OSA2080L

```
D TCPIP,TCPIPC,N,DEV,INTFN=OSA2080L
EZD0101I NETSTAT CS V1R9 TCPIPC 140
DEVNAME: OSA2080           DEVTYPE: MPCIPA
  DEVSTATUS: READY
  LNKNAME: OSA2080L          LNKTYPE: IPAQENET   LNKSTATUS: READY
    NETNUM: N/A  QUESIZE: N/A  SPEED: 0000001000
    IPBROADCASTCAPABILITY: NO
    CFGROUTER: NON                  ACTROUTER: NON
    ARPOFFLOAD: YES                 ARPOFFLOADINFO: YES
    ACTMTU: 8992
```

```
    VLANID: 10                      VLANPRIORITY: DISABLED
    DYNVLANREGCFG: NO               DYNVLANREGCAP: YES
    READSTORAGE: GLOBAL (4096K)     INBPERF: BALANCED
    CHECKSUMOFFLOAD: YES
    SECCLASS: 255                   MONSYSPLEX: NO
  BSD ROUTING PARAMETERS:
    MTU SIZE: N/A              METRIC: 00
    DESTADDR: 0.0.0.0          SUBNETMASK: 255.0.0.0
  MULTICAST SPECIFIC:
    MULTICAST CAPABILITY: YES
    GROUP             REFCNT        SRCFLTMD
    GROUP             REFCNT        SRCFLTMD
    -----             ------        --------
    224.0.0.1         0000000001    EXCLUDE
      SRCADDR: NONE
  LINK STATISTICS:
    BYTESIN                          = 240
    INBOUND PACKETS                  = 3
    INBOUND PACKETS IN ERROR         = 0
    INBOUND PACKETS DISCARDED        = 0
    INBOUND PACKETS WITH NO PROTOCOL = 0
    BYTESOUT                         = 84
    OUTBOUND PACKETS                 = 1
    OUTBOUND PACKETS IN ERROR        = 0
    OUTBOUND PACKETS DISCARDED       = 0
IPV4 LAN GROUP SUMMARY
LANGROUP: 00001
  LNKNAME          LNKSTATUS   ARPOWNER          VIPAOWNER
  -------          ---------   --------          ---------
  OSA20A0L         ACTIVE      OSA20A0L          YES
  OSA2080L         ACTIVE      OSA2080L          NO
1 OF 1 RECORDS DISPLAYED Outbound Packets Discarded       = 0
```

The **D TCPIP,*tcpipproc*,NETSTAT,ROUTE** command displays the current routing tables for TCP/IP, and it can be filtered to show specific routes as shown in Example 8-10.

Example 8-10 D TCPIP,TCPIPC,N,ROUTE,IPADDR=10.1.100.0/24

```
D TCPIP,TCPIPC,N,ROUTE,IPADDR=10.1.100.0/24
EZD0101I NETSTAT CS V1R9 TCPIPC 196
IPV4 DESTINATIONS
DESTINATION         GATEWAY          FLAGS     REFCNT  INTERFACE
10.1.100.0/24       10.1.2.240       UGO       000000  OSA2080L
10.1.100.0/24       10.1.2.240       UGO       000000  OSA20A0L
10.1.100.0/24       10.1.3.240       UGO       000000  OSA20C0L
10.1.100.0/24       10.1.3.240       UGO       000000  OSA20E0L
4 OF 4 RECORDS DISPLAYED
END OF THE REPORT
```

You can optionally display additional application connection data by using the APPLDATA parameter on the NETSTAT CONN and NETSTAT ALLCONN commands. Example 8-11 on page 225 contrasts the output of two NETSTAT CONN commands: one without the APPLDATA parameter **1**, the other with the APPLDATA parameter **2**. The TN3270 server populates the APPLDATA field with connection data, as documented in *z/OS Communications Server: IP Configuration Reference*, SC31-8776. The TN3270 appldata fields shown for the connection are the component ID, LU name, the SNA application name, connection mode, client type, security method, security level, and security cipher **3**.

Example 8-11 NETSTAT CONN without and with the APPLDATA option

```
-D TCPIP,TCPIPb,N,conn                                    1
 EZD0101I NETSTAT CS V1R9 TCPIPB 385
 USER ID  CONN     STATE
TN3270B  00000111 ESTBLSH
  LOCAL SOCKET:   ::FFFF:10.1.1.20..23
  FOREIGN SOCKET: ::FFFF:10.1.1.20..1028

-D TCPIP,TCPIPb,N,conn,appldata                           2
EZD0101I NETSTAT CS V1R9 TCPIPB 385
 USER ID  CONN     STATE
TN3270B  00000111 ESTBLSH
  LOCAL SOCKET:   ::FFFF:10.1.1.20..23
  FOREIGN SOCKET: ::FFFF:10.1.1.20..1028
  APPLICATION DATA:  EZBTNSRV SC31BB05 SC31TS03 3T B    3
```

You can filter the output of the NETSTAT CONN,APPLDATA command by adding the APPLD filter option and specifying the filter criteria. The APPLDATA field is a total of 40 bytes. By using an asterisk (*) in the filter criteria, you can filter on any part of the 40 bytes. Example 8-12 shows several filter criteria strings being used.

Example 8-12 NETSTAT CONN APPLDATA with APPLD filter

```
D TCPIP,TCPIPb,N,conn,appldata,appld=*tnsrv*
EZD0101I NETSTAT CS V1R9 TCPIPB 459
USER ID  CONN     STATE
TN3270B  00000111 ESTBLSH
  LOCAL SOCKET:   ::FFFF:10.1.1.20..23
  FOREIGN SOCKET: ::FFFF:10.1.1.20..1028
  APPLICATION DATA:  EZBTNSRV SC31BB05 SC31TS03 3T B
1 OF 1 RECORDS DISPLAYED
END OF THE REPORT

D TCPIP,TCPIPb,N,conn,appldata,appld=*sc31*
EZD0101I NETSTAT CS V1R9 TCPIPB 462
USER ID  CONN     STATE
TN3270B  00000111 ESTBLSH
  LOCAL SOCKET:   ::FFFF:10.1.1.20..23
  FOREIGN SOCKET: ::FFFF:10.1.1.20..1028
  APPLICATION DATA:  EZBTNSRV SC31BB05 SC31TS03 3T B
1 OF 1 RECORDS DISPLAYED
END OF THE REPORT
```

8.4 Gathering traces in CS for z/OS IP

Using trace tools is helpful when you have a concern about what is happening in the *flow* of data. Communications Server for z/OS IP provides general trace and application-specific trace facilities. Included in the general traces are packet trace and event trace. Both utilize the Component Trace (CTRACE) facilities of z/OS.

- A *packet trace* captures data packets that flow in or out of the IP stack.
- An *event trace* can capture data flows within the stack, through the application socket interfaces as well as other network flows, such as the ARP process.

This section deals with the trace facilities that are available to analyze TCP/IP problems on z/OS servers and clients. It also discusses how to process those traces.

The MVS component trace can be used to diagnose most TCP/IP problems. Some components of TCP/IP continue to maintain their own tracing mechanisms, for example, the FTP server. Consult *z/OS Communications Server: IP Diagnosis Guide*, GC31-8782, and *z/OS 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 (SYSTCPIP)
- Packet trace (SYSTCPDA)
- Socket data trace
- OMPROUTE trace (SYSTCPRT)
- Resolver trace (SYSTCPRE)
- Intrusion detection services trace (SYSTCPIS)
- IKE daemon trace (SYSTCPIK)
- OSAENTA trace (SYSTCPOT)
- Network security services (NSS) server trace (SYSTCPNS)
- Configuration profile trace

Figure 8-3 shows the traces that can be used for debugging. Some applications have their own internal trace functions. The output from those traces can be to the screen, a file, or to the syslogd logging function. The data from the z/OS Component Trace is written to either an external writer or the TCP/IP data space TCPIPDS1.

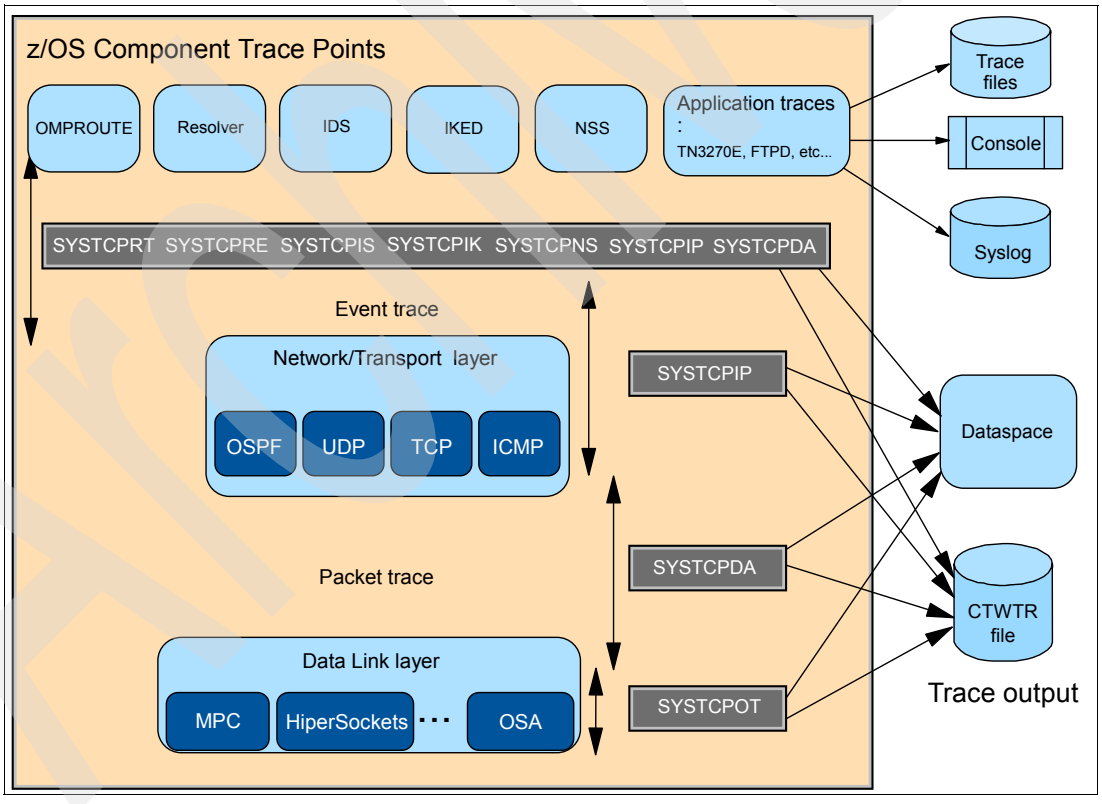


Figure 8-3 Trace points

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 MVS Diagnosis: Tools and Service Aids*, GA22-7589.

8.4.1 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). In this section we show the necessary steps to start 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.

Before starting the traces, create the external write procedure in the SYS1.PROCLIB library, which allocates the trace data set. This procedure will be activated using the trace command. A sample procedure named CTWTR is shown in Figure 8-4.

```
//CTWTR    PROC
//IEFPROC  EXEC PGM=ITTTRCWR
//TRCOUT01 DD  DSNAME=SYS1.&SYSNAME..CTRACE,
//         VOL=SER=COMST2,UNIT=3390,
//         SPACE=(CYL,10),DISP=(NEW,CATLG),DSORG=PS
//*
```

Figure 8-4 sample External Write procedure

Next, follow these steps using the **trace** command to activate, capture data, and stop the trace process:

1. Start the external writer (CTRACE writer).

   ```
   TRACE CT,WTRSTART=ctwrt
   ```

   Where *ctwrt* is the name of the procedure created to allocate the trace data set.

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
   ```

   Where:

   component is the component name of the trace being started, and can be any of these:

   ```
   SYSTCPIP (Event trace)
   SYSTCPDA (Packet trace)
   SYSTCPDA (Data trace)
   SYSTCPIS (Intrusion Detection Services trace)
   SYSTCPIK (IKE daemon trace)
   SYSTCPOT (OSAENTA trace)
   SYSTCPNS (Network security services (NSS) server trace)
   SYSTCPRT (OMPROUTE trace)
   SYSTCPRE (RESOLVER trace)
   Configuration profile trace
   ```

   proc_name is the procedure related to the component trace being started, and can be any of these:

   ```
   tcpip_proc
   iked_proc
   nss_proc
   omp_proc
   ```

   ctwtr is the started procedure name of the external writer.

3. To verify the trace is started, use the **display trace** command.

   ```
   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
```

The next sections describe each component trace used by z/OS Communications Server - TCP/IP component for documenting problems. For a detailed explanation for each component trace, refer to *z/OS Communications Server: IP Diagnosis Guide*, GC31-8782.

8.4.2 Event trace for TCP/IP stacks (SYSTCPIP)

The TCP/IP event trace, SYSTCPIP, traces TCP/IP stack components such as IP, ARP, TCP, UDP, TELNET, VTAM, and Socket API (SOCKAPI). It is automatically started at TCP/IP initialization using the CTRACE parm option in the parms statement of the TCP/IP sack startup procedure.

z/OS Communications Server provides a default trace options set in the SYS1.PARMLIB member (CTIEZB00 for SYSTCPIP, and CTIEZBTN for the TN3270 Telnet server). The options provided can be changed using an alternate member with the desired options (for example, CTIEZBXX), and then changing the value in the parm CTRACE keyword in your TCP/IP procedure; see Figure 8-5.

Note: The buffer size option is defined during TCP/IP startup only, so any change needs to be done using the CTIEZBxx parmlib member and cannot be reset without restarting the TCP/IP address space. The default is 8 MB.

```
//TCPIPC   PROC PARMS='CTRACE(CTIEZBXX)'
//*
//TCPIP    EXEC PGM=EZBTCPIP,REGION=0M,TIME=1440,
//              PARM='&PARMS'
```

Figure 8-5 Overriding CTIEZB00 with CTIEZBXX

If you want to specify different trace options after TCP/IP initialization, you can execute the **TRACE CT** command and either specify the new component trace options file or respond to prompts from the command.

Figure 8-6 on page 229 shows the status of the component trace for TCP/IP procedure TCPIPA as it has been initialized using SYS1.PARMLIB member CTIEZB01. Note that we have changed the default value for BUFSIZE to 4 M.

```
RESPONSE=SC30
 IEE843I 17.09.02  TRACE DISPLAY 466
         SYSTEM STATUS INFORMATION
  ST=(ON,0256K,00512K) AS=ON  BR=OFF EX=ON  MO=OFF MT=(ON,024K)
   TRACENAME
   =========
   SYSTCPIP
                        MODE BUFFER HEAD SUBS
                        ====================
                        OFF          HEAD    4
      NO HEAD OPTIONS
   SUBTRACE             MODE BUFFER HEAD SUBS
  ------------------------------------------------------------
   TCPIPA               ON   0004M
      ASIDS      *NONE*
      JOBNAMES   *NONE*
      OPTIONS    MINIMUM
      WRITER     *NONE*
```

Figure 8-6 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 applications. The SOCKAPI option captures trace information related to the socket API calls that an application may issue.

When you need to trace application-related problems using the SOCKAPI option, it is recommended that you follow these guidelines:

- Trace only one application. Use the job name or ASID option when capturing the trace to limit the trace data to one application.
- Trace only the SOCKAPI option. To get the maximum number of SOCKAPI trace records, specify only the SOCKAPI option.
- Use an external writer. The external writer is recommended to save more trace data.
- Trace only one TCP/IP stack.
- Activate the data trace only if more data is required. The SOCKAPI trace contains the first 96 bytes of data sent or received, which is usually sufficient.

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. Refer to *z/OS CS: IP Diagnosis,* GC31-8782, for further details on the SOCKAPI option.

Sample SYSTCPIP trace

In this section we followed the steps described in 8.4.1, “Taking a component trace” on page 227 to start, get data, and stop a CTRACE for component SYSTCPIP. The TN3270 server 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; it uses its own private storage.
- A subset of the trace commands are used 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.

The resulting messages are shown after each command, as follows:

1. Start the external writer (CTRACE writer).

```
TRACE CT,WTRSTART=CTWTR
ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
WERE SUCCESSFULLY EXECUTED.
IEF196I IGD100I 8623 ALLOCATED TO DDNAME TRCOUT01 DATACLAS (        )
ITT110I INITIALIZATION OF CTRACE WRITER CTWTR COMPLETE.
```

2. Connect to the CTRACE external writer and specify trace options.

```
TRACE CT,ON,COMP=SYSTCPIP,SUB=(TCPIPC)
*060 ITT006A SPECIFY OPERAND(S) FOR TRACE CT COMMAND.
 R 60,JOBNAME=(FTPDC),OPTIONS=(SOCKAPI),WTR=CTWTR,END
 IEE600I REPLY TO 060 IS;JOBNAME=(FTPDC),OPTIONS=(SOCKAPI),WTR=CTWTR
 ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
 WERE SUCCESSFULLY EXECUTED.
```

Note: We could use the parmlib member CTIEZBxx to provide the same options: TRACE CT,ON,COMP=SYSTCPIP,SUB=(TCPIPC),PARM=(CTIEZBXX)

3. Display the active component trace options to verify they are correct.

```
DISPLAY TRACE,COMP=SYSTCPIP,SUB=(TCPIPC)
IEE843I 12.12.22  TRACE DISPLAY 206
        SYSTEM STATUS INFORMATION
 ST=(ON,0256K,00512K) AS=ON  BR=OFF EX=ON  MO=OFF MT=(ON,024K)
  TRACENAME
  =========
  SYSTCPIP
                      MODE BUFFER HEAD SUBS
                      ====================
                      OFF         HEAD    3
     NO HEAD OPTIONS
  SUBTRACE            MODE BUFFER HEAD SUBS
 ------------------------------------------------------------
  TCPIPC              ON   0008M
     ASIDS      *NONE*
     JOBNAMES   FTPDC
     OPTIONS    SOCKAPI
```

```
     WRITER     CTWTR
```

4. Reproduce the failure you want to trace.
5. Disconnect the external writer.

```
TRACE CT,ON,COMP=SYSTCPIP,SUB=(TCPIPC)
*061 ITT006A SPECIFY OPERAND(S) FOR TRACE CT COMMAND.
 R 61,WTR=DISCONNECT,END
 IEE600I REPLY TO 061 IS;WTR=DISCONNECT,END
 ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
 WERE SUCCESSFULLY EXECUTED.
```

6. Stop the component trace.

```
TRACE CT,OFF,COMP=SYSTCPIP,SUB=(TCPIPC)
ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
WERE SUCCESSFULLY EXECUTED.
```

7. Stop the external writer.

```
TRACE CT,WTRSTOP=CTWTR
ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
WERE SUCCESSFULLY EXECUTED.
ITT111I CTRACE WRITER CTWTR TERMINATED BECAUSE OF A WTRSTOP REQUEST.
IEF196I IEF142I CTWTR CTWTR - STEP WAS EXECUTED - COND CODE 0000
IEF196I IEF285I   SYS1.SC32.CTRACE CATALOGED
```

After your events trace data is captured, the trace data set created by the external writer procedure is saved and IPCS is used to format and analyze its contents. Refer to *z/OS CS: IP Diagnosis,* GC31-8782, for further details about SYSTCPIP events trace.

8.4.3 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.

You can also use the packet trace to capture data traffic going through fast local socket (local traffic).

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.

To run a packet trace we follow the steps described in 8.4.1, “Taking a component trace” on page 227 to activate the component trace for component SYSTCPDA, a.d then activate the packet trace with the desired options and filters. All data in this sample is written to an external writer:

1. Start the CTRACE external writer.

```
TRACE CT,WTRSTART=CTWTR
ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
WERE SUCCESSFULLY EXECUTED.
IEF196I IGD100I 8623 ALLOCATED TO DDNAME TRCOUT01 DATACLAS (        )
ITT110I INITIALIZATION OF CTRACE WRITER CTWTR COMPLETE.
```

2. Start the CTRACE and connect the external writer to the TCP/IP address space.

```
TRACE CT,ON,COMP=SYSTCPDA,SUB=(TCPIPC)
```

```
063 ITT006A SPECIFY OPERAND(S) FOR TRACE CT COMMAND.
R 63,WTR=CTWTR,END
IEE600I REPLY TO 063 IS;WTR=CTWTR,END
ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
WERE SUCCESSFULLY EXECUTED.
```

3. Check that the trace started successfully.

```
D TRACE,COMP=SYSTCPDA,SUB=(TCPIPC)
IEE843I 14.00.29  TRACE DISPLAY 388
        SYSTEM STATUS INFORMATION
TRACENAME
  =========
  SYSTCPDA
                      MODE BUFFER HEAD SUBS
                      ====================
                      OFF         HEAD    2
     NO HEAD OPTIONS
  SUBTRACE            MODE BUFFER HEAD SUBS
 ------------------------------------------------------------
  TCPIPC              ON   0016M
     ASIDS      *NONE*
     JOBNAMES   *NONE*
     OPTIONS    MINIMUM
     WRITER     CTWTR
```

4. Start the trace through the PROFILE.TCPIP statement and the VARY OBEYFILE command, or through the V TCPIP,,PKT command.

```
VARY TCPIP,TCPIPC,PKT,ON
EZZ0060I PROCESSING COMMAND: VARY TCPIP,TCPIPC,PKT,ON
EZZ0053I COMMAND VARY PKTTRACE COMPLETED SUCCESSFULLY
```

   The trace options may modify the data being captured using the VARY command.

   If both DEST and SRCP are specified in the same command, an AND condition is created so data is only captured if *both* conditions are met. For example, issuing the following VARY command records only the packets with both a destination port of xxxx AND a source port of yyyy.

```
V TCPIP,TCPIPC,PKTTRACE,DEST=xxxx,SRCP=yyyy
```

   It can also create an OR condition issuing multiple VARY commands to apply filters together. 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
```

   A new filter option has been introduced in z/OS V1R9 Communications Server: the PORTNUM option allows you to filter the packet trace by port number as shown.

```
VARY TCPIP,TCPIPC,PKT,ON
EZZ0060I PROCESSING COMMAND: VARY TCPIP,TCPIPC,PKT,ON
EZZ0053I COMMAND VARY PKTTRACE COMPLETED SUCCESSFULLY
VARY TCPIP,TCPIPC,PKT,PORTNUM=23
EZZ0060I PROCESSING COMMAND: VARY TCPIP,TCPIPC,PKT,PORTNUM=23
EZZ0053I COMMAND VARY PKTTRACE COMPLETED SUCCESSFULLY
```

5. Perform the operation that you want to trace.

6. Stop the trace.

```
VARY TCPIP,TCPIPC,PKT,OFF
EZZ0060I PROCESSING COMMAND: VARY TCPIP,TCPIPC,PKT,OFF
EZZ0053I COMMAND VARY PKTTRACE COMPLETED SUCCESSFULLY
```

7. Disconnect the external writer from TCP/IP.

```
TRACE CT,ON,COMP=SYSTCPDA,SUB=(TCPIPC)
064 ITT006A SPECIFY OPERAND(S) FOR TRACE CT COMMAND.
R 64,WTR=DISCONNECT,END
IEE600I REPLY TO 064 IS;WTR=DISCONNECT,END
ITT120I SOME CTRACE DATA LOST, LAST 9 BUFFER(S) NOT WRITTEN
ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
WERE SUCCESSFULLY EXECUTED.
```

8. Stop the CTRACE.

```
TRACE CT,OFF,COMP=SYSTCPDA,SUB=(TCPIPC)
ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
WERE SUCCESSFULLY EXECUTED
```

9. Stop the external writer.

```
TRACE CT,WTRSTOP=CTWTR
ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
WERE SUCCESSFULLY EXECUTED.
ITT111I CTRACE WRITER CTWTR TERMINATED BECAUSE OF A WTRSTOP REQUEST.
IEF196I IEF142I CTWTR CTWTR - STEP WAS EXECUTED - COND CODE 0000
IEF196I IEF285I   SYS1.SC32.CTRACE CATALOGED
```

After the packet trace data is captured, the trace data set created by the external writer procedure is saved and we use IPCS to format and analyze its contents. Refer to *z/OS CS: IP Diagnosis,* GC31-8782, for further details about packet trace.

Socket data trace

Using the SYSTCPDA component CS for z/OS IP provides a way to capture socket data into and out of the Physical File System (PFS). It helps to diagnose application data-related problems. To activate this trace, we follow the same steps we used to activate a packet trace and change only the command to start and stop the socket data trace:

```
V TCPIP,tcpproc,DATTRACE,ON
V TCPIP,tcpproc,DATTRACE,OFF
```

8.4.4 OMPROUTE trace (SYSTCPRT)

To diagnose OMPROUTE problems, z/OS Communications Server provides the debug and trace parameter that can be defined during OMPROUTE initialization. The resulting output is written to the OMPROUTE log and can cause increased overhead. This performance issue can be solved by using the CTRACE facility. To do so, we highly recommend that you use the 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 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.

In addition to 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.

Here we shown the necessary steps to start the CTRACE for OMPROUTE during OMPROUTE initialization using the parmlib member CTIORA00 and directing the trace output to an external writer.

1. Prepare the SYS1.PARMLIB member CTIORA00 to get the desired output data. Example 8-13 on page 234 shows a sample of CTIORA00 contents.

Example 8-13 CTIORA00 sample

```
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 100M                           */
/*            CTRACE buffers reside in OMPROUTE Private storage       */
/*            which is in the regions address space.                  */
/* ------------------------------------------------------------------ */
         BUFSIZE(50M)
         WTR(CTWTR)
/* ------------------------------------------------------------------ */
/*   OPTIONS: NAMES OF FUNCTIONS TO BE TRACED, OR "ALL"               */
/* ------------------------------------------------------------------ */
/*       OPTIONS(                                                     */
/*              'ALL     '                                            */
/*               ,'MINIMUM '                                         */
/*               ,'ROUTE '                                           */
/*               ,'PACKET '                                          */
/*               ,'OPACKET '                                         */
/*               ,'RPACKET '                                         */
/*               ,'IPACKET '                                         */
/*               ,'SPACKET '                                         */
/*               ,'DEBUGTRC'                                         */
/*                                 )                                */
```

2. Start the OMPROUTE procedure using the desired Debug and Trace options, as shown in Example 8-14.

Example 8-14 OMPROUTE procedure

```
///OMPC32 PROC STDENV=OMPENC&SYSCLONE
//OMPC32 EXEC PGM=OMPROUTE,REGION=OM,TIME=NOLIMIT,
//       PARM=('POSIX(ON) ALL31(ON)',
//            'ENVAR("_BPXK_SETIBMOPT_TRANSPORT=TCPIPC"',
//            '"_CEE_ENVFILE=DD:STDENV")/-d1 -t2')1
//STDENV   DD DISP=SHR,DSN=TCPIPC.TCPPARMS(&STDENV)
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//CEEDUMP  DD SYSOUT=*,DCB=(RECFM=FB,LRECL=132,BLKSIZE=132)
```

The description for the tag shown in Example 8-14 is as follows:

- 1 The parameters **-t** (trace) and **-d** (debug) define how detailed we want the output data to be. We recommend using **-t2** and **-d**1.

3. Verify that CTRACE has been started as expected, issuing the console command as shown:

```
D TRACE,COMP=SYSTCPRT,SUB=(OMPC)
IEE843I 16.31.37  TRACE DISPLAY 058
        SYSTEM STATUS INFORMATION
 ST=(ON,0256K,00512K) AS=ON  BR=OFF EX=ON  MO=OFF MT=(ON,024K)
  TRACENAME
  =========
  SYSTCPRT
                      MODE BUFFER HEAD SUBS
                      =====================
                      OFF         HEAD    1
     NO HEAD OPTIONS
  SUBTRACE            MODE BUFFER HEAD SUBS
 ------------------------------------------------------------
  OMPC                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 shown:

```
TRACE CT,ON,COMP=SYSTCPRT,SUB=(OMPC)
R 66,OPTIONS=(ALL),END
IEE600I REPLY TO 066 IS;OPTIONS=(ALL),END
ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
WERE SUCCESSFULLY EXECUTED.
```

5. Reproduce the problem.

6. Disconnect the external writer.

```
TRACE CT,ON,COMP=SYSTCPRT,SUB=(OMPC)
067 ITT006A SPECIFY OPERAND(S) FOR TRACE CT COMMAND.
R 67,WTR=DISCONNECT,END
IEE600I REPLY TO 067 IS;WTR=DISCONNECT,END
ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
WERE SUCCESSFULLY EXECUTED.
```

7. Stop the component trace.

```
TRACE CT,OFF,COMP=SYSTCPRT,SUB=(OMPC)
ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
WERE SUCCESSFULLY EXECUTED.
```

8. Stop the external writer.

```
TRACE CT,WTRSTOP=CTWTR
ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND
WERE SUCCESSFULLY EXECUTED.
ITT111I CTRACE WRITER CTWTR TERMINATED BECAUSE OF A WTRSTOP REQUEST.
IEF196I IEF142I CTWTR CTWTR - STEP WAS EXECUTED - COND CODE 0000
IEF196I IEF285I   SYS1.SC32.CTRACE CATALOGED
```

9. Change the OMPROUTE Debug and Trace level to avoid performance problems using the MODIFY command, as shown in:

```
F OMPC,TRACE=0
EZZ7866I OMPROUTE MODIFY COMMAND ACCEPTED
```

```
F OMPC,DEBUG=0
EZZ7866I OMPROUTE MODIFY COMMAND ACCEPTED
```

After these steps, the generated trace file must be formatted using the IPCS. For further information about OMPROUTE diagnosis, refer to *z/OS Communications Server: IP Diagnosis Guide*, GC31-8782.

8.4.5 Resolver trace (SYSTCPRE)

z/OS Communications Server 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. In addition to specifying the trace options, you can change the Resolver trace buffer size. Note that 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 8.4.1, “Taking a component trace” on page 227 and, in step 2 on page 227, specify the **comp=** parameter with the resolver component name, SYSTCPRE and the **sub=** parameter with the Resolver proc_name.

The generated trace file created after the problem is reproduced must be formatted using the IPCS. For further information about Resolver diagnosis, refer to *z/OS Communications Server: IP Diagnosis Guide*, GC31-8782.

8.4.6 IKE daemon trace (SYSTCPIK)

z/OS Communications Server provides component trace support for the IKE daemon and, as other components, a default minimum component trace is always started during IKE daemon initialization. Use a parmlib member to customize the parameters that are used to initialize the trace. The default IKE daemon component trace parmlib member is the SYS1.PARMLIB member CTIIKE00. The parmlib member name can be changed using the IKED_CTRACE_MEMBER environment variable.

Tip: The IKE daemon reads the IKED_CTRACE_MEMBER environment variable only during initialization. Changes to IKED_CTRACE_MEMBER after daemon initialization have no affect. After IKE daemon initialization, you must use the TRACE CT command to change component trace options.

After IKE daemon is initialized you can start CTRACE to modify trace options or send data to a external writer, using the commands listed in 8.4.1, “Taking a component trace” on page 227 and, in step 2 on page 227, specify the **comp=** parameter with the IKE daemon component name, SYSTCPIK and the **sub=** parameter with the iked proc_name.

The generated trace file created after the problem is reproduced must be formatted using the IPCS. For further information about IKE daemon diagnosis, refer to *z/OS Communications Server: IP Diagnosis Guide*, GC31-8782.

8.4.7 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. Refer to “Intrusion Detection Services” in *z/OS Communications Server: IP Configuration Guide*, SC31-8775, for information about defining policy.

If the EZZ4210I message indicates the parmlib member name CTIIDS00, then the IDS CTRACE space is set up using the default BUFSIZE of 32 M.

The CTIIDS00 member is used to specify the IDS CTRACE parameters. To eliminate this message, ensure that a CTIIDS00 member exists within Parmlib and that the options are correctly specified. A sample CTIIDS00 member is shipped with z/OS Communications Server.

See *z/OS Communications Server: IP Diagnosis Guide*, GC31-8782, for details about the intrusion detection services trace and *z/OS Communications Server: IP Configuration Guide*, SC31-8775, for information about defining policy.

8.4.8 OSAENTA trace (SYSTCPOT)

TCP/IP Services component trace is also available for use with the OSA-Express Network Traffic Analyzer (OSAENTA) trace facility. The OSAENTA trace is a diagnostic method for obtaining frames flowing to and from an OSA adapter. You can use the OSAENTA statement to copy frames as they enter or leave an OSA adapter for an attached host. The host can be an LPAR with z/OS, VM, or Linux. For more information about OSAENTA, refer to 8.5, “OSA-Express2 Network Traffic Analyzer” on page 239.

8.4.9 Queued Direct I/O diagnostic synchronization

Communications Server provides support for a new Queued Direct I/O Diagnostic Synchronization (QDIOSYNC) facility. It provides the ability to synchronize OSA-Express2 diagnostic data with host diagnostic data. The QDIOSYNC facility also provides for optional filtering of the OSA-Express2 diagnostic data. If a filter is specified, the OSA-Express2 adapter honors the filter by limiting the types of diagnostic data collected. Although the QDIOSYNC trace differs from a traditional VTAM TRACE command, you use the VTAM MODIFY TRACE and NOTRACE commands to control it. The DISPLAY TRACES command is modified to show the state of the QDIOSYNC trace.

The QDIOSYNC trace is not a traditional trace in which output is generated based on specific events. Instead, the QDIOSYNC trace freezes and captures (logs) OSA-Express2 diagnostic data in a timely manner. In addition to (or instead of) using the hardware management console (HMC) to manually capture the diagnostic data, you can arm the OSA-Express2 adapter to automatically capture diagnostic data when one of the following occurs:

- The OSA-Express2 adapter detects an unexpected loss of host connectivity.

  Unexpected loss of host connectivity occurs when the OSA-Express2 adapter receives an unexpected halt signal from the host or when the host is unresponsive to OSA requests.
- The OSA-Express2 adapter receives a CAPTURE signal from the host.

  A CAPTURE signal is sent by the host when one of the following occurs:

  – The VTAM-supplied message processing facility (MPF) exit (IUTLLCMP) is driven.

  – Either the VTAM or TCP/IP functional recovery routine (FRR) is driven with ABEND06F.

    (ABEND06F is the result of a SLIP PER trap that specifies ACTION=RECOVERY).

When arming an OSA-Express2 adapter for QDIOSYNC, you can specify an optional filter that alters what type of diagnostic data is collected by the OSA-Express2 adapter. This filtering reduces the overall amount of diagnostic data collected, and therefore decreases the likelihood that pertinent data is lost.

Note: Do not use QDIOSYNC to unconditionally arm an OSA-Express2 adapter when it is shared by other operating systems and those operating systems might use this function. In this case, the function should be coordinated between all sharing operating systems.

If you have several OSAs to arm, but you do not want to arm all of them, consider first arming all OSAs and then individually disarm those you do not want armed.

For more information about how to set up the trace, refer to:

- *z/OS Communications Server: SNA Diagnosis Vol. 1, Techniques and Procedures,* GC31-6850
- *z/OS Communications Server: SNA Operation*, SC31-8779
- *MVS Installation Exits*, SA22-7593

8.4.10 Network security services (NSS) server trace (SYSTCPNS)

z/OS Communications Server provides component trace support for the Network Security Services (NSS) and, as with other components, a default minimum component trace is always started during NSS server initialization. Use a parmlib member to customize the parameters that are used to initialize the trace. The default NSS server component trace parmlib member is the SYS1.PARMLIB member CTINSS00. In addition to specifying the trace options, you can also change the NSS trace buffer size. The buffer size can be changed only at NSS initialization and has a maximum of 256 MB.

You can change the parmlib member name using the NSSD_CTRACE_MEMBER environment variable.

Tip: The NSS server reads the NSSD_CTRACE_MEMBER environment variable *only* during initialization. Changes to NSSD_CTRACE_MEMBER after server initialization have no effect.

After the NSS server is initialized, you can start CTRACE to modify trace options or send data to a external writer by using the commands listed in 8.4.1, “Taking a component trace” on page 227 and, in step 2 on page 227, specify the **comp=** parameter with the NSS server component name, SYSTCPNS and the **sub=** parameter with the nss_proc_name.

The generated trace file created after the problem iss reproduced must be formatted using the IPCS. For more information about NSS server diagnosis, refer to *z/OS Communications Server: IP Diagnosis Guide*, GC31-8782.

8.4.11 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, because

it contains the trace data for the SYSTCPIP, SYSTCPDA, and SYSTCPIS components. Be sure to include “region” (RGN) in the SDATA dump options, as shown here:

```
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 shown here:

```
DUMP COMM=(enter_dump _title_here)
Rxx,JOBNAME=proc_started_task_name,SDATA=(RGN,CSA,ALLPSA,SQA,SUM,TRT,ALLNUC),END
```

For more information about how to get a dump, refer to *z/OS Communications Server: IP Diagnosis Guide*, GC31-8782.

8.4.12 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 can use in diagnosing problems. There is little information in the documentation on interpreting trace data. If you want to analyze the packet trace or data trace, you can do so by formatting the trace data using a z/OS tool in TSO called IPCS. For more information about trace and dump analysis using IPCS, refer to *z/OS Communications Server: IP Diagnosis Guide*, GC31-8782.

8.4.13 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, refer to *z/OS Communications Server: IP Diagnosis Guide*, GC31-8782.

8.5 OSA-Express2 Network Traffic Analyzer

When data problems occur in a LAN environment, multiple traces are usually required. A sniffer trace might be required to see the data as it was received from or sent to the network. An OSA hardware trace might be required if the problem is suspected in the OSA, and z/OS Communications Server traces are required to diagnose VTAM or TCP/IP problems.

To assist in problem diagnosis, the OSA-Express network traffic analyzer (OSAENTA) function provides a way to trace inbound and outbound frames for an OSA-Express2 feature. The OSAENTA trace function is controlled and formatted by z/OS Communications Server, but is collected in the OSA at the network port.

Note: To enable the OSA-Express network traffic analyzer, you must be running at least an IBM System z9 EC or z9 BC and OSA-Express2 feature in QDIO mode (CHPID type OSD). See the 2094DEVICE Preventive Service Planning (PSP) and the 2096DEVICE Preventive Service Planning (PSP) buckets for more information about these topics.

This section discusses the steps that are necessary for setting up and using OSAENTA:

- Determining the microcode level for OSA-Express2
- Defining TRLE definitions
- Checking TCPIP definitions
- Customizing OSA-Express Network Traffic Analyzer (NTA)
- Defining a resource profile in RACF
- Allocating a VSAM linear data set
- Starting the OSAENTA trace

8.5.1 Determining the microcode level for OSA-Express2

There are two ways to determine the OSA-Express2 microcode level: from the Hardware Management Console (HMC), or by issuing the **D NET,TRL,TRLE=OSA2080P** command. Each method is discussed in more detail in this section.

- From the HMC:
  - Select your system.
  - Double-click **OSA Advanced Facilities**.
  - Select appropriate PCHID.
  - Select **View code level**.

Figure 8-7 shows the microcode level installed in one of our OSA-Express2 features.

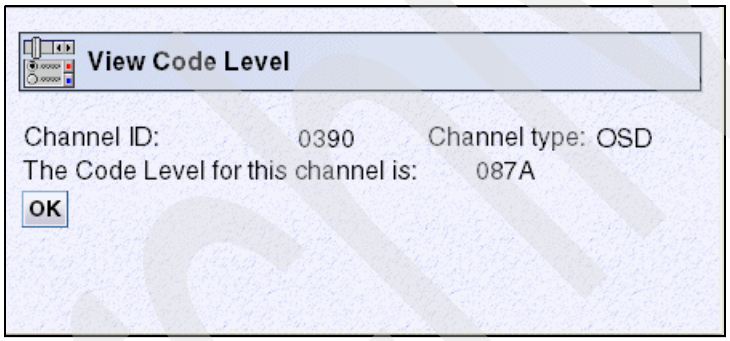


Figure 8-7 View code level

- Or, you can issue the **D NET,TRL,TRLE=OSA2080P** command; Example 8-15 shows the output.

Example 8-15 Output Display TRL

```
NAME = OSA2080P, TYPE = TRLE 756
 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 = 087A
 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
```

```
DEVICEID PARAMETER FOR OSAENTA TRACE COMMAND = 02-03-00-02
UNITS OF WORK FOR NCB AT ADDRESS X'0F4E7010'
P1 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
P2 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
P3 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
P4 CURRENT = 0 AVERAGE = 2 MAXIMUM = 3
TRACE DEV = 2083 STATUS = RESET      STATE = N/A
```

8.5.2 Defining TRLE definitions

Use the **D U,,,2080,16** command to ensure that you have defined enough devices; see Example 8-16.

Example 8-16 Verifying the number of OSA devices

```
D U,,,2080,16
IEE457I 16.50.55 UNIT STATUS 833
 UNIT TYPE STATUS        VOLSER     VOLSTATE
 2080 OSA  A-BSY
 2081 OSA  A
 2082 OSA  A-BSY
 2083 OSA  O
 2084 OSA  O
 2085 OSA  O
 2086 OSA  O
 2087 OSA  O
 2088 OSA  O
 2089 OSA  O
 208A OSA  O
 208B OSA  O
 208C OSA  O
 208D OSA  O
 208E OSA  O
 208F OSAD O-RAL
```

The OSA-Express2 needs an additional “DATAPATH” statement on the TRL (see Example 8-17).

Example 8-17 TRL definition

```
OSA2080  VBUILD TYPE=TRL
OSA2080P TRLE  LNCTL=MPC,
               READ=2080,
               WRITE=2081,
               DATAPATH=(2082-208E),
               PORTNAME=OSA2080,
               MPCLEVEL=QDIO
```

8.5.3 Checking TCPIP definitions

An excerpt of TCP/IP profile, displayed in Example 8-18, shows the information that will be need when starting the OSAENTA trace in a later step. Keep this information handy.

Example 8-18 TCP/IP definitions

```
;OSA DEFINITION
DEVICE OSA2080    MPCIPA
LINK   OSA2080L   IPAQENET    OSA2080 VLANID 10
HOME
```

```
   10.1.2.11      OSA2080L

START OSA2080
```

After TCP/IP is started, you can also see the OAT entries using OSA/SF (see Example 8-19 on page 242).

Example 8-19 OAT entries

```
                              Image 2.3 (A23     ) CULA 0
00(2080)* MPC       N/A     OSA2080P   (QDIO control)        SIU    ALL
02(2082)  MPC  00 No4  No6  OSA2080P   (QDIO data)           SIU    ALL
                  VLAN   10   (IPv4)

                  Group Address   Multicast Address
                  01005E000001    224.000.000.001

                  VMAC            IP address
          HOME    00096B1A7490    010.001.000.010
          HOME    00096B1A7490    010.001.001.010
          HOME    00096B1A7490    010.001.002.010
          HOME    00096B1A7490    010.001.002.011
          REG     00096B1A7490    010.001.002.012
          REG     00096B1A7490    010.001.003.011
          REG     00096B1A7490    010.001.003.012
          REG     00096B1A7490    010.001.004.011
          REG     00096B1A7490    010.001.005.011
          REG     00096B1A7490    010.001.006.011
          REG     00096B1A7490    010.001.007.011
          REG     00096B1A7490    010.001.008.010
          REG     00096B1A7490    010.001.008.020

03(2083)  N/A                                         N/A    CSS
```

8.5.4 Customizing OSA-Express Network Traffic Analyzer (NTA)

Use this task to select an OSA-Express Network Traffic Analyzer (NTA) support element control, to customize the OSA-Express NTA settings in Advanced Facilities, or to check the current OSA-Express NTA authorization.

Customizing OENTA for the support element:

- Allows the support element to set up the OSA LAN Analyzer traces and capture data to the support element hard disk
- Allows the support element to change authorization to allow host operating systems to enable the NTA traces outside their own partition

Note: The OSA-Express Network Traffic Analyzer is mutually exclusive with the OSA LAN Analyzer for tracing on a specified CHPID. Only one or the other can be enabled for a specified CHPID at any one time.

1. Log on to the Support Element (SE) on the Hardware Management Console (HMC) through Single Object Operations (SOO).

Important: Enabling the OENTA support could allow tracing of sensitive information. Therefore, the user ID used to do the following steps must have the “Access Administrator Tasks” role assigned.

2. Select the CPC you want to work with, as shown in Figure 8-8 on page 243.

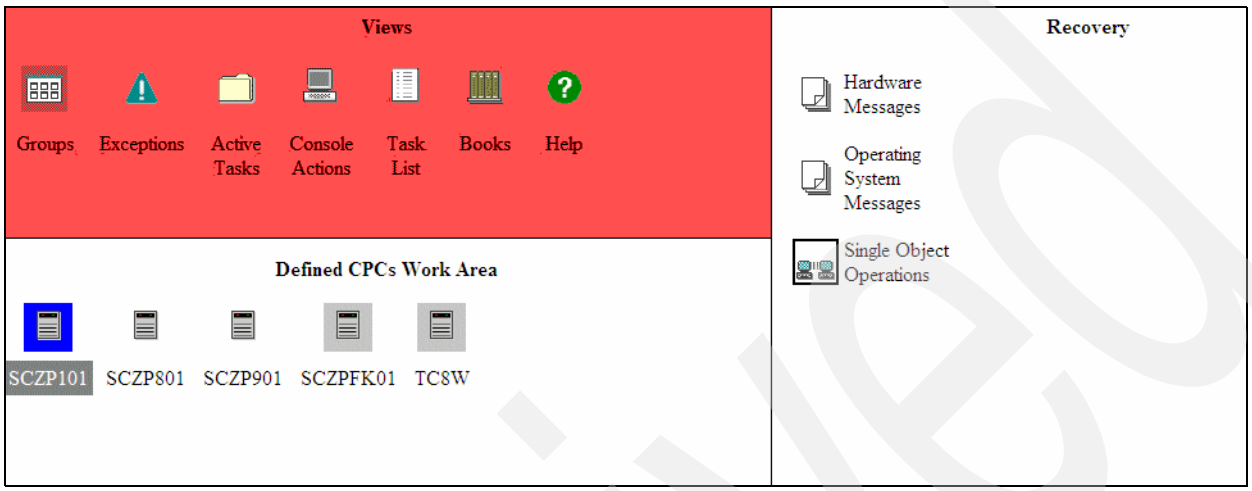


Figure 8-8 From the HMC, log on to SE

3. Select and open the Service task list; see Figure 8-9.

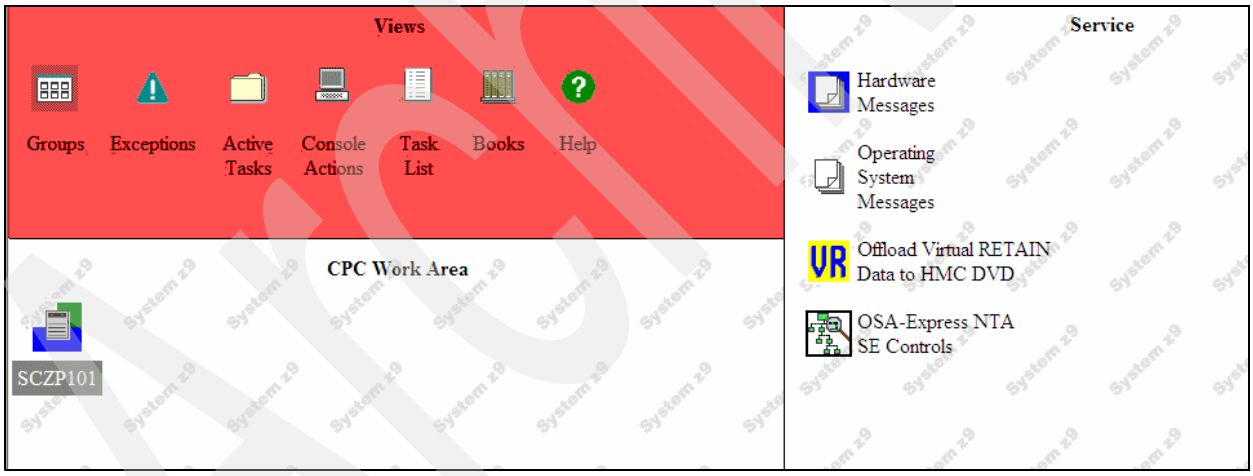


Figure 8-9 OSA-Express NTA

4. Double-click the OSA-Express NTA SE Controls task; see Figure 8-10.

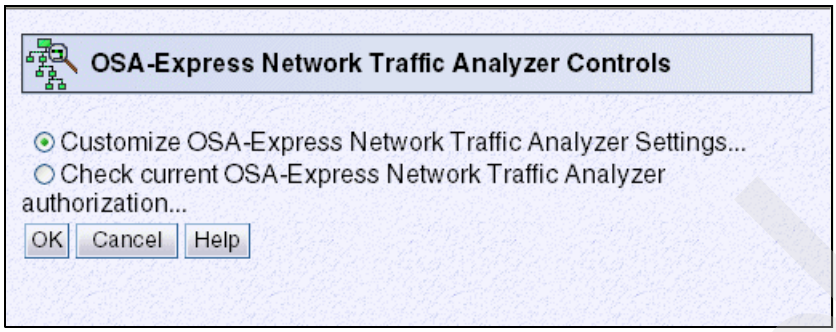


Figure 8-10 OSA NTA Controls

5. Select the control to work with:
   - Customize OSA-Express Network Traffic Analyzer Settings... provides the capability to allow or disallow the support element to change authorization to allow host operating systems to enable the Network Traffic Analyzer to trace outside their own partition.
   - Check current OSA-Express Network Traffic Analyzer authorization... allows the support element to scan all the OSAs and reports back which OSAs are authorized for NTA to trace outside its own partition.
6. Click OK to change the current OSA-Express NTA control; see Figure 8-11.

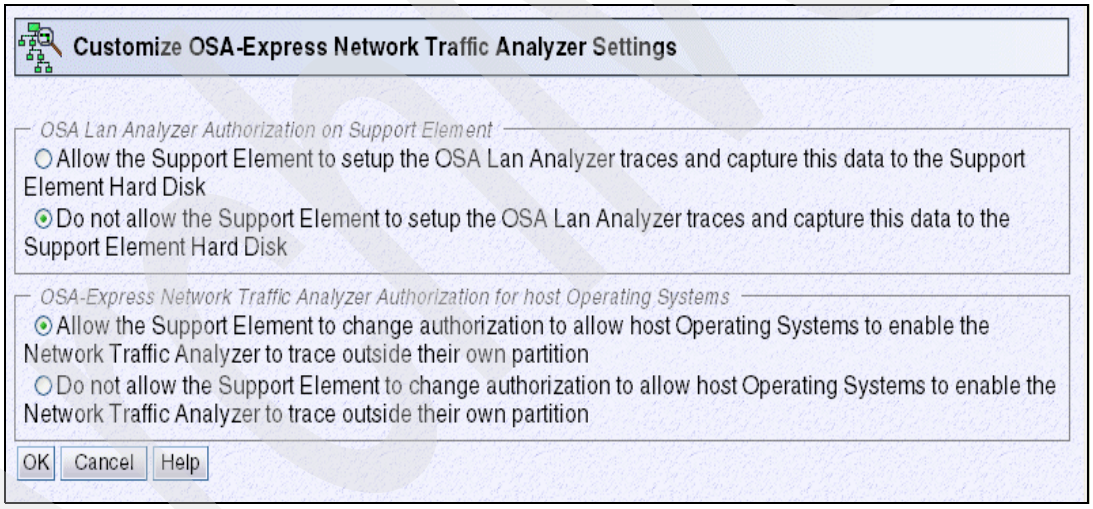


Figure 8-11 Change the current OSA-Express NTA control

7. Click **Allow the Support Element to allow Host Operating System to enable NTA.**
8. Click **OK**; see Figure 8-12.

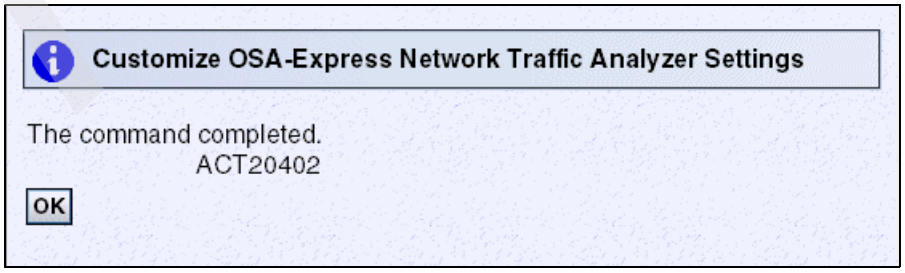


Figure 8-12 Command completed

9. Log off from the SE and from the HMC.
10. Log on to the SE on the HMC through SOO (Single Object Operations) using the SYSPROG user ID.
11. Select **Channels work area** (on the left side of the screen) **Channel Operation** (on the right side of the screen).
12. Select the channel that you want to manage (see Figure 8-13 on page 245).

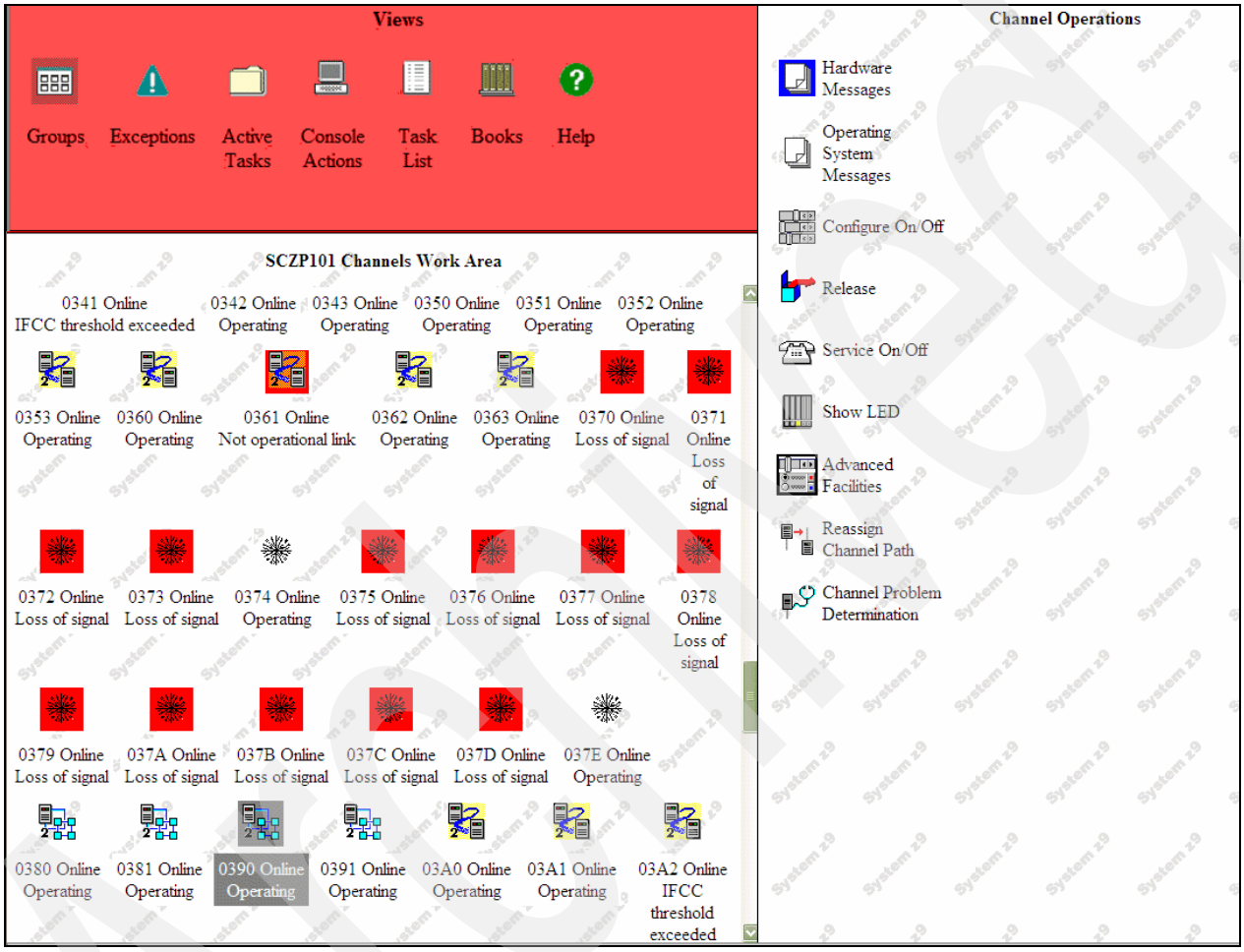


Figure 8-13 Channel Operations menu

13. In our case we selected PCHID 0390 (CHPID 02). We double-clicked **Advanced Facilities**; see Figure 8-14.

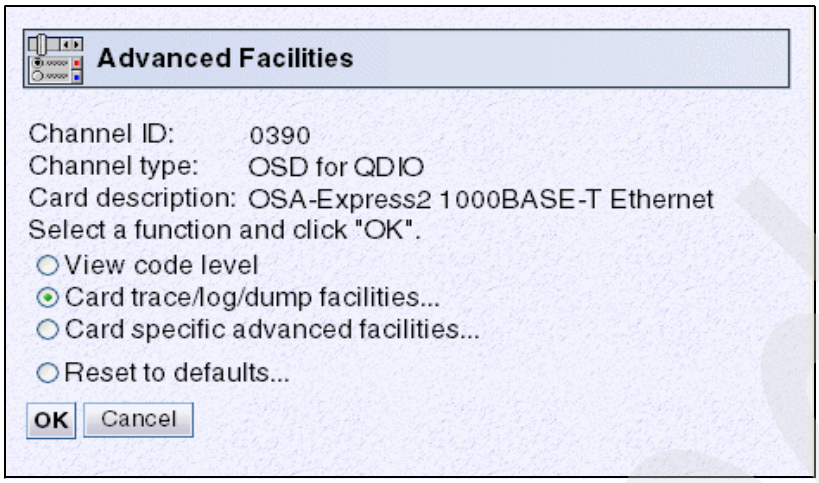


Figure 8-14 Advanced Facilities options

14. Select C**ard trace/log/dump facilities...**, then click **OK**; see Figure 8-15 on page 246.

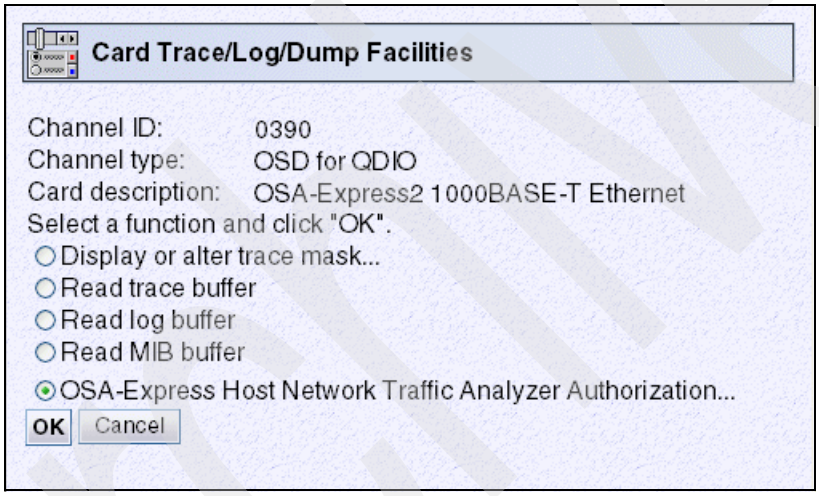


Figure 8-15 Card Trace/Log/Dump Facilities

15. Select **OSA-Express Host Network Traffic Analyzer Authorization**, then click **OK**; see Figure 8-16.

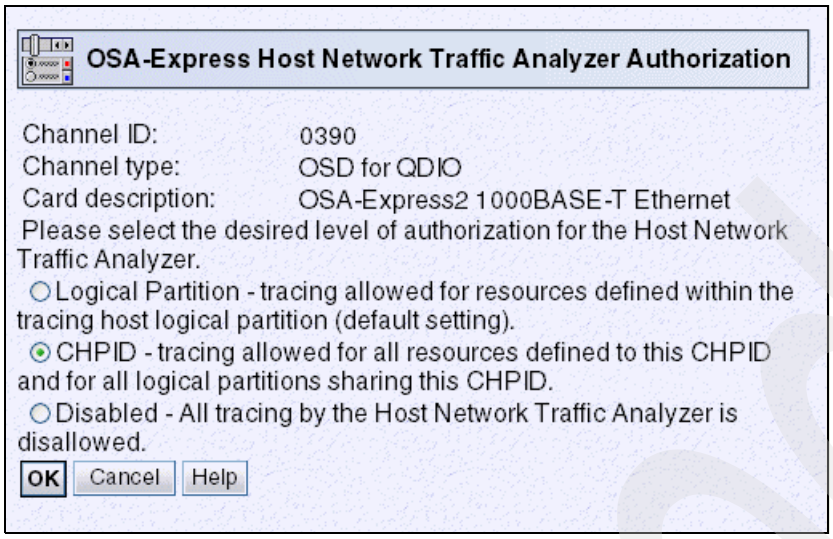


Figure 8-16 NTA Authorization

16. If your CHPID is shared between several LPARs, we suggest you take the second option shown in Figure 8-16, then click **OK**. Figure 8-17 shows the results.

Figure 8-17 Command completed

17. To verify if the command has been set as required:
    - Log off the SYSPROG user ID.
    - Log on to the SE on the HMC through SOO; see Figure 8-8 on page 243.

> **Important:** For checking the authorization of OENTA support, the user ID must have the Access Administrator Tasks role assigned.

    - Select **Check current OSA-Express Network Traffic Analyzer Authorization**, as shown in Figure 8-10 on page 244.
    - Click **OK**; see Figure 8-18.

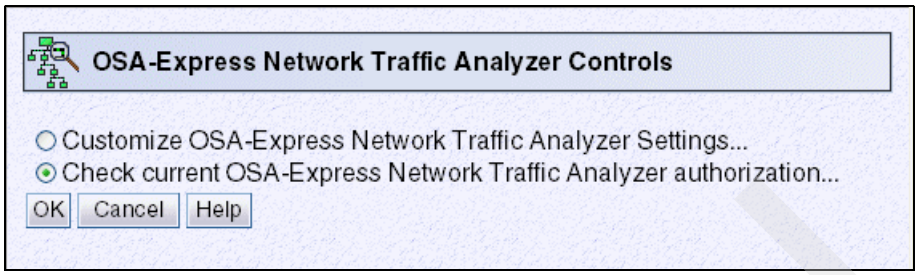


Figure 8-18 OSA-Express NTA controls

18. Figure 8-19 shows that PCHID 0390 is allowed to be traced.

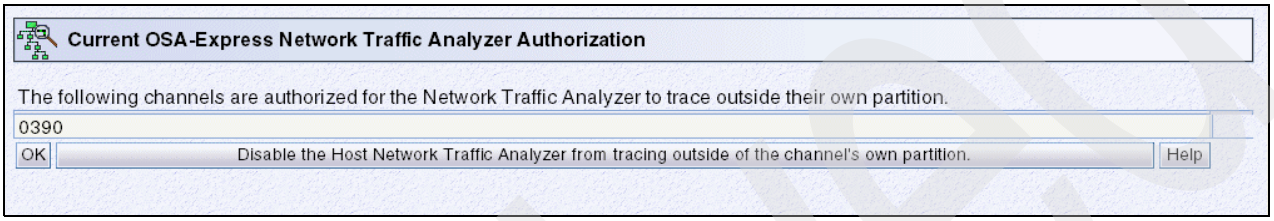


Figure 8-19 PCHID NTA Authorization

8.5.5 Defining a resource profile in RACF

See Example 8-20 for the RACF commands needed to allow users to issue the VARY TCPIP.... command.

Example 8-20 RACF commands

```
RDEFINE OPERCMDS MVS.VARY.TCPIP.OSAENTA UACC(NONE) PERMIT MVS.VARY.TCPIP.OSAENTA
ACCESS(CONTROL) CLASS(OPERCMDS) ID(CSO3) SETR GENERIC(OPERCMDS) REFRESH SETR
RACLIST(OPERCMDS) REFRESH
```

8.5.6 Allocating a VSAM linear data set

Example 8-21 shows how to create the VSAM linear dataset. This VSAM linear dataset is “Optional; however, we recommend its use.

Example 8-21 Allocate VSAM linear dataset

```
//DEFINE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DELETE +
(CSO3.CTRACE.LINEAR) +
CLUSTER
DEFINE CLUSTER( +
NAME(CSO3.CTRACE.LINEAR) +
LINEAR +
MEGABYTES(10) +
VOLUME(CPDLBO) +
CONTROLINTERVALSIZE(32768) +
) +
DATA( +
NAME(CSO3.CTRACE.DATA) +
)
LISTCAT ENT(USER41.CTRACE.LINEAR)
```

```
ALL
```

8.5.7 Starting the OSAENTA trace

The OSAENTA statement dynamically defines a QDIO interface to the OSA-Express being traced, called an OSAENTA interface. That interface is used exclusively for capturing OSA-Express Network Traffic Analyzer traces.

The OSAENTA statement enables an installation to trace data from other hosts connected to OSA-Express.

Important: The trace data collected should be considered confidential and TCP/IP system dumps and external trace files containing this trace data should be protected.

To see the complete syntax of the OSAENTA command, refer to *z/OS Communications Server: IP Configuration Reference*, SC31-8776.

Components involved for z/OS CTRACE

The CTRACE component for collecting NTA trace data is called SYSTCPOT. The member in SYS1.PARMLIB is named CTINTA00. This member is used to define the size of the buffer space in the TCPIPDS1 data space reserved for OSAENTA CTRACE. The size can range from 1 M to 624 M, with a default of 64 M.

Note: Update CTINTA00 to set the CTRACE buffer size. Keep in mind that this will use up auxiliary page space storage.

Using the OSAENTA command

An internal interface is created when PORTNAME is defined on the OSAENTA statement. The dynamically-defined interface name is EZANTA concatenated with the port name. These EZANTA interfaces are displayed at the end of the NETSTAT DEV output.

When the ON keyword of the OSAENTA parameter is used, VTAM allocates the next available TRLE data path associated with the port. This data path is used only for inbound trace data.

When the OFF keyword of the OSAENTA parameter is used (or the trace limits of the TIME, DATA, or FRAMES keyword are reached), the data path is released.

Setting the OSAENTA traces

You can set the OSAENTA trace in two ways: by coding the OSAENTA statement in the profile TCP/IP, or by issuing a command in z/OS. These methods are explained in this section.

- To code the OSAENTA statement in the profile TCP/IP, see Example 8-22.

Example 8-22 TCP/IP profile

```
; set up the filters to trace for TCP packets on PORT 2323 with a source
;or destination
; IP address of 10.1.2.11 over MAC address 00096B1A7490
OSAENTA PORTNAME=OSA2080 PROT=TCP IP=10.1.2.11 PORTNUM=2323
OSAENTA PORTNAME=OSA2080 MAC=00096B1A7490
; activate the tracing (the trace will self-deactivate after 20,000 frames)
OSAENTA PORTNAME=OSA2080 ON FRAMES=20000
```

```
; deactivate the tracing
OSAENTA OFF PORTNAME=OSA2080
```

In this case, OSAENTA traces the portname OSA2080 only for traffic matching the following filters:

- Protocol = UDP
- IP address = 10.1.2.11
- Port number = 2323

There are seven filters available to define the packets to be captured:

- MAC address
- VLAN ID
- Ethernet frame type
- IP address (or range)
- IP protocol
- Device ID
- TCP/UDP

Note: Use filters to limit the trace records to prevent over consummation of the OSA CPU resources, the LPAR CPU resources, the TCPIPDS1 trace data space, memory, auxiliary page space and the IO subsystem writing trace data to disk.

- To issue the following command in z/OS:

  ```
  V TCPIP,TCPIPA,OSAENTA,ON,PORTNAME=OSA2080,IP=10.1.2.11,PORTNUM=2323
  ```

  The messages you receive in response to this command are shown in Figure 8-20.

```
RESPONSE=SC30      EZZ0060I PROCESSING COMMAND: VARY TCPIP,TCPIPA,OSAENTA,ON,

RESPONSE=PORTNAME=OSA2080,IP=10.1.2.11,PORTNUM=2323
RESPONSE=SC30      EZZ0053I COMMAND VARY OSAENTA COMPLETED SUCCESSFULLY
```

Figure 8-20 OSAENTA results

Important: If you receive ERROR CODE **0003** it means that an attempt was made to enable OSA-Express Network Traffic Analyzer (OSAENTA) tracing for a specified OSA, but the current authorization level does not permit it.

Refer to 8.5.4, “Customizing OSA-Express Network Traffic Analyzer (NTA)” on page 242 for directions about how to change the authorization to allow OSAENTA to be used on this specified OSA. Also read *Support Element Operations Guide,* SC28-6860, for complete information about this topic.

The command NETSTAT DEVLINKS has been enhanced to show the OSAENTA definition; see Example 8-23.

Example 8-23 Netstat Devlinks command output

```
OSA-EXPRESS NETWORK TRAFFIC ANALYZER INFORMATION:
  OSA PORTNAME: OSA2080           OSA DEVSTATUS:      READY
    OSA INTFNAME: EZANTAOSA2080   OSA INTFSTATUS:     READY
    OSA SPEED:    1000            OSA AUTHORIZATION: CHPID
    OSAENTA CUMULATIVE TRACE STATISTICS:
      DATAMEGS:   0                     FRAMES:          0
      DATABYTES:  0                     FRAMESDISCARDED: 0
```

```
  FRAMESLOST: 0
OSAENTA ACTIVE TRACE STATISTICS:
  DATAMEGS:   0                    FRAMES:          0
  DATABYTES:  0                    FRAMESDISCARDED: 0
  FRAMESLOST: 0                    TIMEACTIVE:      0
OSAENTA TRACE SETTINGS:          STATUS: ON
  DATAMEGSLIMIT: 1024              FRAMESLIMIT:     2147483647
  ABBREV:        224               TIMELIMIT:       10080
  DISCARD:       EXCEPTION
OSAENTA TRACE FILTERS:           NOFILTER: NONE
  DEVICEID: *
  MAC:      *
  VLANID:   *
  ETHTYPE:  *
  IPADDR:   10.1.2.11/32
  PROTOCOL: * TCP
  PORTNUM:  * 2323
```

The NETSTAT display for devices shows the Network Traffic Analyzer interfaces. The interface name has prefixed the OSA port name with EZANTA.

To display a specific NTA interface, use the INTFName=EZANTAosaportname keyword.

Traces are placed in an internal buffer, which can then be written out using a CTRACE external writer. The MVS TRACE command must also be issued for component SYSTCPOT to activate the OSAENTA trace.

Attention: If you receive ERROR CODE **0005** it means that an attempt was made to enable OSA-Express Network Traffic Analyzer tracing for a specified OSA that already has either OSAENTA or OSA LAN Analyzer tracing enabled elsewhere on the system for this OSA.

Only one instance of active tracing (either OSAENTA or LAN Analyzer) for a specified OSA is permitted on the system at any one time.

When the trace is started from OSA/SF, you can see that another device has been allocated for trace; see Example 8-24.

Example 8-24 OAT with OSAENTA started

```
Image 2.3 (A23     ) CULA 0
00(2080)* MPC      N/A     OSA2080P   (QDIO control)        SIU    ALL
02(2082)  MPC  00 No4  No6  OSA2080P   (QDIO data)           SIU    ALL
               VLAN   10   (IPv4)

               Group Address   Multicast Address
               01005E000001    224.000.000.001

               VMAC            IP address
         HOME  00096B1A7490    010.001.000.010
         HOME  00096B1A7490    010.001.001.010
         HOME  00096B1A7490    010.001.002.010
         HOME  00096B1A7490    010.001.002.011
         REG   00096B1A7490    010.001.002.012
         REG   00096B1A7490    010.001.003.011
         REG   00096B1A7490    010.001.003.012
         REG   00096B1A7490    010.001.004.011
         REG   00096B1A7490    010.001.005.011
```

```
          REG     00096B1A7490    010.001.006.011
          REG     00096B1A7490    010.001.007.011
          REG     00096B1A7490    010.001.008.010
          REG     00096B1A7490    010.001.008.020

03(2083)  MPC  00 No4  No6  OSA2080P   (QDIO data)          SIU     ALL
```

You can also use the `D NET,TRL,TRLE=OSA2080` command, as shown in Example 8-25.

Example 8-25 Output Display TRLE

```
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 = 087A
 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
 DEVICEID PARAMETER FOR OSAENTA TRACE COMMAND = 02-03-00-02
 UNITS OF WORK FOR NCB AT ADDRESS X'0F4E7010'
 P1 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
 P2 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
 P3 CURRENT = 0 AVERAGE = 0 MAXIMUM = 0
 P4 CURRENT = 0 AVERAGE = 2 MAXIMUM = 3
 TRACE DEV = 2083 STATUS = ACTIVE     STATE = N/A
```

Starting the CTRACE

- Start the external writer (CTRACE writer).

  ```
  TRACE CT,WTRSTART=CTWRT
  ```

- Start the CTRACE and connect to the external writer.

  ```
  TRACE CT,ON,COMP=SYSTCPOT,SUB=(TCPIPA)
  R xx,WTR=CTWTR,END
  ```

- Display the active component trace options with this command:

  ```
  DISPLAY TRACE,COMP=SYSTCPOT,SUB=(TCPIPA)
  ```

  Example 8-26 shows the output of this command.

Example 8-26 Display Trace output

```
RESPONSE=SC30
 IEE843I 16.45.15  TRACE DISPLAY 165
         SYSTEM STATUS INFORMATION
  ST=(ON,0256K,00512K) AS=ON  BR=OFF EX=ON  MO=OFF MT=(ON,024K)
   TRACENAME
   =========
   SYSTCPOT
                        MODE BUFFER HEAD SUBS
                        ====================
```

```
                    OFF         HEAD    2
     NO HEAD OPTIONS
  SUBTRACE            MODE BUFFER HEAD SUBS
 ------------------------------------------------------------
  TCPIPA              ON   0128M
     ASIDS      *NONE*
     JOBNAMES   *NONE*
     OPTIONS    MINIMUM
     WRITER     CTWTR
```

To display information about the status of the component trace for all active procedures, issue the following command:

```
DISPLAY TRACE,COMP=SYSTCPOT,SUBLEVEL,N=8
```

Example 8-27 displays the output.

Example 8-27 Status of Component Trace

```
IEE843I 10.35.04  TRACE DISPLAY 821
        SYSTEM STATUS INFORMATION
 ST=(ON,0256K,00512K) AS=ON  BR=OFF EX=ON  MO=OFF MT=(ON,024K)
  TRACENAME
  =========
  SYSTCPOT
                      MODE BUFFER HEAD SUBS
                      ====================
                      OFF         HEAD    2
     NO HEAD OPTIONS
  SUBTRACE            MODE BUFFER HEAD SUBS
 ------------------------------------------------------------
  TCPIPA              ON   0128M
     ASIDS      *NONE*
     JOBNAMES   *NONE*
     OPTIONS    MINIMUM
     WRITER     CTWTR
 ------------------------------------------------------------
  TCPIP               MIN  0016M
     ASIDS      *NONE*
     JOBNAMES   *NONE*
     OPTIONS    MINIMUM
     WRITER     *NONE*
```

- Reproduce the problem.
- Disconnect the external writer.

  ```
  TRACE CT,ON,COMP=SYSTCPOT,SUB=(TCPIPA)
  R xx,WTR=DISCONNECT,END
  ```

- Stop the component trace.

  ```
  TRACE CT,OFF,COMP=SYSTCPOT,SUB=(TCPIPA)
  ```

- Stop the external writer.

  ```
  TRACE CT,WTRSTOP=CTWRT
  ```

Analyzing the trace

There are two ways for formatting the CTRACE: you can use IPCS, or by using a batch job. In this section, we explain how to use each method.

- Using IPCS to format CTRACE

  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.

  From the IPCS PRIMARY OPTION MENU, select: **0 DEFAULTS - Specify default dump and options**; see Example 8-28 for details.

Example 8-28 IPCS default value

```
------------------------ IPCS Default Values --------------------------------
 Command ===>

   You may change any of the defaults listed below. The defaults shown before
   any changes are LOCAL. Change scope to GLOBAL to display global defaults.

   Scope   ==> LOCAL   (LOCAL, GLOBAL, or BOTH)

   If you change the Source default, IPCS will display the current default
   Address Space for the new source and will ignore any data entered in
   the Address Space field.

   Source  ==> DSNAME('SYS1.SC30.CTRACE')
   Address Space   ==>
   Message Routing ==> NOPRINT TERMINAL
   Message Control ==> CONFIRM VERIFY FLAG(WARNING)
   Display Content ==> NOMACHINE REMARK REQUEST NOSTORAGE SYMBOL
```

Modify the DSNAME and OPTIONS to match your environment, then select the following options:

- 2 ANALYSIS - Analyze dump contents
- 7 TRACES - Trace formatting
- 1 CTRACE - Component trace
- D DISPLAY - Specify parameters to display CTRACE entries

Fill in the parameters necessary to format the OSAENTA trace; see Example 8-29 on page 254.

Example 8-29 CTRACE parameters

```
-------------------- CTRACE DISPLAY PARAMETERS  -----------------------
COMMAND ===>

  System      ===>           (System name or blank)
  Component   ===> SYSTCPOT  (Component name (required))
  Subnames    ===> TCPIPA

  GMT/LOCAL   ===> G                       (G or L, GMT is default)
  Start time  ===>                         (mm/dd/yy,hh:mm:ss.dddddd or
  Stop time   ===>                          mm/dd/yy,hh.mm.ss.dddddd)
  Limit       ===> 0         Exception ===>
  Report type ===> SHORT     (SHort, SUmmary, Full, Tally)
  User exit   ===>           (Exit program name)
  Override source ===>
  Options         ===>

  To enter/verify required values, type any character
```

```
  Entry IDs ===>   Jobnames ===>   ASIDs ===>   OPTIONS ===>   SUBS ===>

  CTRACE COMP(SYSTCPOT) SUB((TCPIPA)) SHORT


  ENTER = update CTRACE definition.  END/PF3 = return to previous panel.
  S = start CTRACE.  R = reset all fields.
```

On the Command line, enter the **S** command. Example 8-30 shows the trace formatted by IPCS.

Example 8-30 TRACE format

```
COMPONENT TRACE SHORT FORMAT
 SYSNAME(SC30)
 COMP(SYSTCPOT)SUBNAME((TCPIPA))
 z/OS TCP/IP Packet Trace Formatter, (C) IBM 2000-2006, 2007.052
 DSNAME('SYS1.SC30.CTRACE')

**** 2007/09/11
RcdNr Sysname  Mnemonic Entry Id   Time Stamp    Description
----- -------- -------- -------- --------------- ---------------------------

-------------------------------------------------------------------------------
  365 SC30     OSAENTA  00000007 15:01:23.356987 OSA-Express NTA
 To Interface      : EZANTAOSA2080                              Full=86
  Tod Clock        : 2007/09/11 14:25:44.160269
  Sequence #       : 0              Flags: Pkt Out Nta Vlan Lpar L3
  Source           : 10.1.2.11
  Destination      : 224.0.0.5
  Source Port      : 0              Dest Port: 0     Asid: 0000 TCB: 0000000
 Frame: Device ID  : 02030002       Sequence Nr: 372         Discard: 0 (OK)
  EtherNet II      : 8100           IEEE 802.1 Vlan          Len: 0x0044 (68
  Destination Mac  : 01005E-000005  ()
  Source Mac       : 00096B-1A7490  (IBM)
  Vlan_id          : 10             Priority: 0              Type: 0800 (Int
 IpHeader: Version : 4              Header Length: 20
  Tos              : 00             QOS: Routine Normal Service
  Packet Length    : 68             ID Number: 0AFD
  Fragment         :                Offset: 0
  TTL              : 1              Protocol: OSPFIGP        CheckSum: C253
  Source           : 10.1.2.11
  Destination      : 224.0.0.5

-------------------------------------------------------------------------------
  366 SC30     OSAENTA  00000007 15:01:33.360143 OSA-Express NTA
 To Interface      : EZANTAOSA2080                              Full=86
  Tod Clock        : 2007/09/11 14:25:54.163264
  Sequence #       : 0              Flags: Pkt Out Nta Vlan Lpar L3
  Source           : 10.1.2.11
  Destination      : 224.0.0.5
  Source Port      : 0              Dest Port: 0     Asid: 0000 TCB: 0000000
 Frame: Device ID  : 02030002       Sequence Nr: 373         Discard: 0 (OK)
  EtherNet II      : 8100           IEEE 802.1 Vlan          Len: 0x0044 (68
  Destination Mac  : 01005E-000005  ()
  Source Mac       : 00096B-1A7490  (IBM)
  Vlan_id          : 10             Priority: 0              Type: 0800 (Int
 IpHeader: Version : 4              Header Length: 20
  Tos              : 00             QOS: Routine Normal Service
  Packet Length    : 68             ID Number: 0B07
  Fragment         :                Offset: 0
```

```
  TTL              : 1               Protocol: OSPFIGP        CheckSum: C249
  Source           : 10.1.2.11
  Destination      : 224.0.0.5

-------------------------------------------------------------------------------
```

- Using a batch job to format CTRACE

  We used a batch job to generate the TRACE file, as shown in Example 8-31.

Example 8-31 CTRACE batch job format

```
//PKT2SNIF JOB (999,POK),'CS03',NOTIFY=&SYSUID,
//    CLASS=A,MSGCLASS=T,TIME=1439,
//    REGION=0M,MSGLEVEL=(1,1)
//    SET INDUMP='SYS1.SC30.CTRACE'
//IPCSBTCH EXEC PGM=IKJEFT01,DYNAMNBR=30
//IPCSDDIR DD DISP=SHR,DSN=SYS1.DDIR
//IPCSDUMP DD *
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//INDMP    DD DISP=SHR,DSN=&INDUMP.
//IPCSPRNT DD DSN=ENTA.CTRACE.SHORT,UNIT=SYSDA,
//    DISP=(NEW,CATLG),LRECL=133,SPACE=(CYL,(10,1)),RECFM=VBS,DSORG=PS
//IPCSTOC  DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN  DD *
 PROFILE MSGID
  IPCS NOPARM
  SETD PRINT NOTERM LENGTH(160000) NOCONFIRM FILE(INDMP)
  DROPD
 CTRACE COMP(SYSTCPOT) SUB((TCPIPA)) SHORT
 END
```

We received the output shown in Example 8-32 from the batch job.

Example 8-32 Output from the batch job

```
COMPONENT TRACE SHORT FORMAT
 SYSNAME(SC30)
 COMP(SYSTCPOT)SUBNAME((TCPIPA))
 z/OS TCP/IP Packet Trace Formatter, (C) IBM 2000-2006, 2007.052
 FILE(INDMP)
**** 2007/09/11
RcdNr Sysname  Mnemonic Entry Id   Time Stamp    Description
----- -------- -------- -------- -------------- ------------------------------
-------------------------------------------------------------------------------
  365 SC30     OSAENTA  00000007 15:01:23.356987 OSA-Express NTA
 To Interface      : EZANTAOSA2080                          Full=86
  Tod Clock        : 2007/09/11 14:25:44.160269
  Sequence #       : 0               Flags: Pkt Out Nta Vlan Lpar L3
  Source           : 10.1.2.11
  Destination      : 224.0.0.5
  Source Port      : 0               Dest Port: 0     Asid: 0000 TCB: 00000000
 Frame: Device ID  : 02030002        Sequence Nr: 372         Discard: 0 (OK)
  EtherNet II      : 8100            IEEE 802.1 Vlan          Len: 0x0044 (68)
  Destination Mac  : 01005E-000005   ()
  Source Mac       : 00096B-1A7490   (IBM)
  Vlan_id          : 10              Priority: 0              Type: 0800 (Inter
 IpHeader: Version : 4               Header Length: 20
  Tos              : 00              QOS: Routine Normal Service
```

```
  Packet Length    : 68              ID Number: 0AFD
  Fragment         :                 Offset: 0
  TTL              : 1               Protocol: OSPFIGP       CheckSum: C253 FF
  Source           : 10.1.2.11
  Destination      : 224.0.0.5
-------------------------------------------------------------------------------
  366 SC30     OSAENTA  00000007 15:01:33.360143 OSA-Express NTA
 To Interface      : EZANTAOSA2080                          Full=86
  Tod Clock        : 2007/09/11 14:25:54.163264
  Sequence #       : 0               Flags: Pkt Out Nta Vlan Lpar L3
  Source           : 10.1.2.11
  Destination      : 224.0.0.5
  Source Port      : 0               Dest Port: 0     Asid: 0000 TCB: 00000000
 Frame: Device ID  : 02030002        Sequence Nr: 373         Discard: 0 (OK)
  EtherNet II      : 8100            IEEE 802.1 Vlan          Len: 0x0044 (68)
  Destination Mac  : 01005E-000005   ()
  Source Mac       : 00096B-1A7490   (IBM)
  Vlan_id          : 10              Priority: 0              Type: 0800 (Inter
 IpHeader: Version : 4               Header Length: 20
  Tos              : 00              QOS: Routine Normal Service
  Packet Length    : 68              ID Number: 0B07
  Fragment         :                 Offset: 0
  TTL              : 1               Protocol: OSPFIGP       CheckSum: C249 FF
  Source           : 10.1.2.11
  Destination      : 224.0.0.5
-------------------------------------------------------------------------------
```

8.6 Additional tools for diagnosing CS for z/OS IP problems

IBM and other vendors have developed tools to assist in diagnosing problems in the network from the perspective of z/OS. The tools often run as GUIs on a workstation, but retrieve their problem diagnosis information using data from SNMP, SMF, and from MVS control blocks. Some of these tools also interface with the Network Management Interface API, provided by IBM.

8.6.1 Network Management Interface API (NMI)

Figure 8-21 depicts a high level view of the NMI and its interfaces to network management products.

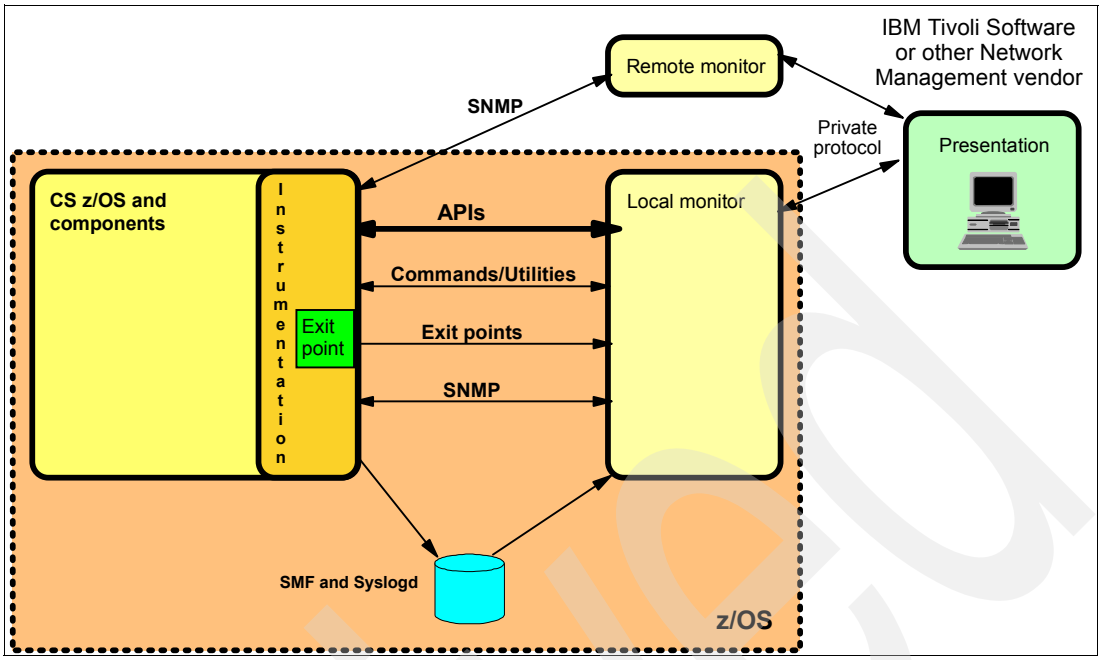


Figure 8-21 Network Management Interface Architecture

The NMI API can interface with Tivoli® OMEGAMON® XE for Mainframe Networks (or other products) to provide the following types of functions:

- Trace assistance:
  - Real-time tracing and formatting for packet and data traces
- Information gathering:
  - TCP connection initiation and termination notifications
  - API for real-time access to TN3270 server and FTP event data and to IPSec
  - APIs to poll information about currently active connections
  - TCP listeners (server processes)
  - TCP connections (detailed information about individual connections and UDP endpoints)
  - CS storage usage
  - API to receive and poll for Enterprise Extender management data
  - Information and statistics for IP filtering and IPSec security associations on the local TCP/IP stacks.
  - Information and statistics for IP filtering and IPSec security associations on remote Network Security Services (NSS) clients when using the NSS server.
- Control activities
- Control the activation and inactivation of IPSec tunnels
- Loading policies for IP filtering and IPSec security associations on the local TCP/IP stacks
  - Loading policies for IP filtering and IPSec security associations on remote Network Security Services (NSS) clients when using the NSS server
  - Drop one or multiple TCP connections or UDP endpoints

8.7 References

Refer to the following content for more information regarding use of logs, standard commands, tools, and utilities:

- *z/OS Communications Server: IP System Administrator's Commands*, SC31-8781
- *z/OS Communications Server: IP Diagnosis Guide*, GC31-8782
- *z/OS Communications Server: IP Configuration Reference*, SC31-8776
- *z/OS MVS Diagnosis: Tools and Service Aids*, GA22-7589
- *z/OS Communications Server: SNA Diagnosis Vol. 1, Techniques and Procedures,* GC31-6850
- *z/OS Communications Server: SNA Operation*, SC31-8779
- *MVS Installation Exits*, SA22-7593
- *Support Element Operations Guide,* SC28-6860

z/OS Communications Server product support information can be obtained at:

`http://www-306.ibm.com/software/network/commserver/zos/support/`

For information on IBM Tivoli OMEGAMON XE for Mainframe Networks, go to:

`http://publib.boulder.ibm.com/tividd/td/IBMTivoliOMEGAMONXEforMainframeNetworks1.0.html`

A

IPv6 support

Over 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 following areas:

- IP address space
- Auto-configuration
- Security
- Quality of service
- Anycast and multicast

This appendix discusses the following topics.

Section	Topic
"Overview of IPv6" on page 262	Basic concepts of IPv6
"Importance of IPv6" on page 262	Key characteristics of IPv6 and why it may be important in your environment
"Common design scenarios for IPv6" on page 262	Commonly implemented IPv6 design scenarios, their dependencies, advantages, considerations, and our recommendations
"How IPv6 is implemented in z/OS Communications Server" on page 265	Selected implementation scenarios, tasks, configuration examples, and problem determination suggestions

Overview of 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 developed to resolve impending problems related to the limitations of IPv4 and the rapidly-growing demand 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, and Internet-connected commodities such as automotive and integrated telephone services. IPv6 uses 128-bit addressing and will generate a space large enough to last for the forseeable future.

Importance of IPv6

IPv6 is important because it addresses the limitations 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.

Common design scenarios for IPv6

Although as explained, there are predictable improvements over IPv4, 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 some time. 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 available today.

Tunneling

Tunneling is the transmission of IPv6 traffic encapsulated within IPv4 packets over an IPv4 connection. Tunnels are used primarily to connect remote IPv6 networks, or to simply connect 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. The *Automatic* tunnel mechanism requires IPv6 prefixes. ISATAP tunnels only require a dual-stack router, but they are 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, so you must give careful consideration to the mechanism you choose. Some are primarily used for stable and secure links for regular communications. Others are primarily used for single hosts or small sites, with low data traffic volumes.

Dedicated data links

Network architects can 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). Note that 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 is needed to resolve IPv6 names and addresses.

Advantages

Use of the 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 there is 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.

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 that the closer the IPv6 implementation is to the client edge, the less expensive it becomes. In contrast, the closer the IPv6 implementation is to the service provider edge, the more expensive it becomes.

Dependencies

Dependencies vary from router configuration to specific hardware requirements to software upgrades, depending on the service provider solution.

Advantages

Use of this strategy requires minor modifications to the infrastructure and minor reconfigurations of the core routers. It is therefore a strategy that could have little or no impact to your environment, involving low costs and low risks.

Considerations

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. IPv6 on service provider edge routers do not support VPNs or VRF, currently.

Dual-stack backbones

A dual-stack backbone is a core network with all routers configured to support dual-stacks. It essentially consists of two network types existing side by side. The IPv4 stack routes the IPv4 traffic through the IPv4 network. 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 a software upgrade to support dual-stack. Having dual-stack requires additional router management of a dual addressing scheme and additional router memory.

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 stacks) 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

A 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).

Recommendation

Using 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.

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 you start to configure the stack to support IPv6 traffic, you need to understand a few things about IPv6.

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 the next list item.

- 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 to use 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
  ```

IPv6 TCP/IP Network part (prefix)

Designers have defined some address types, known as "address scopes", and have left room for future definitions, because unknown requirements may arise. RFC 2373: IP version 6 Addressing Architecture (July 1998) defines the current addressing scheme. Figure A-1 depicts the layout of three types or "scopes" of addresses: the Link-local scope, the Site-local scope, and the Global scope.

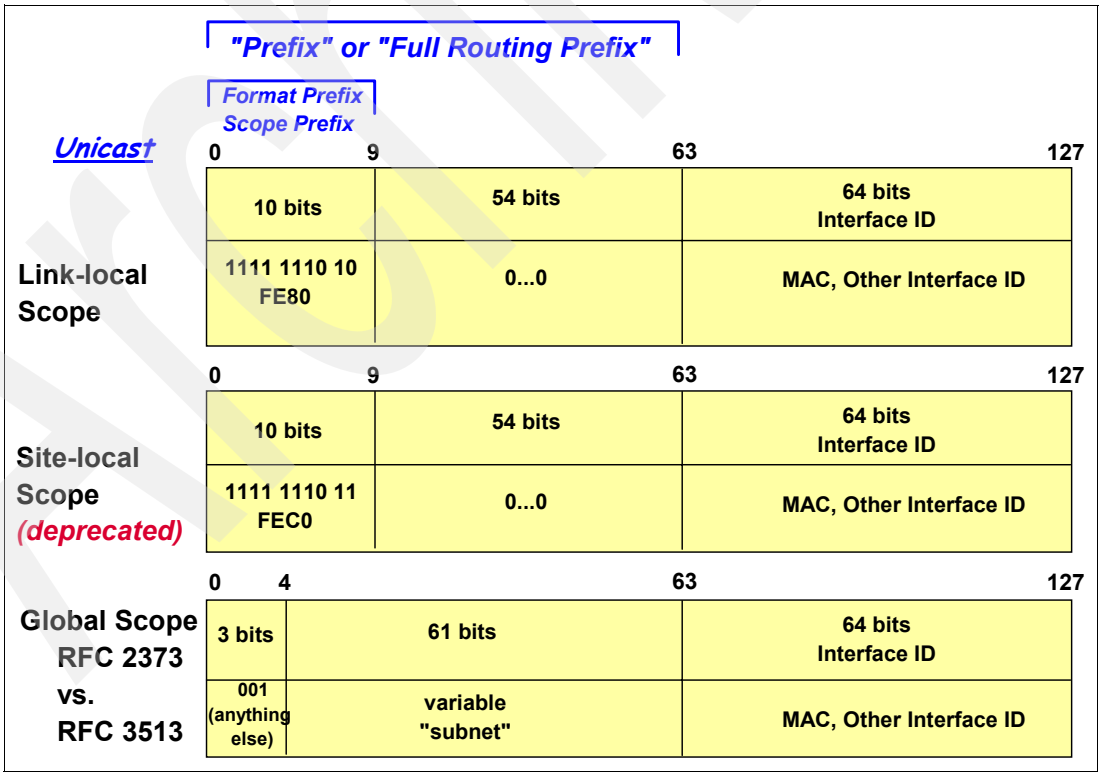


Figure A-1 Unicast IPv6 Addressing Formats

Each address begins with a format or scope prefix of 10 bits, followed by a second field and then an interface identifier field. Each of these addresses serves a unique purpose:

- Link local scope

  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. A packet with a link-local source or destination address will not leave its originating LAN. A router receiving the packet will not forward it onto another physical LAN. An address of this type bears the prefix of fe80.

  A link-local address is assigned to each IPv6-enabled interface after stateless auto-configuration, commonly used in IPv6 implementations. The link-local address is used for link communications such as:

  – Neighbor discovery, that is, discovering whether there is anyone else on this link
  – Communication with a neighbor when a router is unnecessary

  Consider the example shown in Figure A-2.

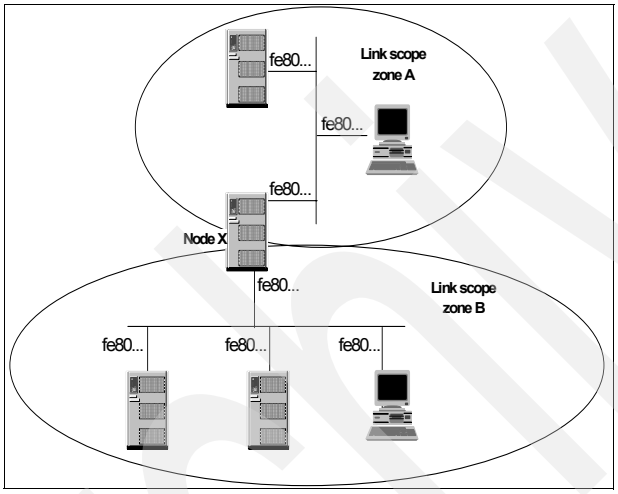


Figure A-2 Link-local addresses and link-local scope zones

Figure A-2 shows a LAN environment separated into two LAN segments, which are represented by Link scope zone A with three nodes and Link scope zone B with four nodes. The link local addresses in each zone begin with the prefix “fe80”.

Within a zone, nodes communicate with each other using link-local addresses. Across zones, nodes must communicate with each other using global scope addresses, which are discussed later.

Note that Node X has link-local addresses in two zones: in zone A and in zone B. Because link-local addresses use the same prefix value, it is necessary to understand which zone a packet should be sent to, particularly when a default route is to be used. So if a route exists on Node X for any destination address with a prefix of “fe80”, then the routing table needs to be able to distinguish between “fe80” in zone A and “fe80” in zone B.

Therefore, both the address and the zone index value need to be specified in the routing table. The *zone index* is a value assigned by the stack to represent the correct entry (or interface) in the routing table. If the zone index is not present, then the stack uses the “default route” for this configuration.

If the default route uses the interface that matches the IPv6 link-local address that was specified, everything works just fine. If, however, the default route does *not* use the correct

interface for the specified IPv6 link-local address, then a routing error is encountered and the application request fails or times out. So the zone index helps the stack to distinguish whether the routing path should flow into zone A or into zone B.

z/OS Communications Server supports scope zone information on Getaddrinfo and Getnameinfo invocations, and also on the z/OS Socket APIs that support IPv6, thus satisfying requirements for IPv6 compliance. In addition, scope zone information can be included on command line operations and in configuration files for FTP, Ping, Traceroute, REXEC, orexec, RSH, and orsh.

- Site local address type

  These addresses are now deprecated, that is, no longer recommended by the IETF. Use and deployment difficulties caused by the use of such addresses has led the IETF to discourage their use.

  Originally, site-local addresses were used to communicate across routers or zones within the same intranet. They were similar to the RFC 1918/Address Allocation for Private Internets in IPv4 today, such as the address ranges represented by10.0.0.0/8, 172.16.0.0/16 - 172.31.0.0/16, and 192.168.0.0/24. Since their deprecation they are treated as global unicast addresses.

  This deprecated address scope 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 by blanking the suffix (least significant 64 bits) completely:

```
3ffe:ffff:100:f101::/64
```

IPv6 implementation in z/OS

The z/OS Communications Server provides support for both IPv4 and IPv6 protocols to coexist. In this book we review how this strategy may be implemented in a z/OS networking environment.

Further details about configuration options not referenced here are available in *z/OS Communications Server: IP Configuration Reference*, SC31-8776, and *z/OS Communications Server: IPv6 Network and Application Design Guide*, SC31-8885.

Table A-1 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 A-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			

z/OS TCP/IP stack function	IPv4 support	IPv6 support	Comments
Sysplex support	Y	Y	IP Sysplex functions: ► Dynamic VIPA ► Sysplex Distributor ► SourceVIPA ► SNMP MIBs for dynamic VIPA ► SysplexPorts Support does not include these functions: ► Sysplex Distributor integration with Cisco MNLB ► Sysplex Wide Security Association (SWSA)
IP routing functions			
Dynamic routing - OSPF and RIP	Y	Y	IPv6-related changes were made to OMPROUTE: ► RIPng support ► OSPFv3 support
Dynamic routing - Auto Configuration	N	Y	
Static route configuration using BEGINROUTES statement	Y	Y	Related configuration statement: BEGINROUTES
Static route configuration using GATEWAY statement	Y	N	Related configuration statement: GATEWAY
Multipath routing groups	Y	Y	Related configuration statements: IPCONFIG IPCONFIG6
Multicast Listener Discovery	N	Y	MLDv2 supported using Source-Filtered multicast (SFM)
Miscellaneous IP/IF-layer 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)
ARP/Neighbor cache PURGE capability	Y	Y	Use the V TCPIP,PURGECACHE command. For information see *z/OS Communications Server: IP System Administrator's Commands*, SC31-8781.
Datagram forwarding enable/disable	Y	Y	Related configuration statements: IPCONFIG IPCONFIG6
Transport-layer functions			
Fast response cache accelerator	Y	N	

z/OS TCP/IP stack function	IPv4 support	IPv6 support	Comments
Enterprise extender	Y	Y	IPv6 Enterprise Extender support requires a virtual IP address and IUTSAMEH. Related configuration statements: INTERFACE VIRTUAL6 and MPCPTP6 IPCONFIG6 DYNAMICXCF
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 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
Network SLAPM2 subagent	Y	Y	
Policy-based networking	Y	Y	IPv6 support in Policy Agent: ► 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 statement: SMFCONFIG
Security function			
IPSec	Y	Y	Related configuration statements: IPCONFIG IPCONFIG6
IP filtering	Y	Y	
NAT traversal	Y	N	
Network access control	Y	N	Related configuration statement: NETACCESS
Stack and port access control	Y	Y	Related configuration statements: PORT DELETE
Application transparent TLS	Y	Y	

z/OS TCP/IP stack function	IPv4 support	IPv6 support	Comments
Intrusion detection services	Y	N	
Server applications			
Rpcbind server	Y	Y	IPv6-enabled RPC applications require a Rpcbind server. The following RPC facilities are not IPv6 enabled, and they do not support RPC binding protocols Version 3 and Version 4: ► rpcgen and orpcgen ► rpcinfo and orpcinfo ► RPC library for the z/OS Communications Server environment ► RPC library for the z/OS UNIX System Services environment For more information see *z/OS Communications Server: IP Configuration Guide*, SC31-8775

Based on the discussion and recommendation in “Common design scenarios for IPv6” on page 262, here we concentrate on a single stack environment running in dual-mode. A single stack environment is one TCP/IP stack running in an LPAR.

Dual-mode stack

As previously discussed, 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, and 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, then 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, then 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 that 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 A-2 summarizes the application communication rules when running in dual-mode.

Table A-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 A-3 on page 274 depicts a dual-mode stack, which is the IPv6 configuration we implemented in our networking environment. The following sections walk you through the setup.

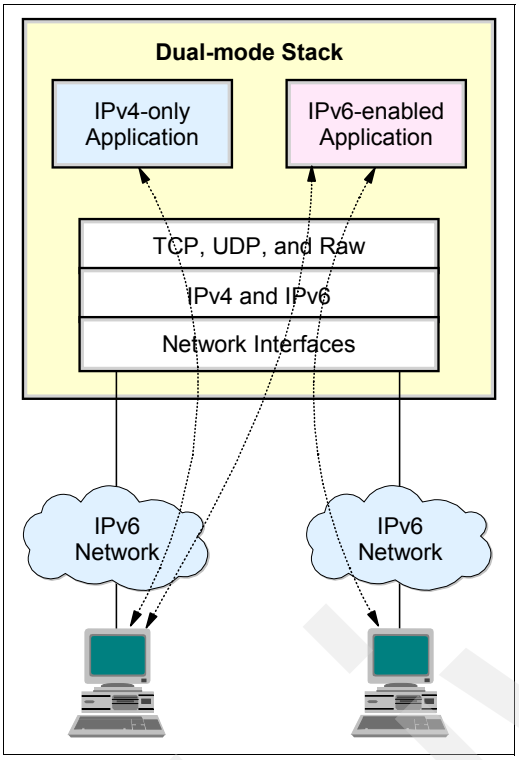


Figure A-3 Dual-mode TCP/IP stack

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 A-1, to our BPXPRMxx member.

Example: A-1 BPXPRMxx NETWORK statement

```
NETWORK DOMAINNAME (AF_INET6)
DOMAINNUMBER(19)
MAXSOCKETS(2000)
TYPE(INET)
```

The TYPE option in our case is INET, because we used 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 Communications Server: IP Configuration Guide*, SC31-8775.

VTAM definitions

As previously mentioned, 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. Because VTAM provides the DLCs for TCP/IP, all TRLEs are defined as VTAM major nodes (see Example A-2).

Example: A-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*, SC31-8776.

We used the following syntax:

```
INTERFace interfname DEFINE linktype PORTNAME portname IPADDR ipaddr
```

The syntax is explained here:

- *interfname* specifies a name for the interface with no more than 16 characters in length.
- *linktype* must be IPAQENET6, which is 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 Example A-3 on page 276. The INTERFACE statement defines the configuration of the OSA-Express device (OSA2080) that we used 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 A-3 on page 276 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: A-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 Mesas
;
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 A-4 shows the DEVICE, LINK, HOME, INTERFACE, and IPADDR definitions we used to support IPv4 and IPv6 and their addressing schemes.

Example: A-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 A-5 show static routes in a flat network (no dynamic routing protocol).

Example: A-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 A-6 shows our VTAM and Telnet (TN3270) definitions.

Example: A-6 Our definitions for TN3270

```
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 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 messages shown in Example A-7 were written to the z/OS console when the TCP/IP stack was initializing.

Example: A-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 LNK6208O       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.

Verification

Next, we verified our environment. Because the TRLE must be active before the interface is started, we ensured that the TRLE is in an active state.

The results are shown in Example A-8. You can also verify the OSA-Express code level with this command.

Example: A-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 A-9 displays the HOME addresses after initialization.

Example: A-9 HOME addresses displayed

```
D TCPIP,TCPIPE,N,HOME
EZD0101I NETSTAT CS V1R8 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:   IQDIOLNK0A0A1482
  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 A-10 shows the output of NETSTAT DEV, with the IPv6 Loopback and device interfaces shown as READY.

Example: A-10 Device display, using Netstat

```
D TCPIP,TCPIPE,N,DEV
EZD0101I NETSTAT CS V1R8 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 Example A-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: A-11 Sockets in the stack

```
D TCPIP,TCPIPE,N,SOCKETS
EZD0101I NETSTAT CS V1R8 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 A-12).

Example: A-12 Results of the ping command

```
TSO PING FEC0:0:0:1001::3302
CS V1R8: Pinging host FEC0:0:0:1001::3302
Ping #1 response took 0.000 seconds.

TSO PING FE80::9:6B00:21A:7490
CS V1R8: Pinging host FE80::9:6B00:21A:7490
Ping #1 response took 0.000 seconds.

TSO PING 10.10.2.232 (TCP tcpipe
CS V1R8: Pinging host 10.10.2.232
Ping #1 response took 0.000 seconds.
```

B

Additional parameters and functions

This appendix contains examples and a discussion of the following topics:

- MVS System symbols
- PROFILE.TCPIP parameters
- TCP/IP built-in security functions

MVS System symbols

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 you need to run many stacks, you 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.

In our case, we used MVS System symbols to enable us to share the definitions for our TCPIP stacks across LPAR SC30, SC31, SC32, and SC33. MVS System symbols 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.

MVS System symbols processing

Use of MVS system symbols in the following files or environment variables is automatically supported:

- PROFILE.TCPIP file
- Resolver SETUP file
- TCPIP.DATA file
- OMPROUTE configuration file
- Resolver environment variables, such as RESOLVER_CONFIG and RESOLVER_TRACE

For the use of MVS system symbols in other configuration files, use the symbol translator utility, EZACFSM1. EZACFSM1 reads an input file which includes the system symbols, and creates an output file with the symbols translated to the system-specific values. This process is done before the files are read by TCP/IP.

The sample JCL for EZACFSM1 utility is included in hlq.SEZAINST(CONVSYM), as shown in 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=OK
//SYSIN   DD DSN=TCP.DATA.INPUT,DISP=SHR
//*SYSIN   DD    PATH='/tmp/tcp.data.input'
//*          The input file can be either an MVS data set or an z/OS
//*           UNIX 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, 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 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 V1R1.0-V1R2.0 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. Our SYSNAME is SC30, so SYSCLONE resolves to 30.

Example: B-3 IEASYSxx definition

```
COUPLE=00,
OMVS=7A,
PROD=01,
PROG=(A0,S0,D0,C1,L0),  Authorization list
SMF=00,
SMS=00,
SYSNAME=SC30, 1
SSN=00,
VAL=00,
```

1 The SYSNAME is defined as SC30 in this LPAR (LPAR A23).

You can also define and use your own variable in configuring CS for z/OS IP, aside from &SYSNAME or &SYSCLONE. Refer to *z/OS Communications Server: IP Configuration Guide*, SC31-8775, for information about creating symbols output data set.

Include files

Together with the MVS System symbols support, we also used a facility (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.

Sample PROFILE.TCPIP definition using MVS System symbols

Example B-4 on page 288 shows the use of MVS system symbols in TCP/IP profile. Because &SYSCLONE is unique in each system, it ensures that the files and IDs that will be generated when the stacks initialize are also unique.

Important: The system symbols are stored in upper case by MVS. Because you can code the TCP/IP configuration statements in either upper case or lower case, you must ensure that you code the system symbol name in upper case.

In our environment, all stacks across LPARs shared the same OSAs and used the same HiperSockets interfaces. We could share the device-related definitions: DEVICE, LINK, BEGINROUTES, and START. We could not share the definitions for HOME and VIPADynamic statements because they are unique in each TCP/IP stack, so we made them into separate members and used the INCLUDE statement. We used the SYSCLONE value to point to those members (the members name must include SYSCLONE).

Example: B-4 Use of system symbols in our TCP/IP profile

```
.....
;
;OSA DEFINITIONS 1
;TRL MAJ NODE: OSA2080,OSA20A0,OSA20C0,and OSA20E0
DEVICE OSA2080    MPCIPA
LINK   OSA2080L   IPAQENET    OSA2080 VLANID 10
DEVICE OSA20C0    MPCIPA
LINK   OSA20C0L   IPAQENET    OSA20C0 VLANID 11
DEVICE OSA20E0    MPCIPA
LINK   OSA20E0L   IPAQENET    OSA20E0
DEVICE OSA20A0    MPCIPA
LINK   OSA20A0L   IPAQENET    OSA20A0
;
;HIPERSOCKETS DEFINITIONS
DEVICE IUTIQDF4   MPCIPA
LINK   IUTIQDF4L  IPAQIDIO    IUTIQDF4
DEVICE IUTIQDF5   MPCIPA
LINK   IUTIQDF5L  IPAQIDIO    IUTIQDF5
DEVICE IUTIQDF6   MPCIPA
LINK   IUTIQDF6L  IPAQIDIO    IUTIQDF6
;
;STATIC VIPA DEFINITIONS
DEVICE VIPA1      VIRTUAL 0
LINK   VIPA1L     VIRTUAL 0   VIPA1
DEVICE VIPA2      VIRTUAL 0
LINK   VIPA2L     VIRTUAL 0   VIPA2
DEVICE VIPA3      VIRTUAL 0
LINK   VIPA3L     VIRTUAL 0   VIPA3
;
;*****************************************************************
; Include the stack-specific Dynamic VIPA definitions
;*****************************************************************
INCLUDE TCPIPA.TCPPARMS(DVIPA&SYSCLONE) 2
;
;*****************************************************************
; Include the stack-specific HOME definitions
;*****************************************************************
INCLUDE TCPIPA.TCPPARMS(HOME&SYSCLONE) 2
;
BEGINRoutes
;     Destination     Subnet Mask   First Hop Link Name     Packet Size ;
 ROUTE 10.1.2.0/24                    =        OSA2080L      MTU 1492
 ROUTE 10.1.2.0/24                    =        OSA20A0L      MTU 1492
 ROUTE 10.1.3.0/24                    =        OSA20C0L      MTU 1492
 ROUTE 10.1.3.0/24                    =        OSA20E0L      MTU 1492
 ROUTE 10.1.4.0/24                    =        IUTIQDF4L     MTU 8192
 ROUTE 10.1.5.0/24                    =        IUTIQDF5L     MTU 8192
 ROUTE 10.1.6.0/24                    =        IUTIQDF6L     MTU 8192
 ROUTE 10.1.100.0/24               10.1.2.240   OSA2080L    MTU 1492
 ROUTE DEFAULT                     10.1.2.240   OSA2080L    MTU 1492
 ROUTE DEFAULT                     10.1.2.220   OSA2080L    MTU 1492
```

```
 ROUTE DEFAULT                     10.1.3.240   OSA20C0L    MTU 1492
 ROUTE DEFAULT                     10.1.3.220   OSA20C0L    MTU 1492
ENDRoutes
;*************************************************************
; start the ftp daemon in each of the A stack
;*************************************************************
AUTOLOG 5
   FTPD&SYSCLONE JOBNAME FTPD&SYSCLONE.1 3
;  OMP&SYSCLONE             ; OSPF daemon
;  SMTP                     ; SMTP Server
ENDAUTOLOG
;*************************************************************
; Include the stack-specific PORT definitions
;*************************************************************
INCLUDE TCPIPA.TCPPARMS(PORT&SYSCLONE) 2
....
START OSA2080
START OSA20C0
START OSA20E0
START OSA20A0
START IUTIQDF4
START IUTIQDF5
START IUTIQDF6
```

1 sets a unique IP address for the dynamic XCF definitions.

2 Include file for system-specific device definitions.

3 Defines the AUTOLOG with unique FTP server daemon Jobname.

Note: A dot (.) is needed at the end of &SYSCLONE because the next character is not a space.

Example B-5 shows the sample definition of a separate member for a stack-specific statement. It contains only the HOME statement for system SC30, called HOME30. This member is included in the PROFILE.TCPIP file in SC30 system. Likewise, define separate members for other LPARs.

Example: B-5 Included device file HOME30 for SC30

```
;*************************************************************
; TCPIPA.TCPPARMS(HOME30)
; HOME definitions for stack on SC30 image
;*************************************************************
HOME
   10.1.1.10     VIPA1L
   10.1.2.11     OSA2080L
   10.1.3.11     OSA20C0L
   10.1.3.12     OSA20E0L
   10.1.2.12     OSA20A0L
   10.1.4.11     IUTIQDF4L
   10.1.5.11     IUTIQDF5L
   10.1.6.11     IUTIQDF6L
   10.1.2.10     VIPA2L
   10.1.&SYSCLONE..10  VIPA3L
```

PROFILE.TCPIP statements

In this section we show PROFILE.TCPIP statements that are not always necessary, but are important. For detailed descriptions of statement, refer to *z/OS Communications Server: IP Configuration Guide*, SC31-8775. The syntax for the statement in the PROFILE can be found in *z/OS Communications Server: IP Configuration Reference*, SC31-8776.

IPCONFIG statements

This section provides information about IPCONFIG statements.

SOURCEVIPA

When the packet is sent to the destination host, the source IP address is included in the packet. In most cases the source IP address of the packet is used as the destination IP address of the returning packet from the other host. For the inbound traffic, z/OS Communications Server sets the destination IP address of the incoming packet to the source IP address of the return packet. However, for outbound traffic, the source IP address is determined by several parameters.

By default (IPCONFIG NOSOURCEVIPA), z/OS Communications Server sets the IP address of the interface which is used to send out a packet to a specific destination as the source IP address. The sending interface is selected depending on the routing table of the TCP/IP stack.

When IPCONFIG SOURCEVIPA is set, outbound datagrams use the virtual IP address (VIPA) for the source IP address of the packet instead of the physical interface IP address. By using VIPA as the source IP address, and therefore the destination IP address of the return packets from other hosts, SOURCEVIPA provides the tolerance of device and adapter failures.

The order of the HOME list is important if SOURCEVIPA is specified. The source IP address is the first static VIPA listed above the interface chosen for sending the packet. In Example B-6, if OSA20C0 **2** is chosen as the actual physical interface for sending the outbound packet, then the IP address of the first VIPA above the HOME list, 10.1.2.10, is the source IP address.

Example: B-6 Source IP Address selection with IPCONFIG SOURCEVIPA

```
....
IPCONFIG SOURCEVIPA

HOME
   10.1.1.10     VIPA1L
   10.1.2.10     VIPA2L 1
   10.1.2.11     OSA2080L
   10.1.3.11     OSA20C0L 2
....
```

Note: The source IP address selection can be overridden with SRCIP statement. Refer to “SRCIP” on page 299 for details.

SOURCEVIPA has no effect on OSPF or RIP route information exchange packets generated by the OMPROUTE routing daemon, which means that it is only applicable for data diagrams.

MULTIPATH

With the IPCONFIG MULTIPATH statement, packets can 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). 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-connection or per-packet basis, but we recommend that you do not use the per-packet basis because it requires high CPU processing for reassembly of out-of-order packets at the receiver. See Chapter 5, “Routing” on page 141 for details about this topic.

By default (IPCONFIG NOMULTIPATH), there is no multipath support and all connections use the first active route to the destination network or host even if there are other, equal-cost routes available.

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.

The default is IPCONFIG NOPATHMTUDISCOVERY. Aside from enabling PMTU during stack initialization, you could also enable or disable PMTU discovery by using VARY OBEYFILE.

IQDIOROUTING

When IPCONFIG IQDIOROUTING is configured, the inbound packets that are to be forwarded by this TCP/IP stack use HiperSockets (also known as Internal Queued Direct I/O or iQDIO) and Queued Direct I/O (QDIO) directly and bypass the TCP/IP stack. 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. The default is NOIQDIOROUTING. For further information about HiperSockets, see Chapter 4, “Connectivity” on page 101.

ARPTO

IPCONFIG ARPTO and ARPAGE statements have the same function: they specify the time interval between the 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 or ARPAGE because they use the ARP offload function. The ARP cache timer for ARP offload is set to 20 minutes. It is hard-oded and not configurable. For more information about devices that are affected by ARPTO, refer to *z/OS Communications Server: IP Configuration Guide*, SC31-8775.

The UNIX shell command **`onetstat -R`** displays the current ARP cache entries. The upper case 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 upper case letters). If you do not specify in upper case letters, the reserved word is not recognized (see Example B-7 on page 292).

Example: B-7 ARP display

```
D TCPIP,TCPIPA,N,ARP
EZD0101I NETSTAT CS V1R9 TCPIPA 262
QUERYING ARP CACHE FOR ADDRESS 10.1.3.11
LINK: OSA20C0L          ETHERNET: 00096B1A7454
QUERYING ARP CACHE FOR ADDRESS 10.1.3.21
LINK: OSA20C0L          ETHERNET: 00096B1A7454
QUERYING ARP CACHE FOR ADDRESS 10.1.3.22
LINK: OSA20C0L          ETHERNET: 00096B1A7454
QUERYING ARP CACHE FOR ADDRESS 10.1.3.220
LINK: OSA20C0L          ETHERNET: 000785F37302
QUERYING ARP CACHE FOR ADDRESS 10.1.3.240
LINK: OSA20C0L          ETHERNET: 0014F1464600
QUERYING ARP CACHE FOR ADDRESS 10.1.2.12
LINK: OSA2080L          ETHERNET: 00096B1A74C2
QUERYING ARP CACHE FOR ADDRESS 10.1.2.22
LINK: OSA2080L          ETHERNET: 00096B1A74C2
QUERYING ARP CACHE FOR ADDRESS 10.1.2.21
LINK: OSA2080L          ETHERNET: 00096B1A74C2
QUERYING ARP CACHE FOR ADDRESS 10.1.2.240
LINK: OSA2080L          ETHERNET: 0014F1464600
QUERYING ARP CACHE FOR ADDRESS 10.1.2.220
LINK: OSA2080L          ETHERNET: 000785F37302
QUERYING ARP CACHE FOR ADDRESS 10.1.4.21
LINK: IUTIQDF4L
QUERYING ARP CACHE FOR ADDRESS 10.1.4.11
LINK: IUTIQDF4L
QUERYING ARP CACHE FOR ADDRESS 10.1.5.21
LINK: IUTIQDF5L
QUERYING ARP CACHE FOR ADDRESS 10.1.5.11
LINK: IUTIQDF5L
QUERYING ARP CACHE FOR ADDRESS 10.1.6.21
LINK: IUTIQDF6L
QUERYING ARP CACHE FOR ADDRESS 10.1.6.11
LINK: IUTIQDF6L
16 OF 16 RECORDS DISPLAYED
END OF THE REPORT
```

GLOBALCONFIG statements

This section provides information about GLOBALCONFIG statements.

TCPIPSTATISTICS

This statement 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.

SEGMENTATIONOFFLOAD

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.

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 or later release 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 supported by OSA-Express in QDIO mode.

Example B-8 displays the Netstat Devlinks of an OSA-Express that has SegmentationOffload enabled.

Example: B-8 Segmentation Offload enabled

```
D TCPIP,TCPIPA,N,DEV
EZD0101I NETSTAT CS V1R9 TCPIPA 548
DEVNAME: OSA2080           DEVTYPE: MPCIPA
  DEVSTATUS: READY
  LNKNAME: OSA2080L          LNKTYPE: IPAQENET   LNKSTATUS: READY
    NETNUM: N/A  QUESIZE: N/A  SPEED: 0000001000
    IPBROADCASTCAPABILITY: NO
    CFGROUTER: NON                   ACTROUTER: NON
    ARPOFFLOAD: YES 1                ARPOFFLOADINFO: YES 1
    ACTMTU: 8992
    VLANID: 10                       VLANPRIORITY: DISABLED
    DYNVLANREGCFG: NO                DYNVLANREGCAP: YES
    READSTORAGE: GLOBAL (4096K)      INBPERF: BALANCED
    CHECKSUMOFFLOAD: YES 1           SEGMENTATIONOFFLOAD: YES 1
    SECCLASS: 255                    MONSYSPLEX: NO
.....
```

1 Indicates the enabled features of ARP, Segmentation, and Checksum Offload.

PORT statement

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

The BIND option 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 client that wants to access both Telnet servers, TN3270 and UNIX Telnet, would connect to different ports or different TCP/IP stacks, depending on which Telnet server it wanted 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 TN3270 and UNIX Telnet. All that is needed is to code the BIND keyword in the PORT statement for each server:

```
PORT
    23 TCP  TN3270A BIND 10.1.1.10
    23 TCP  OMVS    BIND 10.1.1.20
```

In this case, the TN3270A is a jobname for TN3270 server. When it BINDs to port 23 and INADDR_ANY, it is associated with IP address 10.1.1.10. The OMVS job name identifies any UNIX server, including the UNIX Telnet server. When UNIX Telnet Server BINDs to port 23 and INADDR_ANY, it is associated with IP address 10.1.1.20.

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 it is not valid with the PORTRANGE statement.

TCPCONFIG/UDPCONFIG statements

This section provides information about TCPCONFIG/UDPCONFIG statements.

TCPCONFIG/UDPCONFIG RESTRICTLOWPORTS

The PORT reservations that are defined in the PROFILE data set are the ports that are used by specific applications.

You may decide not to explicitly 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-9 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 FTP server); see Example B-10. You would need to reserve the port using the new address space name.

Example: B-10 PROFILE.TCPIP: PORT & PORTRANGE

```
TCPCONFIG
   RESTRICTLOWPORTS

UDPCONFIG
   RESTRICTLOWPORTS

PORT
    20 TCP OMVS      NOAUTOLOG ; FTP Server  1
    21 TCP FTPDA1  BIND 10.1.1.10 ; FTP Server  2
    23 TCP TN3270A            ; Telnet Server
    25 TCP SMTP               ; SMTP Server
    514 UDP OMVS              ; UNIX SyslogD Server   3
    520 UDP OMPA NOAUTOLOG    ; OMPROUTE RIP IPv4
    521 UDP OMPA NOAUTOLOG    ; OMPROUTE RIP IPv6
PORTRANGE 10000 2000 TCP OMVS    ; TCP 10000 - 11999    4
PORTRANGE 10000 2000 UDP OMVS    ; UDP 10000 - 11999    4
```

Normally you can specify either OMVS or the job name in the PORT statement. However, certain daemons have special considerations on this matter.

When the FTP server starts, it forks the listener process to run in the background, requiring that the name of the forked address space (FTPDA1, in this example), not the original procedure name, be used on the PORT statement of the control connection 2. You must specify OMVS as the name on the PORT for FTP's PORT 20 1, which is used for the data connection managed by the child process. If you specify the forked name on the data connection (Port 20), the data connections will fail.

Note that we also reserve UDP port 514 3 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.

PORTRANGE statements 4 reserve a range of ephemeral TCP and UDP ports for UNIX System Services.

In Example B-10, ports 10000 to 11999 are reserved. The range must match the INADDRANYPORT and INADDRANYCOUNT in your BPXPRMxx member 5 (see Example B-11 on page 296).

Example: B-11 INADDRANYPORT and INADDRANYCOUNT in BPXPRMxx member

```
NETWORK DOMAINNAME(AF_INET)
        DOMAINNUMBER(2)
        MAXSOCKETS(10000)
        TYPE(INET)
        INADDRANYPORT(10000)   5
        INADDRANYCOUNT(2000)   5

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 *procname* -o**.

Example: B-12 Viewing port reservation list

```
D TCPIP,TCPIPA,N,PORTL
EZD0101I NETSTAT CS V1R9 TCPIPA 420
PORT# PROT USER     FLAGS    RANGE       SAF NAME
00020 TCP  OMVS     D
00021 TCP  FTPDA1   DABU
      BINDSPECIFIC: 10.1.1.10
00023 TCP  TN3270A  DA
00025 TCP  SMTP     DA
00500 UDP  IKED     DA
00514 UDP  OMVS     DA
00520 UDP  OMPROUTE D
00521 UDP  OMPROUTE D
8 OF 8 RECORDS DISPLAYED
```

TCPCONFIG TCPSENDBFRSIZE

TCPCONFIG TCPSENDBFRSIZE specifies the TCP send buffer size. This value is used as the default send buffer size for those applications that do not explicitly set the buffer size using SETSOCKOPT(). The default is 16384 (16K).

TCPCONFIG TCPRCVBUFRSIZE

TCPCONFIG TCPRCVBUFRSIZE specifies the TCP receive buffer size. This value is used as the default receive buffer size for those applications that do not explicitly set the buffer size using SETSOCKOPT(). You can specify value from 256 and TCPMAXRCVBUFRSIZE. The default is 16384 (16K).

TCPCONFIG TCPMAXRCVBUFRSIZE

TCPCONFIG TCPMAXRCVBUFRSIZE specifies the TCP maximum receive buffer size an application can set as its receive buffer size using SETSOCKOPT(). You can use this parameter to limit the receive buffer size that an application can set. The minimum value you can specify is TCPRCVBUFRSIZE, and the maximum is 512K. The default is 256K.

Note: The FTP server and client applications override the default settings and use 64KB as the TCP window size and 180KB for send/recv buffers. No changes are required in the TCPCONFIG statement for the FTP server and client.

TCPCONFIG FINWAIT2TIME

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 the 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.

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 V1R9 TCP/IP Implementation Volume 3: High Availability, Scalability, and Performance*, SG24-7534, for further details.

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 V1R9 TCP/IP Implementation Volume 2: Standard Applications*, SG24-7533, for further information.

```
SACONFig
  COMMUNity public   ; Community string
  OSASF 760          ;OSASF port number
; AGENT 161          ;Agent port number
  ENABLed
  SETSENAbled
  ATMENabled
```

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, and so on), 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 preceding example can also be coded this as shown here, because 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 V1R9 TCP/IP Implementation Volume 2: Standard Applications*, SG24-7533.)

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 record layouts and standardized subtype numbers, refer to *z/OS Communications Server: IP Configuration Reference*, SC31-8776.

DEVICE AND LINK statements

Device and Link statements now support features to support VLAN IDs and OFFLOAD processing to the OSA-Express adapter.

VLAN ID

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. With VLANID specified, the TCP/IP stacks that share an OSA can have an IP address assigned from different IP subnets.

It is configured and implemented in the z/OS environment through the LINK definitions in the PROFILE.TCPIP for OSA-Express in QDIO mode. VLANs support ARP takeover in a *flat network* (no routing protocol) when connected appropriately. Refer to Chapter 4,

"Connectivity" on page 101 for more information about this implementation. Following is a link definition example of OSA2080LNK attached to virtual LAN 10:

```
DEVICE OSA2080    MPCIPA
LINK   OSA2080L   IPAQENET    OSA2080 VLANID 10
```

Example B-13 displays the Netstat Devlinks display of an OSA-Express that has VLAN ID enabled.

Example: B-13 VLAN ID enabled

```
D TCPIP,TCPIPA,N,DEV
EZD0101I NETSTAT CS V1R9 TCPIPA 548
DEVNAME: OSA2080           DEVTYPE: MPCIPA
  DEVSTATUS: READY
  LNKNAME: OSA2080L          LNKTYPE: IPAQENET   LNKSTATUS: READY
    NETNUM: N/A  QUESIZE: N/A  SPEED: 0000001000
    IPBROADCASTCAPABILITY: NO
    CFGROUTER: NON                     ACTROUTER: NON
    ARPOFFLOAD: YES                    ARPOFFLOADINFO: YES
    ACTMTU: 8992
    VLANID: 10 1                       VLANPRIORITY: DISABLED
.....
```

1 VLAN tagging is enabled on this device (VLAN 10).

SRCIP

For inbound packets, the source IP address of a returning packet is always the destination IP address of a receiving packet. For outbound packets, the default source IP address is the HOME IP address of the interface chosen for sending the packet according to the routing table. If you specify IPCONFIG SOURCEVIPA, the source IP address is the first static VIPA listed above the interface chosen for sending the packet.

z/OS V1R6 or a later release provides an additional option for specifying the source IP address. We can designate source IP addresses to be used for outbound TCP connections initiated by specified jobs or destined for specified IP addresses, networks, or subnets, by using the SRCIP statement, as described here:

- *Job-specific* source IP addressing (with z/S V1R6 or later releases) by using the JOBNAME option in the SRCIP statement
- *Destination-specific* source IP addressing (with z/OS V1R8 or later releases) by using the DESTINATION option in the SRCIP statement

These source IP address definitions override any other source IP address specification in TCP/IP profile. However, the use of SRCIP can also be overridden directly by an application through the use of specific socket API options.

z/OS V1R8 Communications Server has a restriction that a distributed DVIPA may not be specified as the source IP address on the DESTINATION statement. The TCP/IP client application issues a connect() socket call to start a TCP/IP connection, and optionally issues a bind() socket call prior to connect(). A problem occurs when a client application issues an explicit bind() socket call with INADDR_ANY and port 0 to have a port assigned prior to connect(). Until a connect() socket call that includes the destination IP address is issued, z/OS Communications Server is unable to determine the source IP address and therefore fails to choose which sysplex port pool the port should be assigned from.

z/OS V1R9 Communications Server relieves this restriction by adding a new option EXPLICITBINDPORTRANGE. Unlike the sysplexport pools for which each pool is associated with a specific distributed DVIPA, the port range specified by EXPLICITBINDPORTRANGE is not associated with any specific distributed DVIPA, and can be used for any distributed DVIPA.

The EZBEPORT*vvtt* structure in the Coupling Facility, where *vv* is the 2-character VTAM group ID suffix specified on the XCFGRPID start option and *tt* is the TCP group ID suffix specified on the GLOBALCONFIG statement in the TCP/IP profile, coordinates this port range among all members of the sysplex. The port range should be identical in all members of the sysplex.

Note: The use of EXPLICTBINDPORTRANGE has a restriction in CINET environment. It is only available when the application has an affinity to a specific TCPIP stack, or when only one stack is managed by CINET.

Example B-14 shows a sample definition of the SRPIP statement.

Example: B-14 SRCIP definition

```
GLOBALCONFIG EXPLICITBINDPORTRANGE 7000 1024 3

SRCIP
   JOBNAME     *             10.1.1.10 CLIENT 1
   JOBNAME     CUST*         10.1.2.10 SERVER 1
   DESTINATION 10.1.2.240    10.1.1.10 2
   DESTINATION 10.1.2.0/24   10.1.2.10 2
   DESTINATION 10.1.100.0/24 10.1.8.10 3
ENDSRCIP
```

1 This is a sample definition of a job-specific source IP address. JOBNAME option is enhanced in z/OS V1R9 Communications Server to support the use of JOBNAME for listening server applications with the SERVER option. The CLIENT option indicates the existing support for client applications, and this is the default.

2 This is a sample definition of a destination-specific source IP address feature. The most specific match is applied.

3 This example uses a distributed DVIPA for the source IP address on the DESTINATION option. Define GLOBALCONFIG EXPLICITBINDPORTRANGE to reserve 1024 ports starting from 7000 for any distributed DVIPAs that are to be the source IP addresses. Ensure that the ports specified for EXPLICITBINDPORTRANGE are same among all sysplex members and do not overlap with any other port reservations: PORT, PORTRANGE, SYSPLEXPORTS, or BPXPRMxx INADDRANYPORT.

We used the NETSTAT,SRCIP command to verify our configuration, as shown in Example B-15.

Example: B-15 NETSTAT SRCIP display

```
D TCPIP,TCPIPA,N,SRCIP
EZD0101I NETSTAT CS V1R9 TCPIPA 850
SOURCE IP ADDRESS BASED ON JOB NAME:
JOB NAME  TYPE  FLG  SOURCE
--------  ----  ---  ------
*         IPV4  C    10.1.1.10
CUST*     IPV4  C    10.1.2.10
```

```
SOURCE IP ADDRESS BASED ON DESTINATION:
DESTINATION: 10.1.100.0/24
  SOURCE:    10.1.8.10
DESTINATION: 10.1.2.240
  SOURCE:    10.1.1.10
DESTINATION: 10.1.2.0/24
  SOURCE:    10.1.2.10
5 OF 5 RECORDS DISPLAYED
END OF THE REPORT
```

To verify the destination-specific source IP address feature functions correctly, we issued the TSO Telnet command with an IP address configured in an L3 Switch. Example B-16 shows results of the **show tcp brief** command issued for the L3 Switch.

Example: B-16 show tcp brief commands

```
telnet 10.1.2.240
Router1#sh tcp bri
TCB       Local Address          Foreign Address        (state)
46303D58  10.1.2.240.23          10.1.1.10.1036         ESTAB

telnet 10.1.2.220
Router2#sh tcp bri
TCB       Local Address          Foreign Address        (state)
423B1414  10.1.2.220.23          10.1.2.10.1037         ESTAB
```

We see a different source IP address is used for each specific destination IP address.

TCP/IP built-in security functions

TCP/IP for MVS has built-in security functions that can be activated and used to control specific areas:

- 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 control over the functions performed by the FTP server. Using these exits you can 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, consider implementing SNMPv3, whose messages have data integrity and data origin authentication.

Both the IMS sockets and CICS sockets support provide 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 your decision.

C

Examples used in our environment

This appendix provides the examples used in the configuration of our environment.

It provides the following:

- Resolver
- TCP/IP stack
- OMPROUTE dynamic routing

Resolver

Resolver cataloged procedure

Example: C-1 The Resolver cataloged procedure

```
/*****************************************
/* SYS1.PROCLIB(RESOLV30)
/*****************************************
//RESOLV30 PROC PARMS='CTRACE(CTIRES00)'
//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=TCPIPA.TCPPARMS(RESOLV&SYSCLONE),DISP=SHR,FREE=CLOSE
```

BPXPRMxx

Example: C-2 Specifying the Resolver procedure to be started

```
/*****************************************
/* SYS1.PARMLIB(BPXPRM00)
/*****************************************
/* 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(RESOLV30)
```

Resolver SETUP data set

Example: C-3 Resolver address space SETUP data set

```
; *************************************
; TCPIPA.TCPPARMS(RESOLV30)
; *************************************
GLOBALTCPIPDATA('TCPIPA.TCPPARMS(GLOBAL)')
DEFAULTTCPIPDATA('TCPIPA.TCPPARMS(DEFAULT)')
GLOBALIPNODES('TCPIPA.TCPPARMS(IPNODES)')
COMMONSEARCH
```

Global TCPIP.DATA file

Example: C-4 Global TCPIP.DATA file

```
; ****************************************
; TCPIPA.TCPPARMS(GLOBAL)
; ****************************************
DOMAINORIGIN  ITSO.IBM.COM
NSINTERADDR  10.12.6.7
NSPORTADDR 53
RESOLVEVIA UDP
RESOLVERTIMEOUT 10
RESOLVERUDPRETRIES 1
LOOKUP LOCAL DNS
```

Default TCPIP.DATA file

Example: C-5 Default TCPIP.DATA file

```
; ****************************************
; TCPIPA.TCPPARMS(DEFAULT)
; ****************************************
TCPIPJOBNAME TCPIP
HOSTNAME WTSC30
```

Global ETC.IPNODES data set

Example: C-6 GLOBALIPNODES data set

```
; ****************************************
; TCPIPA.TCPPARMS(IPNODES)
; ****************************************
   10.12.6.7   OURDNS
   10.1.1.10   WTSC30A
   10.1.1.20   WTSC31B
   10.1.1.30   WTSC32C
   10.1.2.240  router1
   10.1.2.220 router2
   1::2 TESTIPV6ADDRESS1
   1:2:3:4:5:6:7:8 TESTIPV6ADDRESS2
```

TCP/IP stack

TCPIPA stack started procedure

Example: C-7 TCPIPA procedure

```
/****************************************
/* SYS1.PROCLIB(TCPIPA)
/****************************************
//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.)
```

TCPIP.DATA file for TCPIPA stack

Example: C-8 TCPIP.DATA file DATAA30

```
; ****************************************
; TCPIPA.TCPPARMS(DATAA30)
; ****************************************
TCPIPJOBNAME TCPIPA
HOSTNAME WTSC30A
DATASETPREFIX TCPIPA
MESSAGECASE MIXED
```

PROFILE.TCPIP (for static routing)

Example: C-9 PROFILE.TCPIP (for static routing)

```
;  This profile is for static routing
ARPAGE 20
;
GLOBALCONFIG NOTCPIPSTATISTICS
;
IPCONFIG DATAGRAMFWD SYSPLEXROUTING
DYNAMICXCF 10.1.7.11 255.255.255.0 1
;
SOMAXCONN 10
;
TCPCONFIG TCPSENDBFRSIZE 64K TCPRCVBUFRSIZE 64K SENDGARBAGE FALSE
TCPCONFIG TCPMAXRCVBUFRSIZE 256K
TCPCONFIG UNRESTRICTLOWPORTS
UDPCONFIG UNRESTRICTLOWPORTS
;
;OSA definitions
;TRL MAJ NODE: OSA2080,OSA20A0,OSA20C0,and OSA20E0
DEVICE OSA2080    MPCIPA
LINK   OSA2080L   IPAQENET    OSA2080 VLANID 10
DEVICE OSA20C0    MPCIPA
LINK   OSA20C0L   IPAQENET    OSA20C0 VLANID 11
DEVICE OSA20E0    MPCIPA
LINK   OSA20E0L   IPAQENET    OSA20E0 VLANID 11
DEVICE OSA20A0    MPCIPA
LINK   OSA20A0L   IPAQENET    OSA20A0 VLANID 10
;
;HiperSockets definitions
DEVICE IUTIQDF4   MPCIPA
LINK   IUTIQDF4L  IPAQIDIO    IUTIQDF4
DEVICE IUTIQDF5   MPCIPA
LINK   IUTIQDF5L  IPAQIDIO    IUTIQDF5
DEVICE IUTIQDF6   MPCIPA
LINK   IUTIQDF6L  IPAQIDIO    IUTIQDF6
;
;Static VIPA definitions
DEVICE VIPA1      VIRTUAL 0
LINK   VIPA1L     VIRTUAL 0   VIPA1
DEVICE VIPA2      VIRTUAL 0
LINK   VIPA2L     VIRTUAL 0   VIPA2
DEVICE VIPA3      VIRTUAL 0
LINK   VIPA3L     VIRTUAL 0   VIPA3
;
```

```
;Dynamic VIPA definitions
VIPADYNAMIC
 VIPADEFINE MOVEABLE IMMEDIATE 255.255.255.0 10.1.8.10
 VIPABACKUP 3 MOVEABLE IMMEDIATE 255.255.255.0 10.1.8.20
 VIPABACKUP 3 MOVEABLE IMMEDIATE 255.255.255.0 10.1.8.30
 VIPABACKUP 3 MOVEABLE IMMEDIATE 255.255.255.0 10.1.8.40
 VIPARANGE 255.255.255.0 10.1.9.0
ENDVIPADYNAMIC
;
HOME
   10.1.&SYSCLONE..10  VIPA3L
   10.1.1.10     VIPA1L
   10.1.2.10     VIPA2L
   10.1.2.11     OSA2080L
   10.1.3.11     OSA20C0L
   10.1.3.12     OSA20E0L
   10.1.2.12     OSA20A0L
   10.1.4.11     IUTIQDF4L
   10.1.5.11     IUTIQDF5L
   10.1.6.11     IUTIQDF6L
;
 PRIMARYINTERFACE VIPA1L
;
BEGINRoutes
; Direct Routes - Routes that are directly connected to my interfaces
;     Destination     Subnet Mask   First Hop Link Name     Packet Size ;
 ROUTE 10.1.2.0/24                    =       OSA2080L      MTU 1492
 ROUTE 10.1.2.0/24                    =       OSA20A0L      MTU 1492
 ROUTE 10.1.3.0/24                    =       OSA20C0L      MTU 1492
 ROUTE 10.1.3.0/24                    =       OSA20E0L      MTU 1492
 ROUTE 10.1.4.0/24                    =       IUTIQDF4L     MTU 8192
 ROUTE 10.1.5.0/24                    =       IUTIQDF5L     MTU 8192
 ROUTE 10.1.6.0/24                    =       IUTIQDF6L     MTU 8192
 ROUTE 10.1.1.20/32               10.1.4.21  IUTIQDF4L     MTU 8192
 ROUTE 10.1.2.20/32               10.1.4.21  IUTIQDF4L     MTU 8192
 ROUTE 10.1.31.10/32              10.1.4.21  IUTIQDF4L     MTU 8192
 ROUTE 10.1.100.0/24              10.1.2.240 OSA2080L      MTU 1492
 ROUTE 10.1.100.0/24              10.1.2.240 OSA20A0L      MTU 1492
 ROUTE 10.1.100.0/24              10.1.3.240 OSA20C0L      MTU 1492
 ROUTE 10.1.100.0/24              10.1.3.240 OSA20E0L      MTU 1492
; Default Route - All packets to an unknown destination are routed
;through this route.
;     Destination     First Hop    Link Name   Packet Size
 ROUTE DEFAULT                     10.1.2.240   OSA2080L    MTU 1492
 ROUTE DEFAULT                     10.1.2.220   OSA2080L    MTU 1492
 ROUTE DEFAULT                     10.1.2.240   OSA20A0L    MTU 1492
 ROUTE DEFAULT                     10.1.2.220   OSA20A0L    MTU 1492
 ROUTE DEFAULT                     10.1.3.240   OSA20C0L    MTU 1492
 ROUTE DEFAULT                     10.1.3.220   OSA20C0L    MTU 1492
 ROUTE DEFAULT                     10.1.3.240   OSA20E0L    MTU 1492
 ROUTE DEFAULT                     10.1.3.220   OSA20E0L    MTU 1492
 ENDRoutes
;
AUTOLOG 5
   FTPDA  JOBNAME FTPDA1
;  OMPA
ENDAUTOLOG
;
PORT
    20 TCP OMVS      NOAUTOLOG     ; FTP Server  1
```

```
    514 UDP OMVS                   ; UNIX SyslogD Server   3
    21 TCP FTPDA1  BIND 10.1.1.10  ; control port
    23 TCP TN3270A                 ; Telnet Server
    25 TCP SMTP                    ; SMTP Server
   500 UDP IKED                    ; @ADI
;  520 UDP OMPA     NOAUTOLOG      ; OMPROUTE RIPV2 port
;  521 UDP OMPA     NOAUTOLOG      ; OMPROUTE RIPV2 port
  4500 UDP IKED                    ; @ADI
;
SACONFIG ENABLED COMMUNITY j0s9m2ap AGENT 161
;
START OSA2080
START OSA20C0
START OSA20E0
START OSA20A0
START IUTIQDF4
START IUTIQDF5
START IUTIQDF6
```

PROFILE.TCPIP (for OMPROUTE dynamic routing)

Example: C-10 PROFILE.TCPIP (for OMPROUTE dynamic routing)

```
;  This profile is for static routing
ARPAGE 20
;
GLOBALCONFIG NOTCPIPSTATISTICS
;
IPCONFIG DATAGRAMFWD SYSPLEXROUTING
DYNAMICXCF 10.1.7.11 255.255.255.0 1
;
SOMAXCONN 10
;
TCPCONFIG TCPSENDBFRSIZE 64K TCPRCVBUFRSIZE 64K SENDGARBAGE FALSE
TCPCONFIG TCPMAXRCVBUFRSIZE 256K
TCPCONFIG UNRESTRICTLOWPORTS
UDPCONFIG UNRESTRICTLOWPORTS
;
;OSA definitions
;TRL MAJ NODE: OSA2080,OSA20A0,OSA20C0,and OSA20E0
DEVICE OSA2080    MPCIPA
LINK   OSA2080L   IPAQENET    OSA2080 VLANID 10
DEVICE OSA20C0    MPCIPA
LINK   OSA20C0L   IPAQENET    OSA20C0 VLANID 11
DEVICE OSA20E0    MPCIPA
LINK   OSA20E0L   IPAQENET    OSA20E0 VLANID 11
DEVICE OSA20A0    MPCIPA
LINK   OSA20A0L   IPAQENET    OSA20A0 VLANID 10
;
;HiperSockets definitions
DEVICE IUTIQDF4   MPCIPA
LINK   IUTIQDF4L  IPAQIDIO    IUTIQDF4
DEVICE IUTIQDF5   MPCIPA
LINK   IUTIQDF5L  IPAQIDIO    IUTIQDF5
DEVICE IUTIQDF6   MPCIPA
LINK   IUTIQDF6L  IPAQIDIO    IUTIQDF6
;
;Static VIPA definitions
DEVICE VIPA1      VIRTUAL 0
LINK   VIPA1L     VIRTUAL 0   VIPA1
```

```
DEVICE VIPA2      VIRTUAL 0
LINK   VIPA2L     VIRTUAL 0   VIPA2
DEVICE VIPA3      VIRTUAL 0
LINK   VIPA3L     VIRTUAL 0   VIPA3
;
;Dynamic VIPA definitions
VIPADYNAMIC
 VIPADEFINE MOVEABLE IMMEDIATE 255.255.255.0 10.1.8.10
 VIPABACKUP 3 MOVEABLE IMMEDIATE 255.255.255.0 10.1.8.20
 VIPABACKUP 3 MOVEABLE IMMEDIATE 255.255.255.0 10.1.8.30
 VIPABACKUP 3 MOVEABLE IMMEDIATE 255.255.255.0 10.1.8.40
 VIPARANGE 255.255.255.0 10.1.9.0
ENDVIPADYNAMIC
;
HOME
   10.1.&SYSCLONE..10  VIPA3L
   10.1.1.10     VIPA1L
   10.1.2.10     VIPA2L
   10.1.2.11     OSA2080L
   10.1.3.11     OSA20C0L
   10.1.3.12     OSA20E0L
   10.1.2.12     OSA20A0L
   10.1.4.11     IUTIQDF4L
   10.1.5.11     IUTIQDF5L
   10.1.6.11     IUTIQDF6L
;
 PRIMARYINTERFACE VIPA1L
;
AUTOLOG 5
   FTPDA  JOBNAME FTPDA1
   OMPA
ENDAUTOLOG
;
PORT
    20 TCP OMVS      NOAUTOLOG     ; FTP Server  1
    514 UDP OMVS                   ; UNIX SyslogD Server   3
    21 TCP FTPDA1  BIND 10.1.1.10  ; control port
    23 TCP TN3270A                 ; Telnet Server
    25 TCP SMTP                    ; SMTP Server
   500 UDP IKED                    ; @ADI
   520 UDP OMPA     NOAUTOLOG      ; OMPROUTE RIPV2 port
   521 UDP OMPA     NOAUTOLOG      ; OMPROUTE RIPV2 port
  4500 UDP IKED                    ; @ADI
;
SACONFIG ENABLED COMMUNITY j0s9m2ap AGENT 161
;
START OSA2080
START OSA20C0
START OSA20E0
START OSA20A0
START IUTIQDF4
START IUTIQDF5
START IUTIQDF6
```

OMPROUTE dynamic routing

These are the complete examples used in our environment and discussed in Chapter 5, “Routing” on page 141.

OMPROUTE cataloged procedure

Example: C-11 OMPROUTE cataloged procedure

```
//OMPA30 PROC STDENV=OMPENA&SYSCLONE
//OMPA30 EXEC PGM=OMPROUTE,REGION=OM,TIME=NOLIMIT,
//        PARM=('POSIX(ON) ALL31(ON)',
//            'ENVAR("_BPXK_SETIBMOPT_TRANSPORT=TCPIPA"',
//            '"_CEE_ENVFILE=DD:STDENV")/')
//STDENV   DD DISP=SHR,DSN=TCPIP.SC&SYSCLONE..STDENV(&STDENV)
//SYSOUT   DD SYSOUT=*
//OMPCFG   DD DSN=TCPIPA.TCPPARMS(OMPA&SYSCLONE.),DISP=SHR
//CEEDUMP  DD SYSOUT=*,DCB=(RECFM=FB,LRECL=132,BLKSIZE=132)
```

OMPROUTE environment variables

Example: C-12 OMPROUTE environment variables

```
; ****************************************
; TCPIP.SC30.STDENV(OMPENA30)
; ****************************************
RESOLVER_CONFIG=//'TCPIPA.TCPPARMS(DATAA&SYSCLONE.)'
;OMPROUTE_FILE=//'TCPIPA.TCPPARMS(OMPA30)'
OMPROUTE_DEBUG_FILE=/etc/omproute/debug30a
OMPROUTE_DEBUG_FILE_CONTROL=100000,5
```

TCPIP.DATA file

Example: C-13 Global TCPIP.DATA file

```
; ****************************************
; TCPIPA.TCPPARMS(GLOBAL)
; ****************************************
DOMAINORIGIN  ITSO.IBM.COM
NSINTERADDR  10.12.6.7
NSPORTADDR 53
RESOLVEVIA UDP
RESOLVERTIMEOUT 10
RESOLVERUDPRETRIES 1
LOOKUP LOCAL DNS
```

Example: C-14 Local TCPIP.DATA file

```
; ****************************************
; TCPIPA.TCPPARMS(DATAA30)
; ****************************************
TCPIPJOBNAME TCPIPA
HOSTNAME WTSC30A
DATASETPREFIX TCPIPA
MESSAGECASE MIXED
```

Syslogd configuration file

Example: C-15 Syslogd configuration file

```
##********************************************************************
#*                                                                    *
#* syslog.conf - Defines the actions to be taken for the specified    *
#*               facilities/priorities by the syslogd daemon.         *
#*                                                                    *
*.OMPA*.*.err /tmp/syslog/ompa.err.log
```

OMPROUTE configuration file

Example: C-16 OMPROUTE configuration file

```
Area Area_Number=0.0.0.2
     Stub_Area=YES
     Authentication_type=None
     Import_Summaries=Yes;
OSPF
   RouterID=10.1.30.10
   Comparison=Type2
   Demand_Circuit=YES;
Global_Options
 Ignore_Undefined_Interfaces=YES
;
Routesa_Config Enabled=No;
; Static vipa
OSPF_Interface IP_address=10.1.30.10
          Name=VIPA3L
          Subnet_mask=255.255.255.0
          Attaches_To_Area=0.0.0.2
          Advertise_VIPA_Routes=HOST_ONLY
          Cost0=10
          MTU=65535;
; Static vipa
OSPF_interface ip_address=10.1.1.10
           name=VIPA1L
           subnet_mask=255.255.255.0
           Advertise_VIPA_Routes=HOST_ONLY
           attaches_to_area=0.0.0.2
           cost0=10
           mtu=65535
OSPF_interface ip_address=10.1.2.10
           name=VIPA2L
           subnet_mask=255.255.255.0
           Advertise_VIPA_Routes=HOST_ONLY
           attaches_to_area=0.0.0.2
           cost0=10
           mtu=65535
; OSA Qdio VLAN10
OSPF_Interface IP_address=10.1.2.*
           Subnet_mask=255.255.255.0
           Router_Priority=0
           Attaches_To_Area=0.0.0.2
           Cost0=100
           MTU=1492;
; OSA Qdio VLAN11
OSPF_Interface IP_address=10.1.3.*
           Subnet_mask=255.255.255.0
           Router_Priority=0
           Attaches_To_Area=0.0.0.2
           Cost0=100
           MTU=1492;
; Hipersockets 10.1.4.x
OSPF_Interface IP_address=10.1.4.*
           Subnet_mask=255.255.255.0
           Router_Priority=1
           Attaches_To_Area=0.0.0.2
           Cost0=90
           MTU=8192;
```

```
; Hipersockets 10.1.5.x
OSPF_Interface IP_address=10.1.5.*
           Subnet_mask=255.255.255.0
           Router_Priority=1
           Attaches_To_Area=0.0.0.2
           Cost0=90
           MTU=8192;
; Hipersockets 10.1.6.x
OSPF_Interface IP_address=10.1.6.*
           Subnet_mask=255.255.255.0
           Router_Priority=1
           Attaches_To_Area=0.0.0.2
           Cost0=90
           MTU=8192;
; Dynamic XCF (HiperSockets only)
ospf_interface ip_address=10.1.7.11
           name=IQDIOLNKOA01070B
           subnet_mask=255.255.255.0
           Router_Priority=1
           attaches_to_area=0.0.0.2
           cost0=90
           mtu=65535;
;
; Dynamic vipa VIPADEFINE
ospf_interface ip_address=10.1.8.*
           subnet_mask=255.255.255.0
           Advertise_VIPA_Routes=HOST_ONLY
           attaches_to_area=0.0.0.2
           cost0=10
           mtu=65535
;
; Dynamic vipa VIPADEFINE
ospf_interface ip_address=10.1.9.*
           subnet_mask=255.255.255.0
           Advertise_VIPA_Routes=HOST_ONLY
           attaches_to_area=0.0.0.2
           cost0=10
           mtu=65535
;
;AS_Boundary_routing
; Import_Direct_Routes=yes;
```

Router configuration

Example: C-17 Router A configuration

```
interface Loopback1
 ip address 10.1.200.1 255.255.255.0
!
interface Vlan10
 ip address 10.1.2.240 255.255.255.0
 ip ospf cost 100
 ip ospf priority 100
!
interface Vlan11
 ip address 10.1.3.240 255.255.255.0
 ip ospf cost 100
 ip ospf priority 100
!
router ospf 100
 router-id 10.1.3.240
```

```
 log-adjacency-changes
 area 2 stub no-summary
 network 10.1.2.0 0.0.0.255 area 2
 network 10.1.3.0 0.0.0.255 area 2
 network 10.1.100.0 0.0.0.255 area 2
 network 10.200.1.0 0.0.0.255 area 0
 default-information originate always metric-type 1
```

Example: C-18 Router B configuration

```
interface Loopback1
 ip address 10.1.200.2 255.255.255.0
!
interface Vlan10
 ip address 10.1.2.220 255.255.255.0
 ip ospf cost 100
 ip ospf priority 101
!
interface Vlan11
 ip address 10.1.3.220 255.255.255.0
 ip ospf cost 100
 ip ospf priority 101
!
router ospf 100
 router-id 10.1.3.220
 log-adjacency-changes
 area 2 stub no-summary
 network 10.1.2.0 0.0.0.255 area 2
 network 10.1.3.0 0.0.0.255 area 2
 network 10.1.200.0 0.0.0.255 area 0
 default-information originate always metric-type 1
```

D

Our implementation environment

We wrote the four *Communications Server for z/OS TCP/IP Implementation* publications at the same time. Given the complexity of this project, we needed to be somewhat creative in organizing the test environment so that each team could work with minimal coordination and interference from the other teams. In this appendix we show the complete environment used for the four books, and the environment used for this IBM Redbooks publication.

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 D-1.

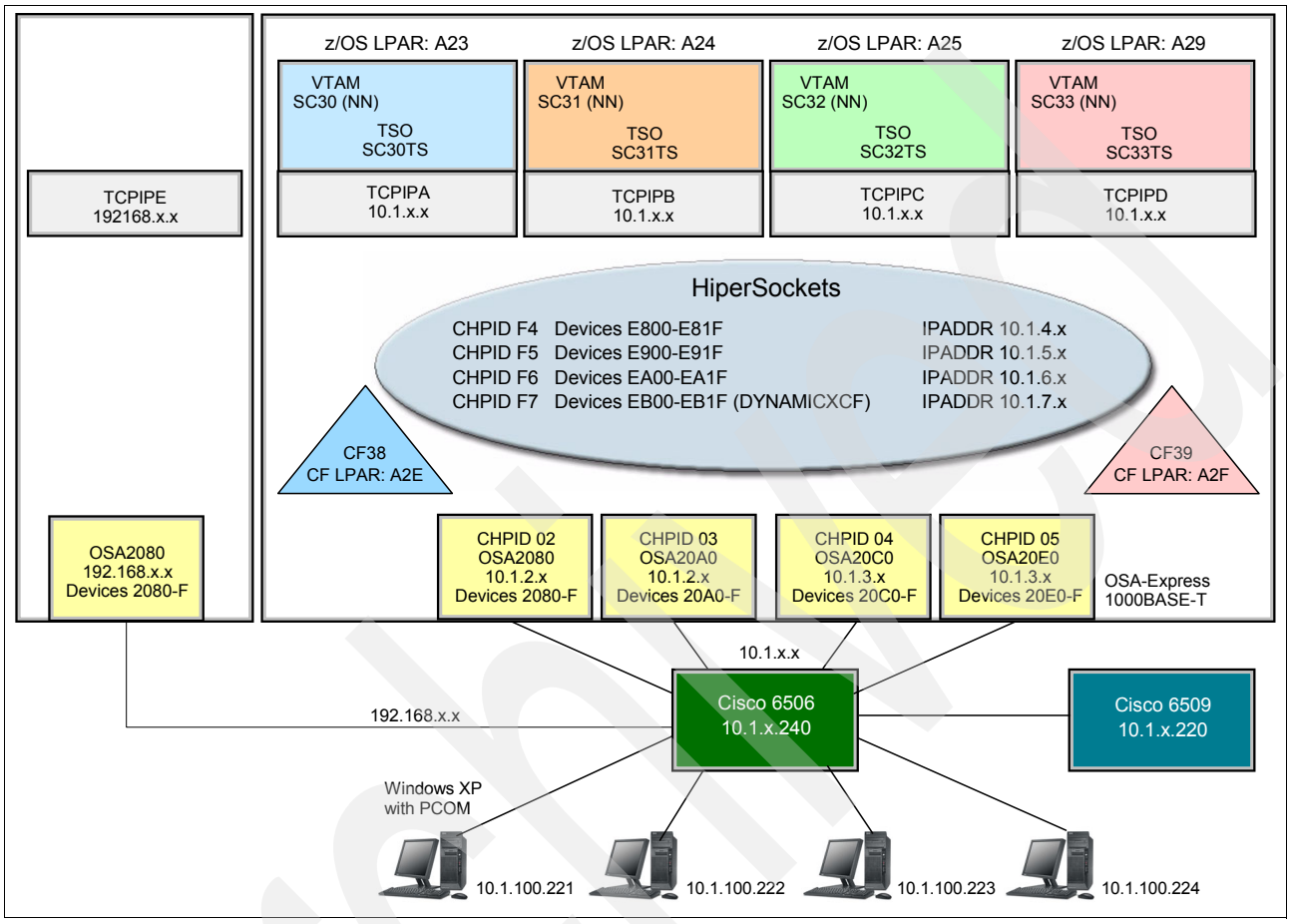


Figure D-1 Our implementation environment

Our books were written (and our implementation scenarios were run) using four logical partitions (LPARs) on an IBM System z9 EC (referred to as LPARs: A23, A24, A25, and A29). We implemented and started one TCP/IP stack on each LPAR. Each LPAR shared the following resources:

- HiperSockets inter-server 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 6500 switches

Note that this environment is based on the premise of high availability (no single points of failure).

Finally, we shared four Windows workstations, representing corporate network access to the z/OS networking environment. The workstations could be connected to either one of the Cisco switches. For verifying our scenarios, we used applications such as TN3270 and FTP.

The IP addressing scheme we used allowed us to build multiple subnetworks so that we would not impede ongoing activities from other team members.

VLANs were also defined to isolate the TCP/IP stacks and portions of the LAN environment (see Figure D-2).

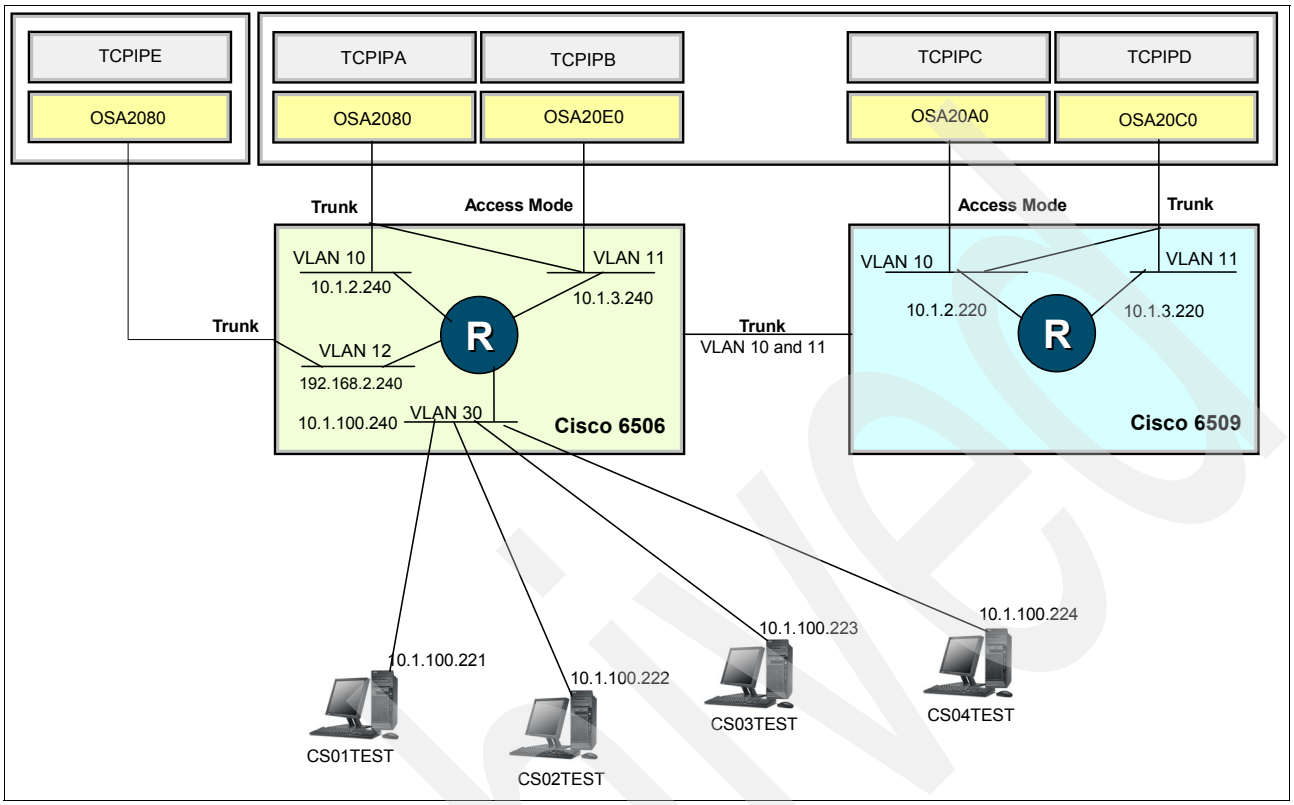


Figure D-2 LAN configuration - VLAN and IP addressing

Our focus for this book

Figure D-3 on page 318 depicts the environment that we worked with, as required for our basic function implementation scenarios.

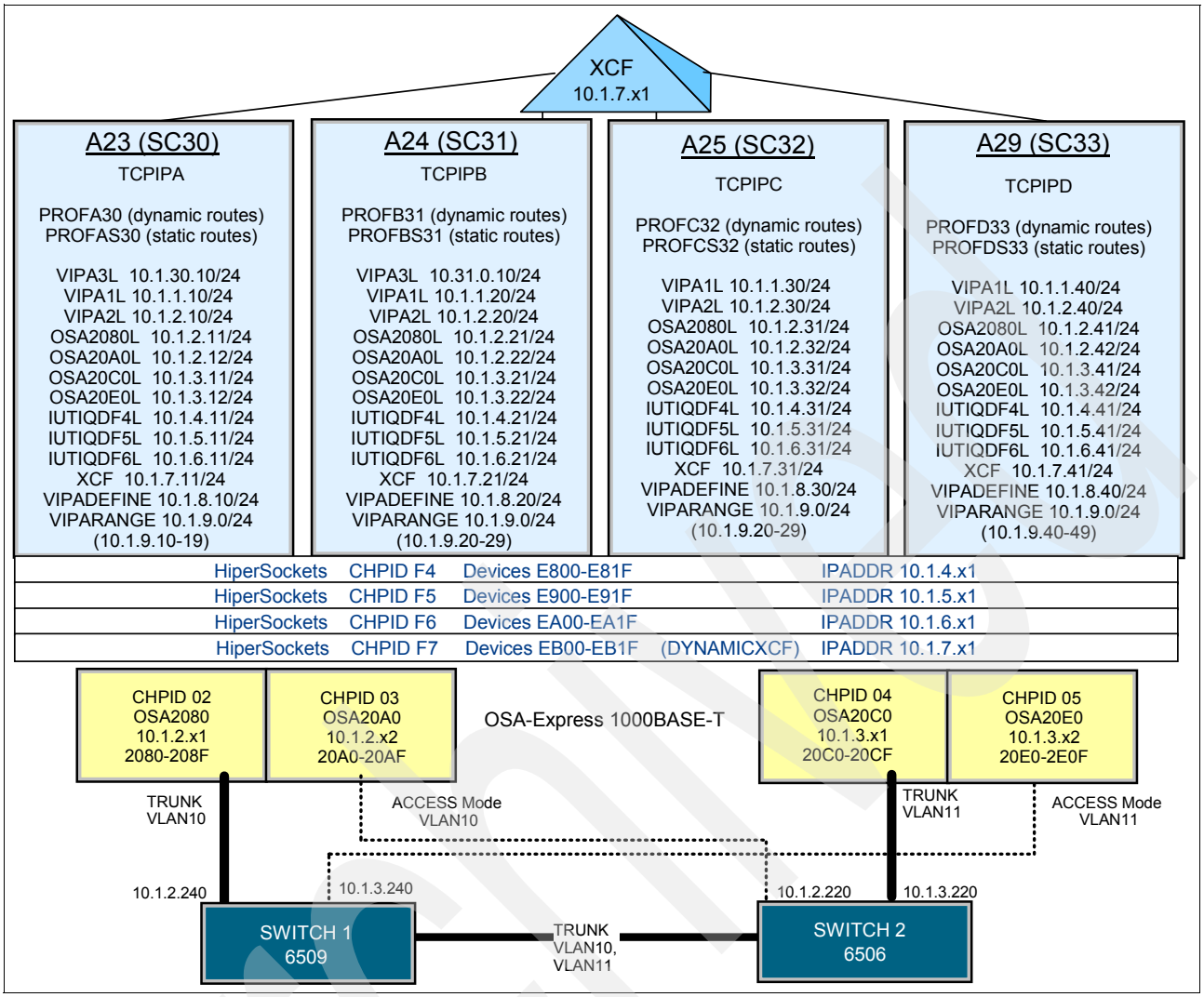


Figure D-3 Our environment for this book

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks publications

For information about ordering these publications, see “How to get IBM Redbooks publications” on page 321. Note that some of the documents referenced here may be available in softcopy only.

- *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 1: Base Functions, Connectivity, and Routing*, SG24-7532
- *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 2: Standard Applications*, SG24-7533
- *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 3: High Availability, Scalability, and Performance*, SG24-7534
- *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 4: Security and Policy-Based Networking*, SG24-7535
- *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
- *HiperSockets Implementation Guide*, SG24-6816
- *SNA in a Parallel Sysplex Environment,* SG24-2113
- *z/OS Infoprint Server Implementation*, SG24- 6234
- *z/OS Security Services Update,* SG24-6448
- *Deploying a Public Key Infrastructure*, SG24-5512

Other publications

These publications are also relevant as further information sources:

- *z/OS XL C/C++ Run-Time Library Reference*, SA22-7821
- *z/OS MVS IPCS Commands*, SA22-7594
- *z/OS MVS System Commands*, SA22-7627
- *z/OS TSO/E Command Reference*, SA22-7782
- *z/OS UNIX System Services Command Reference*, SA22-7802
- *z/OS Communications Server: CSM Guide*, SC31-8808
- *z/OS Communications Server: New Function Summary*, GC31-8771
- *z/OS Communications Server: Quick Reference*, SX75-0124

- *z/OS Communications Server: IP and SNA Codes*, SC31-8791
- *z/OS Communications Server: IP System Administrator's Commands*, SC31-8781
- *z/OS Communications Server: IP Diagnosis Guide*, GC31-8782
- *z/OS Communications Server: IP Configuration Guide*, SC31-8775
- *z/OS Communications Server: IP Messages Volume 1 (EZA)*, SC31-8783
- *z/OS Communications Server: IP Messages Volume 2 (EZB, EZD)*, SC31-8784
- *z/OS Communications Server: IP Messages Volume 3 (EZY)*, SC31-8785
- *z/OS Communications Server: IP Messages Volume 4 (EZZ, SNM)*, SC31-8786
- *z/OS Communications Server: IP Programmer's Guide and Reference*, SC31-8787
- *z/OS Communications Server: IP Configuration Reference*, SC31-8776
- *z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference*, SC31-8788
- *z/OS Communications Server: IP User's Guide and Commands*, SC31-8780
- *z/OS Communications Server: IPv6 Network and Application Design Guide*, SC31-8885
- *z/OS Migration*, GA22-7499
- *OSA-Express Customer's Guide and Reference*, SA22-7935
- *z/OS Communications Server: SNA Network Implementation*, SC31-8777
- *z/OS Communications Server: SNA Resource Definition*, SC31-8778
- *z/OS Communications Server: SNA Operation*, SC31-8779
- *z/OS TSO/E Command Reference*, SA22-7782
- *z/OS UNIX System Services Programming: Assembler Callable Services Reference*, SA22-7803
- *z/OS UNIX System Services Command Reference*, SA22-7802
- *z/OS UNIX System Services File System Interface Reference*, SA22-7808
- *z/OS UNIX System Services Messages and Codes*, SA22-7807
- *z/OS UNIX System Services Parallel Environment: Operation and Use*, SA22-7810
- *z/OS UNIX System Services Programming Tools*, SA22-7805
- *z/OS UNIX System Services Planning*, GA22-7800
- *z/OS UNIX System Services User's Guide*, SA22-7801
- *IBM Health Checker for z/OS: User's Guide,* SA22-7994

Online resources

These Web sites and URLs are also relevant as further information sources:

- z/OS Communications Server product support

  http://www-306.ibm.com/software/network/commserver/zos/support/

- 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/systems/z/os/zos/bkserv/r9pdf/#commserv

How to get IBM Redbooks publications

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

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Communications Server for z/OS V1R9 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

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. The series *Communications Server for z/OS TCP/IP Implementation* provides easy-to-understand, step-by-step guidance about enabling the most commonly used and important functions of CS for z/OS TCP/IP.

This IBM Redbooks publication, *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 1: Base Functions, Connectivity, and Routing*, SG24-7532, introduces CS for z/OS TCP/IP. Then the System Resolver, showing the implementation of global and local settings for single and multi-stack environments, is discussed. Finally, implementation scenarios for TCP/IP Base functions, Connectivity, Routing, Virtual MAC support, and sysplex subplexing are presented. For more specific information about CS for z/OS standard applications, high availability, and security, refer to the other volumes in the series:

- *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 2: Standard Applications*, SG24-7533
- *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 3: High Availability, Scalability, and Performance*, SG24-7534
- *Communications Server for z/OS V1R9 TCP/IP Implementation Volume 4: Security and Policy-Based Networking*, SG24-7535

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

SG24-7532-00 ISBN 0738485667